Videopuhelutoiminnallisuuden toteutus React Native Expo -ympäristössä WebRTC- ja Firebase-teknologioita hyödyntäen

Jami Äijänen

Haaga-Helia ammattikorkeakoulu
Tradenomi, Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
2024

Tiivistelmä

Tekijä(t)
Jami Äijänen

Tutkinto
Tradenomi, Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi
Videopuhelutoiminnallisuuden toteutus React Native Expo -ympäristössä WebRTC- ja Firebase-teknologioita hyödyntäen.

Sivu- ja liitesivumäärä
32

Tässä opinnäytetyössä tarkastellaan mobiilisovelluksen kehittämistä, joka sisältää reaaliaikaiset videopuhelutoiminnot WebRTC-tekniikalla ja Firebase Firestorella toteutetulla signalointiprosessilla. Projekti keskittyy erityisesti sovelluksen videopuhelutoiminnallisuuden toteuttamiseen ja optimointiin, joka mahdollistaa käyttäjien välisen reaaliaikaisen yhteydenpidon. Työn tavoitteena oli kehittää vakaasti toimiva ja käyttäjäystävällinen sovellus, joka hyödyntää modernia teknologiaa videopuheluiden ja viestinnän mahdollistamiseen mobiililaitteilla.

Projektissa hyödynnettiin WebRTC-teknologiaa videon ja äänen siirtoon, joka integroitiin React Native Expo -ympäristöön. Firebase Firestore toimi signalointiprosessissa, halliten puhelujen aloittamista, hyväksymistä ja lopettamista. Lisäksi projektissa käytettiin STUN- ja TURN-palvelimia yhteydenmuodostuksen tukemiseksi. Erityistä huomiota kiinnitettiin sovelluksen tietoturvaan ja yksityisyyteen, erityisesti käyttäjien tiedonsiirrossa ja videopuheluissa.

Työ käsittelee myös kehityksessä kohtattuja haasteita, kuten WebRTC:n toimivuutta Expo-ympäristössä, puhelujen hylkäämisestä johtuvia ongelmia ja käyttöliittymän optimointia, erityisesti modaalien vääristymistä näppäimistön avautuessa. Nämä ongelmat ratkaistiin muun muassa development buildin käytön ja Firebase-sovelluksen optimoinnin avulla.

Projektin lopputuloksena syntyi toimiva sovellus, joka täyttää käyttäjien videopuhelutarpeet ja tarjoaa käyttäjäystävällisen käyttöliittymän. Tämän työn myötä opittiin monia teknisiä taitoja, kuten WebRTC:n ja Firebase Firestoren syvällinen ymmärtäminen sekä projektinhallinnan ja ongelmanratkaisun taidot, jotka ovat keskeisiä tulevissa projekteissa.

Opinnäytetyö dokumentoi koko kehitysprosessin vaiheittain ja esittelee ratkaisut kohtaamiimme haasteisiin, sekä pohdinnan projektin onnistumisista, kehityskohteista ja jatkokehitysmahdollisuuksista.

Asiasanat
Mobiilisovellukset, Viestintä, Tietokannat

Sisällys

1 Johdanto

Monissa nykyaikaisissa mobiilisovelluksissa videopuhelutoiminnallisuus voi olla olennainen tai jopa välttämätön osa sovelluksen toimintaa, erityisesti kun käyttäjät tarvitsevat nopeita ja yksinkertaisia viestintämahdollisuuksia. Tässä projektissa tavoitteena on kehittää videopuheluominaisuus olemassa olevaan sovellukseen, jonka avulla käyttäjät voivat kommunikoida keskenään omassa ryhmässään. Sovellus on suunnattu tukemaan erityisesti vanhusten ja heidän läheistensä välistä viestintää. Sovelluksessa on jo ennestään käytössä Firebase, jota hyödynnetään mm. käyttäjien autentikointiin ja tietokantaratkaisuihin. Tämä videopuheluominaisuus lisää sovelluksen käyttöarvoa ja joustavuutta tarjoamalla turvallisen ja helppokäyttöisen viestintäkanavan.

Olemassa olevan sovelluksen perustoiminnallisuudet, kuten käyttäjien autentikointi, ryhmien hallinta ja yhteystietojen löytäminen, muodostavat videopuheluominaisuuden perustan. Käyttäjät voivat löytää toisensa omasta ryhmästään, joka on jo määritelty sovelluksessa, ja tämän olemassa olevan tiedon avulla voidaan helpottaa videopuheluiden aloittamista ilman, että käyttäjän tarvitsee itse syöttää tai etsiä erillistä koodia tai käyttäjänimeä.

Tässä opinnäytetyössä tavoitteena on rakentaa toimiva videopuheluominaisuus React Native Expo -ympäristössä, joka mahdollistaa käyttäjien reaaliaikaisen kommunikaation omassa ryhmässään. Tämä ominaisuus lisää sovelluksen käyttöarvoa ja tarjoaa käyttäjille nopean, turvallisen ja helposti käytettävän viestintäkanavan. Tärkeimmät käsitteet tässä työssä ovat WebRTC, React Native Expo ja Firebase. React Native Expo on suosittu kehitysympäristö mobiilisovelluksille, ja WebRTC (Web Real-Time Communication) on avoimeen standardiin perustuva teknologia, joka mahdollistaa reaaliaikaisen ääni- ja videoyhteyden laitteiden välillä. Firebase puolestaan tarjoaa yhteyteen soveltuvan tietokannan, joka tukee yhteyksien muodostamista ja ylläpitämistä.

Videopuhelutoiminnallisuuden toteutukseen valittiin WebRTC, sillä se tukee ääni-, video- ja datayhteyksiä ja mahdollistaa laitteiden väliset yhteydet ilman erillistä palvelinta. WebRTC on myös yhteensopiva kaikkien suurimpien alustojen kanssa, ja sen voi integroida React Native -ympäristössä Android- ja iOS-sovelluksiin. WebRTC on avoimen lähdekoodin projekti, jota tukevat monet teknologiayritykset, kuten Google, Apple, Microsoft ja Mozilla, tarjoaa myös merkittävän tietoturvaedun, koska viestintä on salattu. Tämä tekee siitä luotettavan valinnan mobiilisovellusten videopuheluratkaisuihin. Tavoitteena on, että sovellus tarjoaa käyttäjilleen luotettavan ja helppokäyttöisen videopuheluominaisuuden, jonka avulla viestintä sovelluksen sisällä onnistuu sujuvasti ja tehokkaasti. (WebRTC)

Projektin tavoitteena on, että sovellus tarjoaa käyttäjilleen luotettavan ja helppokäyttöisen videopuheluominaisuuden, jonka avulla viestintä sovelluksen sisällä onnistuu sujuvasti ja tehokkaasti.

Tämän kehitystyön lopputuloksena videopuheluominaisuuden tulisi tukea jo olemassa olevaa sovelluskokonaisuutta.

1.1 Tutkimusongelma ja tutkimuskysymykset

Työn keskeinen tutkimusongelma on selvittää, miten videopuhelutoiminnallisuus voidaan toteuttaa React Native Expo -ympäristössä hyödyntäen WebRTC-teknologiaa. Tavoitteena on rakentaa videopuheluominaisuus, joka toimii ongelmitta sovelluksessa ja mahdollistaa käyttäjien reaaliaikaisen kommunikoinnin omassa ryhmässään. Tämä on sovelluksen toiminnan kannalta keskeinen ominaisuus, sillä se tukee käyttäjien välisiä viestintätarpeita tehokkaasti ja joustavasti.

Projektissa pyritään vastaamaan kahteen pääkysymykseen:

1. Mitkä ovat keskeiset tekniset vaatimukset videopuheluominaisuuden toteuttamiselle React Native Expo -ympäristössä?

2. Miten voidaan varmistaa, että videopuheluominaisuus tarjoaa sujuvan ja positiivisen käyttäjäkokemuksen sovelluksen käyttäjille?

Näiden kysymysten kautta tutkimuksessa huomioidaan sekä tekniset että käyttäjäkokemukseen liittyvät vaatimukset. Teknisiin vaatimuksiin lukeutuvat esimerkiksi WebRTC:n integraation yhteensopivuus React Native Expo -ympäristössä, NAT-yhteyksien hallinta yhteyden vakauden takaamiseksi, Firebase-pohjaisen signaloinnin toteutus sekä push-notifikaatioiden luotettavuus ja toiminnallisuus. Käyttäjäkokemuksen osalta tarkastellaan muun muassa videopuheluiden laadun varmistamista (esimerkiksi äänen ja kuvan viiveettömyyttä) sekä käyttäjille selkeitä ja intuitiivisia tapoja aloittaa ja lopettaa puheluita.

1.2 Opinnäytetyön rakenne

Tämä opinnäytetyö koostuu neljästä pääosasta: Johdanto, Tietoperusta, Empiirinen osa ja Pohdinta.

Ensimmäisessä osassa, Johdanto, esitellään projektin tausta, tavoitteet ja tutkimusongelma. Johdanto luo pohjan työn tavoitteille ja selkeyttää tutkimuskysymykset, joihin työssä haetaan vastausta.

Tietoperusta-osiossa tarkastellaan videopuhelutoiminnallisuuden keskeisiä teknologioita ja niiden roolia reaaliaikaisen viestinnän toteutuksessa. Tämä osio käsittelee React Native ja Expo -ympäristöä, WebRTC-teknologiaa sekä Firebase-ympäristöä.

Empiirinen osa käsittelee projektin käytännön toteutusta. Siinä kuvataan sovelluksen suunnittelu- ja kehitysprosessi, WebRTC ja Firebase-tietokannan konfigurointi sekä videopuhelutoiminnallisuuden tekniset ratkaisut. Tässä osiossa tuodaan esille myös kehitysprosessin aikana kohdatut haasteet ja niihin löydetyt ratkaisut.

Lopuksi Pohdinta-osiossa arvioidaan projektin onnistumista ja pohditaan työn tuloksia sekä mahdollisuuksia jatkokehitykseen. Tässä osiossa tarkastellaan projektin vahvuuksia ja kehityskohteita sekä opittuja asioita, jotka voivat olla hyödyksi tulevaisuuden sovelluskehityksessä.

1.3 Työssä käytetyt käsitteet

React Native

- React Native on avoimen lähdekoodin kehityskehys, joka mahdollistaa mobiilisovellusten luomisen JavaScriptillä. Kehyksen avulla voidaan kehittää natiiveja iOS- ja Android-sovelluksia yhdellä koodikannalla.

Expo

- Expo on React Native -sovelluskehitykseen tarkoitettu työkalu ja kirjasto, joka helpottaa projektin aloittamista ja hallintaa. Expo tarjoaa valmiita moduuleja, kuten kameran, ilmoitukset ja paikannuspalvelut.

WebRTC (Web Real-Time Communication)

- WebRTC on avoimen lähdekoodin teknologia, joka mahdollistaa reaaliaikaisen viestinnän, kuten video- ja äänipuhelut, sekä tiedonsiirron suoraan laitteiden välillä ilman välipalvelimia.

STUN-palvelin (Session Traversal Utilities for NAT)

- STUN-palvelin on verkkoprotokolla, jota käytetään määrittämään laitteen julkinen IP-osoite ja portti NAT-verkon takaa. Se on tärkeä WebRTC-yhteyksien muodostamisessa.

TURN-palvelin (Traversal Using Relays around NAT)

- TURN-palvelin on verkkopalvelin, joka toimii tiedonsiirron välittäjänä, jos suoraa yhteyttä laitteiden välillä ei voida muodostaa esimerkiksi NAT- tai palomuurirajoitusten vuoksi.

NAT (Network Address Translation)

- NAT on verkkotekniikka, jonka avulla yksityisen verkon laitteet voivat käyttää yhteistä julkista IP-osoitetta. Tämä mahdollistaa IPv4-osoitteiden säästämisen, mutta voi aiheuttaa haasteita WebRTC-yhteyksissä.

Firebase

- Firebase on Googlen tarjoama pilvipohjainen palvelualusta, joka tukee sovellusten kehitystä tarjoamalla työkaluja, kuten autentikointi, tietokannat ja ilmoitukset.

Firestore

- Firestore on Firebase-alustan NoSQL-tietokanta, joka mahdollistaa datan reaaliaikaisen synkronoinnin sovellusten välillä. Firestorea käytetään WebRTC-sovelluksissa signalointiin, eli yhteydenmuodostukseen liittyvän tiedon vaihtoon.

Signalointi

- Signalointi on prosessi, jossa vaihdetaan tietoja, kuten IP-osoitteita, portteja ja yhteysasetuksia, ennen kuin WebRTC-yhteys voidaan muodostaa.

Peer-to-Peer-yhteys

- Peer-to-Peer (P2P) -yhteys tarkoittaa suoraa tiedonsiirtoa kahden laitteen välillä ilman välipalvelimia. Tämä vähentää viivettä ja tarjoaa paremman suorituskyvyn WebRTC-yhteyksissä.

Development build

- Development build on sovelluskehityksessä käytettävä testiversio, joka mahdollistaa natiivien ominaisuuksien, kuten WebRTC:n käytön. Expo Go -sovelluksessa ei voi käyttää WebRTC-teknologiaa, joten ratkaisun testaamiseen on luotava development build.

Push-ilmoitukset

- Push-ilmoitukset ovat viestejä, jotka lähetetään mobiililaitteeseen sovelluksen palvelimelta. Firebase tarjoaa työkalut push-ilmoitusten hallintaan.

ICE (Interactive Connectivity Establishment)

- ICE on protokolla, joka määrittää parhaimman reitin kahden laitteen välille WebRTC-yhteyden muodostamiseksi. Se yhdistää STUN- ja TURN-palvelimia optimaalisen yhteyden varmistamiseksi.

2 Tietoperusta

Videopuhelutoiminnallisuuden toteutus vaatii monipuolista teknologista ymmärrystä, erityisesti reaaliaikaisen viestinnän teknologioista, jotka mahdollistavat viiveettömän yhteyden käyttäjien välillä. Tämän opinnäytetyön tietoperusta käsittelee keskeisiä komponentteja ja teknologioita, joiden avulla videopuheluominaisuus voidaan rakentaa mobiilisovellukseen React Native Expo -kehitysympäristössä.

Ensiksi tarkastellaan projektin pohjana käytettyjä teknologioita, React Native ja Expo, jotka mahdollistavat tehokkaan ja alustariippumattoman mobiilisovelluskehityksen. Näiden kehitysympäristöjen ansiosta voidaan luoda käyttäjäystävällisiä sovelluksia, jotka toimivat sekä Android- että iOS-alustoilla ilman merkittävää lisätyötä.

Toinen tärkeä osa-alue on WebRTC-teknologia, jonka avulla reaaliaikainen viestintä ja datansiirto mahdollistetaan. WebRTC on erityisen tärkeä, koska se tukee suoraa ääni- ja videoyhteyttä laitteiden välillä, jolloin kolmannen osapuolen palvelimen tarvetta voidaan vähentää. WebRTC:n käyttö mobiilisovelluksissa vaatii myös perusteellista tietoa sen toimintaperiaatteista ja sen konfiguroinnista.

Lisäksi käsitellään Firebase-palvelua, jonka avulla videopuheluiden signalointi toteutetaan. Firebase tarjoaa pilvipohjaisen tietokannan ja backend-ratkaisut, jotka tukevat reaaliaikaisia päivityksiä ja skaalautuvuutta. Tämä mahdollistaa tiedon välittämisen yhteyttä muodostettaessa ja ylläpitäessä.

Tietoperustan seuraavissa osioissa tarkastellaan tarkemmin kutakin komponenttia ja niiden merkitystä projektin toteutuksessa. Näiden osioiden avulla luodaan perusta sille, miten reaaliaikaisen videopuhelutoiminnallisuuden vaatimukset täytetään ja millaisia ratkaisuja React Native Expo -ympäristössä voidaan hyödyntää.

2.1 React Native ja Expo

React Native ja Expo ovat nykyaikaisia ja suosittuja teknologioita mobiilisovellusten kehittämiseen. Niiden avulla voidaan luoda tehokkaita ja alustariippumattomia sovelluksia, jotka toimivat saumattomasti sekä Android- että iOS-alustoilla. Tässä osiossa tarkastellaan React Native -kehitysympäristön ja Expon merkitystä projektissa, niiden keskeisiä ominaisuuksia sekä sitä, miten ne tukevat videopuhelutoiminnallisuuden toteutusta.

2.1.1 React Native -kehitysympäristö

React Native on Metan kehittämä avoimen lähdekoodin sovelluskehys, jonka avulla voidaan luoda natiivisovelluksia käyttämällä JavaScript-koodia ja React-kirjastoa. React Native hyödyntää JavaScriptiä ja Reactin komponenttipohjaista rakennetta, jolloin sovelluksen käyttöliittymä voidaan määritellä joustavasti ja suorituskykyisesti. Tämä mahdollistaa sovellusten rakentamisen yhdellä koodipohjalla, mikä vähentää työmäärää verrattuna siihen, että sovellus koodattaisiin erikseen Androidille sekä iOS:lle. (Shoutem, 2016)

React Native -kehitysympäristön suurimpiin etuihin kuuluu sen kyky tarjota natiivitasoinen käyttökokemus yhdistettynä mahdollisuuteen hyödyntää verkkoteknologioita ja kirjastoja. Tämä tekee siitä hyvin soveltuvan videopuheluominaisuuden kaltaisten reaaliaikaisten toimintojen toteutukseen, sillä React Native tarjoaa monipuoliset yhteydet natiivitoimintoihin, kuten laitteen kameraan ja mikrofoniin. Lisäksi React Native tarjoaa mahdollisuuden integroida kolmannen osapuolen kirjastoja, kuten WebRTC, joka on tässä projektissa olennainen reaaliaikaisen viestinnän mahdollistamiseksi. (Expo Docs, Use libraries)

2.1.2 Expo-kehitysalusta

Expo on kehitysalusta ja työkalu, joka rakentuu React Native -kehyksen päälle ja tekee React Native -kehittämisestä helppokäyttöisempää ja nopeampaa erityisesti aloittelijoille. Expo tarjoaa valmiin työkalupaketin, jossa on useita hyödyllisiä toiminnallisuuksia, kuten automaattinen päivitys, sisäänrakennettu kehittäjän työkalut ja käyttöliittymäelementit, joita ei tarvitse konfiguroida erikseen. Tämä säästää kehitysaikaa ja auttaa keskittymään sovelluksen varsinaisiin toiminnallisuuksiin. (Ömür Bilgili, 2023)

Expo tarjoaa kehittäjille myös mahdollisuuden testata sovellusta reaaliajassa Expon omalla mobiilisovelluksella, mikä helpottaa kehitystä ja nopeuttaa sovelluksen iterointia. Lisäksi Expo on yhteensopiva sekä Android- että iOS-laitteiden kanssa, mikä mahdollistaa molemmilla alustoilla tapahtuvan kehityksen yhdellä koodipohjalla. Expon helppokäyttöisyys ja valmiit työkalut tekevät siitä erityisen sopivan ympäristön nopeasti alkuun pääsemiseen ja toteutuksen rakentamiseen, mikä tukee projektin tavoitteita. (Expo docs, Develop an app with Expo)

2.1.3 React Native ja Expo videopuhelutoiminnallisuuden tukena

React Native ja Expo tarjoavat kattavan ympäristön videopuhelutoiminnallisuuden toteuttamiseen, sillä ne mahdollistavat pääsyn natiivilaitteiden toimintoihin, kuten kameraan ja mikrofoniin, sekä tarjoavat tukea WebRTC-kirjaston kaltaisille ulkoisille laajennuksille. Tämä tekee niistä erittäin sopivan ratkaisun reaaliaikaisen viestinnän toteuttamiseen mobiilisovelluksessa. React Native tarjoaa

hyvän yhteensopivuuden WebRTC-teknologian kanssa, kun taas Expo nopeuttaa kehitystä ja tarjoaa laajan tuen alustariippumattomalle kehitykselle. Näin ne yhdessä tukevat tehokasta ja toimivaa videopuhelutoiminnallisuutta, joka toimii sekä Android- että iOS-laitteilla. (Mathew Pregasen, 2023)

2.2 WebRTC-teknologia

WebRTC (Web Real-Time Communication) mahdollistaa reaaliaikaisen ääni- ja videoviestinnän laitteiden välillä suoraan internet-yhteyden kautta ilman erillisten palvelimien tarvetta. WebRTC on kehitetty erityisesti verkko- ja mobiilisovelluksia varten, jotka vaativat nopeaa ja sujuvaa viestintää, kuten videopuhelut ja ääniyhteydet. Teknologia on integroitu modernien selainten ja mobiilisovellusalustojen kanssa, mikä tekee siitä monikäyttöisen ja laajasti tuetun vaihtoehdon reaaliaikaisille yhteyksille. WebRTC kehitystä tukevat useat suuret teknologiayritykset, kuten Google, Apple, Microsoft ja Mozilla, mikä takaa sen luotettavuuden ja jatkuvan kehittymisen. (WebRTC)

2.2.1 WebRTC perusteet

WebRTC-arkkitehtuuri koostuu kolmesta keskeisestä osa-alueesta: media capture (mediasisällön tallennus), signaling (signalointi) ja peer-to-peer connection (laitteiden välinen yhteys). Näiden osa-alueiden avulla WebRTC mahdollistaa kahden käyttäjän laitteiden välisen suoran yhteyden videopuheluihin, ääniyhteyksiin ja muuhun reaaliaikaiseen tiedonsiirtoon.

1. **Media Capture:** WebRTC voi käyttää laitteen kameraa, mikrofonia ja muita mediatoimintoja MediaStream API avulla. Tämä vaihe on tärkeä, sillä se mahdollistaa videon ja äänen siirron laitteiden välillä.
2. **Signaling:** WebRTC-yhteyden muodostaminen vaatii signalointivaiheen, jossa laitteet löytävät toisensa ja vaihtavat yhteyden vaatimia asetustietoja. Signaloinnissa jaetaan muun muassa yhteysosoitteet ja konfiguraatiotiedot. Tämä prosessi ei kuulu varsinaisesti WebRTC-standardiin, ja se voidaan toteuttaa esimerkiksi Firebasen tai muun erillisen palvelimen avulla.
3. **Peer-to-Peer Connection:** Kun signalointi on valmis, laitteet voivat muodostaa suoran yhteyden toisiinsa. WebRTC hyödyntää STUN- ja TURN-palvelimia mahdollisten verkkorajoitusten, kuten NAT-verkon, kiertämiseksi. Tämän yhteyden kautta siirretään reaaliaikainen data, mikä mahdollistaa viiveettömän ääni- ja videosiirron. (Mary E. Shacklett)

WebRTC:n päätarkoitus video- ja ääniyhteyksien rakentamisessa on mahdollistaa nopea, suora ja turvallinen yhteys kahden laitteen välillä ilman, että dataa joudutaan reitittämään erillisten palvelimien kautta. Tämä tarjoaa optimaalisen suorituskyvyn ja vähentää viivettä, minkä vuoksi WebRTC on suosittu ratkaisu sovelluksissa, joissa tarvitaan välitöntä viestintää ja saumattomia yhteyksiä.

Mobiiliverkoissa laitteet toimivat usein verkko-olosuhteissa, joissa niillä ei ole suoraa yhteyttä ulkoverkkoon. Tällaiset rajoitteet johtuvat tyypillisesti verkkoinfrastruktuurista ja osoitteiden hallintaan liittyvistä käytännöistä, joiden tavoitteena on ylläpitää tietoturvaa ja tehokasta resurssien käyttöä. Nämä verkko-olosuhteet voivat kuitenkin luoda haasteita WebRTC-yhteyksien muodostamiselle, sillä suorat yhteydet laitteiden välillä eivät ole aina mahdollisia ilman erityisiä ratkaisuja, kuten STUN- ja TURN-palvelimia.

Tämän ongelman ratkaisemiseksi WebRTC käyttää STUN- ja TURN-palvelimia:

- STUN-palvelimet auttavat laitteita löytämään julkisen IP-osoitteensa ja porttinsa NAT-verkon takaa, mahdollistaen suoran yhteyden muodostamisen, jos NAT-tyyppi sen sallii.
- TURN-palvelimet toimivat välityspalvelimina tilanteissa, joissa NAT-verkon tai palomuurien asettamat rajoitukset estävät suoran yhteyden. TURN-palvelimien käyttö lisää hieman viivettä, mutta ne takaavat yhteyden vakauden haastavissa verkko-olosuhteissa.

NAT-verkkojen ja mobiiliverkkojen tietoturvaan liittyy myös laajempi näkökulma. NAT-verkon takana olevat laitteet ovat suojassa suoralta ulkoiselta liikenteeltä, mutta tämä rajoittaa myös yhteyksien muodostamista, ellei siihen käytetä erityisiä ratkaisuja, kuten edellä mainittuja STUN- ja TURN-palvelimia. Tämä tietoturva-aspekti on yksi syy siihen, miksi omaa palvelinta voidaan tarvita WebRTC-yhteyksien hallintaan, erityisesti jos sovelluksen käyttäjäkunta toimii laajasti eri verkkoympäristöissä. (Stream)

Näiden käytännön näkökohtien ymmärtäminen on tärkeää, kun WebRTC:ta sovelletaan mobiilisovelluksissa. Mobiiliverkot asettavat erityisvaatimuksia yhteyksien vakaudelle ja suorituskyvylle, ja näiden haasteiden huomioiminen on keskeistä onnistuneen videopuheluratkaisun kehittämisessä.

2.2.2 Signalointi ja yhteyden muodostaminen

WebRTC-yhteyden muodostamiseksi kahden laitteen välillä tarvitaan signalointivaihe, jonka aikana laitteet voivat vaihtaa yhteyden perustamiseen tarvittavia tietoja. Vaikka signalointi on keskeinen osa WebRTC-yhteyden muodostamisprosessia, se ei ole osa itse WebRTC-rajapintaa. Sen sijaan signalointi voidaan toteuttaa eri tavoin, kuten WebSocketin, HTTP POST -pyyntöjen tai muiden reaaliaikaisten viestintäprotokollien avulla. Signaloinnin avulla laitteet pystyvät löytämään toisensa, sopimaan yhteyden teknisistä yksityiskohdista ja varmistamaan, että yhteyden muodostaminen on mahdollista molempiin suuntiin.

Signalointiprosessi kattaa seuraavat vaiheet:

1. **SDP (Session Description Protocol) -tiedonvaihto:** SDP-kuvauksen avulla laitteet voivat vaihtaa tietoa, kuten mediamuodon (esim. ääni tai video) ja verkkoasetukset. Tämä vaihe varmistaa, että molemmat laitteet voivat käyttää samoja yhteysasetuksia. (What is SDP - Session Description Protocol?)
2. **ICE (Interactive Connectivity Establishment) -kandidaattien vaihto:** ICE-protokollan avulla laitteet etsivät parhaat mahdolliset reitit yhteyden muodostamiseksi. Prosessissa käytetään STUN- ja TURN-palvelimia, joiden avulla yhteys voidaan muodostaa myös silloin, kun laitteet ovat NAT-verkon takana tai muissa haastavissa verkkoympäristöissä. STUN-palvelimet auttavat laitteita löytämään omat julkiset IP-osoitteensa, kun taas TURN-palvelimet voivat välittää dataa laitteiden välillä silloin, kun suora yhteys ei ole mahdollinen. (Interactive Connectivity Establishment (ICE))

Kun signalointi on valmis ja laitteet ovat vaihtaneet tarvittavat SDP- ja ICE-tiedot, ne voivat muodostaa suoran peer-to-peer-yhteyden. Tämä yhteys mahdollistaa reaaliaikaisen ääni-, video- ja datayhteyden siirron laitteiden välillä, mikä on WebRTC pääasiallinen toiminnallinen tavoite. Signaloinnin ansiosta WebRTC-yhteydet ovat joustavia ja toimivat erilaisissa verkkoympäristöissä, mikä tekee teknologiasta erittäin monikäyttöisen ja luotettavan ratkaisun reaaliaikaiseen viestintään. (Digital Samba, 2023)

2.2.3 STUN- ja TURN-palvelimet

STUN- ja TURN-palvelimet ovat keskeisiä osia WebRTC-yhteyksien muodostamisessa, erityisesti monimutkaisissa verkkoympäristöissä, joissa laitteet saattavat olla NAT-verkkojen tai palomuurien takana. Näiden palvelinten avulla laitteet voivat löytää julkiset IP-osoitteensa ja määrittää sopivimman tavan viestiä keskenään.

1. **STUN-palvelin**: STUN (Session Traversal Utilities for NAT) -protokolla on suunniteltu auttamaan laitteita, jotka sijaitsevat yksityisessä verkossa, tunnistamaan oman julkisen IP-osoitteensa ja NAT-tyyppinsä. Tämä on tärkeää, koska NAT (Network Address Translation) -verkot yleensä piilottavat laitteiden sisäverkon osoitteet ulkoverkoilta, mikä voi estää laitteita muodostamasta suoria yhteyksiä. STUN-palvelin mahdollistaa laitteiden julkisten IP-osoitteiden ja porttien selvittämisen, jolloin ne voivat kiertää NAT-asetusten asettamat rajoitukset ja muodostaa suoran yhteyden toisiinsa. STUN-palvelin on yksinkertainen ja tehokas ratkaisu, joka toimii suurimmassa osassa tavallisista verkkoympäristöistä.

2. **TURN-palvelin**: Joskus STUN ei riitä, erityisesti silloin, kun NAT-asetukset ovat tiukat tai palomuurit rajoittavat suoria yhteyksiä. Tässä tapauksessa TURN (Traversal Using Relays around NAT) -palvelin tulee avuksi. TURN-palvelin toimii välityspalvelimena, joka ohjaa

liikennettä laitteiden välillä silloin, kun suoraa yhteyttä ei voida muodostaa. Tämä mahdollistaa yhteyden muodostamisen haastavimmissakin verkkoympäristöissä. TURN-palvelimen käyttö saattaa lisätä viivettä, mutta se tarjoaa luotettavan viestintäkanavan silloin, kun muut ratkaisut eivät toimi. (Stream)

Yksi yleisesti käytetty Googlen STUN-palvelimen osoite on stun:stun.l.google.com:19302. Googlen STUN-palvelimien käyttö on kätevää, mutta jos sovelluksessa vaaditaan erittäin luotettava yhteys haastavissa verkko-olosuhteissa, myös TURN-palvelinta saatetaan tarvita, ja tällöin kehittäjän kannattaa harkita oman TURN-palvelimen käyttöönottoa. (Multilogin)

2.3 Firebase ja Firestore

Firebase on Googlen kehittämä kattava alusta mobiili- ja verkkosovellusten backend-palveluille. Se tarjoaa useita palveluita, jotka helpottavat kehittäjiä toteuttamaan monipuolisia ja skaalautuvia toimintoja ilman, että tarvitsee rakentaa ja ylläpitää omaa palvelinympäristöä. Firebase tarjoaa joukon työkaluja, jotka kattavat käyttäjähallinnan, tietokannan, analytiikan, push-ilmoitukset ja muita kehitystoimintoja, jotka tukevat reaaliaikaista viestintää sekä nopeaa sovelluksen kehitystä.

Firestore on yksi Firebase-alustan tietokantaratkaisuista, joka on suunniteltu erityisesti skaalautumaan suurten käyttäjämäärien ja monimutkaisten tietorakenteiden kanssa. Firestore tukee reaaliaikaisia tietokantayhteyksiä ja tarjoaa mahdollisuuden synkronoida tietoja eri laitteiden välillä. Tämän ansiosta se soveltuu erinomaisesti mobiilisovelluksiin, joissa tiedon tulee päivittyä välittömästi kaikille käyttäjille.

2.3.1 Firebase-perusteet

Firebase toimii backend-alustana, joka soveltuu hyvin mobiili- ja verkkosovellusten kehittäjille. Sen avulla kehittäjät voivat lisätä sovelluksiinsa valmiiksi integroituja ratkaisuja ilman, että heidän tarvitsee käsitellä omia palvelinympäristöjä tai monimutkaisia verkkoinfrastruktuureja. Firebase tarjoaa sovellukselle useita olennaisia komponentteja:

1. **Käyttäjähallinta:** Firebase Authentication -työkalun avulla kehittäjät voivat helposti toteuttaa kirjautumis- ja rekisteröitymisominaisuuksia tukien useita yleisiä kirjautumistapoja, kuten sähköpostia, puhelinnumeroa ja sosiaalisen median tunnuksia.
2. **Tietokannat:** Firebase tarjoaa kaksi päätietokantaa, Realtime Database ja Firestore, jotka ovat molemmat optimoitu reaaliaikaiseen tiedon synkronointiin mobiililaitteiden välillä. Firestore on suunniteltu skaalautuvaksi ja tietomalliltaan joustavaksi, minkä ansiosta se soveltuu erityisesti suuriin ja dynaamisiin sovelluksiin.

3. **Ilmoitukset:** Firebase Cloud Messaging (FCM) tarjoaa tavan lähettää push-ilmoituksia sovelluksen käyttäjille nopeasti ja helposti. Tämä mahdollistaa monipuoliset viestintäominaisuudet ja käyttäjäviestinnän ylläpidon sovelluksessa.
4. **Analytiikka ja seuranta:** Firebase Analytics antaa kehittäjille mahdollisuuden seurata sovelluksen käyttöä ja analysoida käyttäjäpolkuja, jolloin sovelluksen toimintaa voidaan kehittää käyttökokemuksen parantamiseksi.

Firebase on kokonaisuutena toimiva ja skaalautuva ratkaisu, erityisesti sovelluksille, jotka vaativat reaaliaikaista tiedonsiirtoa ja joustavaa tietokantakapasiteettia. Alustan avulla kehittäjät voivat keskittyä sovelluksen ydintoimintoihin ja käyttäjäkokemukseen ilman tarvetta rakentaa ja hallita omaa serveripohjaista backend-ympäristöä. (Firebase)

2.3.2 Firestore signalointiprosessissa

Firestore on tehokas valinta WebRTC-signalointiprosessiin sen joustavien ominaisuuksien ansiosta. Firestore tarjoaa keskeisiä ominaisuuksia, jotka tukevat tehokasta tiedonvaihtoa ja yhteyksien hallintaa reaaliajassa, mikä on ratkaisevaa WebRTC-yhteyksien muodostamisessa. Näiden ominaisuuksien ansiosta signalointiprosessi voidaan toteuttaa luotettavasti ja suorituskykyisesti.

Firestoren soveltuvuutta WebRTC-signalointiprosessiin tukevat seuraavat keskeiset ominaisuudet:

1. **Joustava tietokanta**: Firestore tukee joustavia tietorakenteita. Tiedot voidaan tallentaa asiakirjoina, jotka sijoitetaan kokoelmiin, ja näiden asiakirjojen sisällä voidaan hyödyntää monimutkaisia sekä sisäkkäisiä tietorakenteita.

2. **Monipuoliset kyselyt**: Firestoressa voidaan suorittaa kyselyitä, jotka palauttavat yksittäisiä asiakirjoja tai kokoelman dokumentit, jotka vastaavat annettuja kyselyparametreja. Signalointiprosessissa tämä mahdollistaa tiettyjen yhteystietojen, kuten SDP- ja ICE-kandidaattien, tehokkaan hakemisen ja hallinnan.

3. **Reaaliaikaiset päivitykset**: Firestore synkronoi tiedot reaaliajassa kaikille yhteydessä oleville laitteille, mikä tekee siitä ihanteellisen välineen WebRTC-signalointiin. Kun yksi osapuoli tallentaa yhteystietoja Firestoreen, toinen osapuoli voi vastaanottaa nämä päivitykset lähes välittömästi ja käynnistää yhteydenmuodostusprosessin.

4. **Skaalautuvuus**: Firestore hyödyntää Google Cloudin infrastruktuuria tarjoten monialueisen tiedon replikaation ja vahvan johdonmukaisuuden. Tämä mahdollistaa laajojen ja monimutkaisten tietomäärien hallinnan ilman suorituskyvyn heikkenemistä, mikä on tärkeää suuria käyttäjämääriä palvelevissa sovelluksissa.

Näiden ominaisuuksien ansiosta Firestore tarjoaa WebRTC-signalointiprosessiin sopivan backend:in, joka tukee yhteydenmuodostuksen vaatimaa nopeaa ja luotettavaa tiedonsiirtoa sekä skaalautuvuutta. (Cloud Firestore)

3 Empiirinen osa

3.1 Projektin suunnittelu

Projektin suunnittelu on ratkaiseva vaihe onnistuneen ohjelmistokehityksen kannalta, erityisesti monimutkaisia toiminnallisuuksia, kuten videopuhelutoiminnon sisältävissä projekteissa. Tämä vaihe määrittää projektin tavoitteet, käyttötapaukset ja vaatimukset, jotka yhdessä luovat selkeän polun kohti tavoiteltua lopputulosta. Suunnittelun pyritään varmistamaan, että teknologiat ja valinnat tukevat projektissa rakennettavia toiminnallisuuksia ja että jokainen komponentti pystytään lisäämään ongelmitta osaksi kokonaisuutta.

Tässä opinnäytetyössä projektin suunnittelu painottuu erityisesti siihen, miten videopuhelutoiminnallisuus tulisi kehittää. Tämä edellyttää tarkkaa käyttövaatimusten määrittelyä sekä tehokasta signalointi- ja viestintäratkaisua, jotta reaaliaikainen videopuhelu onnistuu luotettavasti. Projektin suunnitteluvaiheessa huomioidaan myös käytettävät teknologiat ja sovellusympäristö, kuten Firebase ja WebRTC, jotka mahdollistavat toimivan ja luotettavan toteutuksen.

Seuraavassa käsitellään tarkemmin projektin keskeisiä käyttötapauksia ja vaatimuksia, joiden avulla videopuhelutoiminnallisuus pyritään tuomaan käytäntöön osaksi rakennettua sovellusta.

3.1.1 Käyttötapaukset ja vaatimukset

Videopuhelutoiminnallisuuden suunnittelu mobiilisovellukseen edellyttää selkeitä käyttötapauksia sekä teknisiä ja toiminnallisia vaatimuksia, jotta kehitys vastaa sekä käyttäjien tarpeita että sovelluksen reunaehtoja. Tärkein käyttötapaus tässä projektissa on mahdollistaa reaaliaikainen videopuheluyhteys ryhmän jäsenten välillä, mikä parantaa sovelluksen käyttökokemusta ja on tärkeä osa sovelluksen toimintaa.

Käyttötapaukset

- **Videopuhelun käynnistäminen:** Käyttäjä voi aloittaa videopuhelun valitsemalla toisen käyttäjän ryhmästään, jolloin sovellus asettaa yhteyden automaattisesti WebRTC avulla. Puhelun tulisi alkaa mahdollisimman nopeasti, ja yhteydenmuodostuksen tulee olla käyttäjälle sujuva ja selkeä.
- **Puheluun vastaaminen ja lopettaminen:** Kun kutsu saapuu, käyttäjälle näytetään ilmoitus saapuvasta puhelusta. Käyttäjä voi vastata tai hylätä puhelun yksinkertaisilla toiminnoilla. Jos käyttäjä vastaa, WebRTC-yhteys muodostuu; hylätessä kutsu katkaistaan välittömästi.
- **Reaaliaikainen video- ja ääniyhteys:** Puhelun aikana sovelluksen on mahdollistettava reaaliaikainen, selkeä ja sujuva video- ja äänilähetys molempiin

suuntiin. Reaaliaikaisen viestinnän tulisi toimia myös vaihtelevissa verkko-olosuhteissa mahdollisimman luotettavasti.

- **Puhelun päättäminen:** Jommankumman käyttäjän lopettaessa puhelun yhteys katkaistaan, ja molemmat käyttäjät palaavat takaisin sovelluksen etusivulle.

Vaatimukset

Järjestelmän vaatimukset jakautuvat kolmeen pääkategoriaan: toiminnallisiin, teknisiin ja suorituskykyvaatimuksiin. Toiminnallisten vaatimusten osalta järjestelmän tulee mahdollistaa reaaliaikainen äänen ja videon siirto WebRTC:n avulla, jotta videopuhelutoiminnallisuus toimii luotettavasti. Signalointiprosessissa käytettävä Firebase Firestore tukee yhteyden muodostamista ja hallintaa käyttäjien välillä, mahdollistaen käyttäjille puhelun aloittamisen, hyväksymisen ja lopettamisen yhdellä painalluksella. Tavoitteena on lisäksi, että videopuheluominaisuus integroituu saumattomasti React Native Expo -ympäristöön, jolloin se sulautuu osaksi sovelluksen kokonaisuutta ilman käyttäjältä vaadittavia konfigurointi tai työvaiheita.

Teknisellä tasolla vaatimuksiin kuuluu WebRTC:n käyttö ääni- ja videopuheluiden toteuttamiseksi, mikä edellyttää yhteensopivuutta Expo-ympäristön kanssa. Tämä varmistaa sen, että sovellus voi käyttää kameran ja mikrofonin rajapintoja sekä muodostaa luotettavia videopuheluyhteyksiä. Firebase Firestore toimii järjestelmän signalointikanavana, jonka avulla hallitaan puheluiden kutsut ja vastaukset reaaliaikaisesti. Firestore takaa välittömät tietokantaoperaatiot, jotka ovat keskeisiä puhelun aloittamisessa ja päättämisessä. Lisäksi STUN-palvelimet tukevat yhteydenmuodostusta, ja TURN-palvelin takaa yhteyden muodostumisen myös silloin, kun NAT-ongelmat estävät suoran yhteyden.

Suorituskykyvaatimukset korostavat videopuheluyhteyden reaaliaikaisuutta ja vakautta. Viive video- ja äänilähetyksessä on pidettävä mahdollisimman pienenä, jotta puhelu toimii riittävän nopeasti käyttäjien tarpeisiin. Käyttöliittymältä vaaditaan yksinkertaisuutta, jotta käyttäjäkokemus säilyy hyvänä puhelun aikana ilman sovelluksen kaatumisia tai merkittäviä häiriöitä.

Nämä vaatimukset muodostavat perustan järjestelmän kehitykselle, ja niiden tarkoituksena on tukea toimintavarman ja käyttäjäystävällisen videopuhelutoiminnon toteutusta.

3.1.2 Tietoturva ja yksityisyys

Tietoturva ja yksityisyys ovat keskeisiä näkökohtia videopuhelutoiminnallisuuden suunnittelussa ja toteutuksessa, erityisesti kun kyseessä on reaaliaikainen viestintä käyttäjien välillä. WebRTC- ja Firebase-pohjaisen ratkaisun kohdalla tietoturvan varmistaminen edellyttää eri tasoilla toimivia

suojausmenetelmiä. WebRTC käyttää oletuksena salattuja yhteyksiä (SRTP, Secure Real-time Transport Protocol), mikä varmistaa, että ääni- ja videodataa ei voida siepata tai muuttaa matkan aikana. Tämä salaus estää kolmansia osapuolia pääsemästä käsiksi käyttäjien väliseen viestintään, suojaten siten käyttäjien yksityisyyttä. (What is SRTP - Secure Real-Time Transport Protocol?)

Järjestelmässä toteutetaan myös käyttäjän tunnistautuminen Firebase Authenticationin avulla. Tämä autentikointijärjestelmä mahdollistaa sen, että vain vahvistetut ja tunnistetut käyttäjät voivat kirjautua sovellukseen tai aloittaa sekä osallistua videopuheluihin. Lisäksi käyttäjien data on rajattu vain sovelluksen sisäiseen käyttöön, eikä sitä käytetä ilman käyttäjän suostumusta muihin tarkoituksiin.

Kaiken kaikkiaan tietoturva ja yksityisyys on luotu tiiviisti osaksi sovelluksen rakenteellista suunnittelua ja tietoliikenteen hallintaa, jolloin käyttäjien viestintä pysyy turvallisena ja yksityisyys suojattuna. Näin taataan sekä luottamuksellisuus että turvallinen käyttökokemus koko sovelluksessa.

3.2 Sovelluksen rakenne ja komponentit

Tässä kappaleessa käydään läpi sovelluksen rakenteelliset osat ja tärkeimmät komponentit, jotka muodostavat sovelluksen ytimen ja toiminnallisuuden. Painopiste on videopuhelutoiminnallisuuteen liittyvissä komponenteissa, jotka vastaavat käyttäjien välisestä reaaliaikaisesta yhteydestä ja viestinnästä.

3.2.1 Käyttöliittymä ja navigointi

Sovelluksen käyttöliittymän rakenne ja navigointilogiikka on suunniteltu yksinkertaiseksi ja intuitiiviseksi, jotta käyttäjät voivat vaivattomasti siirtyä eri toimintojen välillä. Pääasialliset käyttöliittymäkomponentit, kuten MainPage.js, CallScreen.js ja JoinScreen.js, muodostavat videopuhelun keskeisen käyttöliittymän, jossa käyttäjä voi aloittaa, liittyä tai lopettaa puhelun.

- MainPage.js toimii applikaation pääsivuna, jossa on painike videopuhelun aloittamiseksi.
- CallScreen.js toimii videopuhelun pääsivuna, jossa videoyhteys muodostetaan. Komponentti hallinnoi puhelun aikana tarvittavia toimintoja, kuten ääni- ja videovirran kytkemistä päälle tai pois sekä puhelun lopettamista. Näiden toimintojen avulla käyttäjä voi hallita puhelun kulkua.
- JoinScreen.js on liittymisnäyttö, jossa käyttäjä voi liittyä saatuun puhelukutsuun. Tämä komponentti hallitsee puhelun hyväksyntää ja valmistautuu yhteyden muodostamiseen Firestore-signaloinnin kautta. Näytön rakenne

varmistaa, että käyttäjä voi tehdä päätöksen puheluun liittymisestä yhdellä napautuksella.

Navigointilogiikka on rakennettu siten, että käyttäjä voi siirtyä sujuvasti eri näyttöjen välillä sovelluksen sisällä. Tämä mahdollistaa hyvän kokemuksen myös niissä tilanteissa, joissa käyttäjä saa notifikaation puhelun alkamisesta ja voi halutessaan liittyä. Näin ollen käyttöliittymä tukee reaaliaikaista kommunikointia ja tarjoaa käyttäjälle hallinnan ja selkeyden tunteen koko puhelukokemuksen ajan.

3.2.2 WebRTC implementointi

WebRTC implementointi alkaa asennuksella ja tarvittavien komponenttien konfiguroinnilla, jotka mahdollistavat reaaliaikaisen ääni- ja videoyhteyden muodostamisen käyttäjien välille. Tämä prosessi koostuu useista vaiheista, jotka varmistavat yhteyden luotettavan ja turvallisen toiminnan mobiilisovelluksessa.

Ensimmäinen vaihe WebRTC:n käyttöönotossa on tarvittavien kirjastojen asentaminen, erityisesti react-native-webrtc, joka tarjoaa WebRTC ydintoiminnot, kuten äänen ja videon siirron. Koska React Native Expo ei suoraan tue WebRTC:ä, on tärkeää konfiguroida ympäristö niin, että tarvittavat rajapinnat, kuten kameran ja mikrofonin käyttö sekä WebRTC-yhteydet, toimivat odotetusti. Tämä vaatii usein myös natiivia koodia tai erillisiä moduuleja, jotka liittävät WebRTC:n React Native Expo-sovellukseen. WebRTC ei toimi Expo Go -sovelluksessa, koska Expo Go on yleiskäyttöinen sovellus, joka ei sisällä kaikkia projektikohtaisesti lisättyjä natiivimoduuleja, kuten react-native-webrtc. Tämän vuoksi WebRTC:n asennuksen jälkeen sovellusta on kehitettävä "development build"-prosessin kautta, jossa natiivikoodin muutokset sisällytetään sovellukseen. (React-native-webrtc)

Signalointiprosessi on keskeinen osa WebRTC:n toteutusta, ja tässä projektissa signalointi hoidetaan Firebase Firestore -tietokannan avulla. Firestore toimii signalointipalvelimena, jonka kautta käyttäjät voivat vaihtaa yhteyden muodostamiseen tarvittavat tiedot, kuten SDP-tarjoukset (Session Description Protocol) ja ICE-ehdokkaat (Interactive Connectivity Establishment). Signalointivaihe alkaa, kun käyttäjä aloittaa puhelun ja lähettää SDP-tarjouksen Firestoreen. Vastaanottaja lukee tämän tarjouksen ja vastaa siihen, lähettämällä SDP-vastauksen takaisin. Tämä prosessi mahdollistaa kahden laitteen välisen yhteyden muodostamisen ja varmistaa, että kaikki tarvittavat tiedot on vaihdettu ennen yhteyden aloittamista. (Signaling and video calling)

WebRTC:n toimivuuden varmistamiseksi tarvitaan myös STUN- ja TURN-palvelimia, jotka tukevat yhteydenmuodostusta erityisesti silloin, kun NAT (Network Address Translation) tai palomuurit estävät suoran yhteyden. STUN-palvelin auttaa määrittämään julkisen IP-osoitteen, mikä

mahdollistaa suoran yhteyden muodostamisen, kun taas TURN-palvelin tulee käyttöön silloin, kun suora yhteys epäonnistuu. WebRTC-yhteyden konfigurointi sisältääkin näiden palvelimien määrittämisen yhteysasetuksiin. Kuvasta 1 löytyy esimerkki STUN-palvelimen määrittelemisestä.

```
const configuration = {
  iceServers: [
    {
      urls: ["stun:stun1.l.google.com:19302", "stun:stun2.l.google.com:19302"],
    },
  ],
  iceCandidatePoolSize: 10,
};
```

Kuva 1. Palvelimien määritteleminen

Kun signalointi ja ICE-palvelimet on konfiguroitu, seuraava vaihe on luoda RTCPeerConnection-objekti, joka hallitsee yhteyden luomista ja sen tilapäivityksiä. Tämä objekti määrittää, kuinka mediavirrat, kuten video ja ääni, siirretään laitteiden välillä. Yhteyden muodostaminen alkaa luomalla RTCPeerConnection ja liittämällä siihen paikalliset mediatiedot, kuten kamera ja mikrofoni. Kuvasta 2 löytyy esimerkki RTCPeerConnectionin luonnista sekä mediatietojen välityksestä.

```
const localPC = new RTCPeerConnection(configuration);
setCachedLocalPC(localPC);

localStream.getTracks().forEach((track) => {
  localPC.addTrack(track, localStream);
});

localPC.onicecandidate = (e) => {
  if (e.candidate) {
    addDoc(collection(roomRef, "callerCandidates"), e.candidate.toJSON()).catch((err) => {
      console.error("Failed to add ICE candidate: ", err);
    });
  }
};

localPC.ontrack = (e) => {
  const newStream = new MediaStream();
  e.streams[0].getTracks().forEach((track) => newStream.addTrack(track));
  setRemoteStream(newStream);
};
```

Kuva 2. PeerConnectionin luonti

Koodiesimerkissä kuvassa 2 toteutetaan WebRTC-yhteyden luominen käyttäen RTCPeerConnection-rajapintaa. Ensimmäisessä vaiheessa luodaan uusi RTCPeerConnection-objekti, joka saa

parametrikseen configuration-objektin. Tämä konfiguraatio sisältää STUN- ja TURN-palvelimien tiedot, joita käytetään yhteyden muodostamisessa. RTCPeerConnection-objekti tallennetaan sovelluksen muistiin kutsumalla setCachedLocalPC-funktiota, jotta sitä voidaan hyödyntää myöhemmin muissa sovelluksen osissa.

Seuraavaksi lisätään lokaali mediavirta WebRTC-yhteyteen. Tämä tapahtuu hakemalla lokaalin mediavirran kaikki raidat (esimerkiksi video- ja audiovirrat) localStream.getTracks-metodin avulla. Jokainen raita lisätään RTCPeerConnection-objektiin addTrack-metodilla. Tämä mahdollistaa sen, että nämä mediavirrat voidaan jakaa etäosapuolelle.

RTCPeerConnection generoi myös ICE-kandidaatteja, jotka ovat verkkoyhteysvaihtoehtoja yhteyden muodostamiseksi NAT-verkon tai palomuurin takaa. Jokainen ICE-kandidaatti käsitellään onicecandidate-tapahtumakuuntelijassa, jossa se muunnetaan JSON-muotoon e.candidate.toJSON()-metodin avulla ja tallennetaan Firestore-tietokantaan addDoc-funktion kautta. Kandidaatit tallennetaan "callerCandidates"-kokoelmaan käyttäen huonetta vastaavaa roomRef-viitettä. Tallennusvaiheessa mahdolliset virheet käsitellään catch-kohdassa.

Kun etäosapuoli lähettää oman mediavirtansa, se vastaanotetaan ontrack-tapahtumakuuntelijan kautta. Tällöin luodaan uusi MediaStream-objekti, johon lisätään kaikki etäosapuolen lähettämät mediavirran raidat. Tämä uusi mediavirta asetetaan sovelluksessa käyttöön setRemoteStream-funktiolla, mikä mahdollistaa esimerkiksi etäosapuolen videon näyttämisen käyttöliittymässä.

Kokonaisuutena tämä koodi huolehtii WebRTC-yhteyden keskeisistä osista: lokaalin mediavirran lisäämisestä, yhteyden muodostamiseen tarvittavien ICE-kandidaattien hallinnasta sekä etäosapuolen lähettämän mediavirran käsittelystä ja näyttämisestä. Tämä muodostaa perustan reaaliaikaiselle kommunikaatiolle käyttäjien välillä.

Yhteyden aikana on tärkeää seurata yhteyden tilaa ja reagoida mahdollisiin virheisiin tai yhteyskatkoksiin. WebRTC mahdollistaa ICE-yhteyden tilan tarkkailun, ja sovelluksessa voidaan lisätä logiikkaa, joka käsittelee virhetilanteet, kuten yhteyskatkokset. Esimerkiksi, jos yhteys katkeaa, voidaan yrittää uudelleenyhdistämistä tai palauttaa käyttäjä takaisin sovelluksen pääsivulle.

Näiden vaiheiden avulla WebRTC asennus ja konfigurointi mahdollistavat reaaliaikaisen ääni- ja videopuhelutoiminnon integroimisen mobiilisovellukseen. Tämä tarjoaa käyttäjille saumattoman ja luotettavan kokemuksen yhteyksien muodostamisessa ja niiden hallinnassa.

3.2.3 Signalointiprosessi ja yhteydenhallinta

Signalointiprosessi on tärkeä osa WebRTC-pohjaisia videopuhelutoimintoja, sillä se mahdollistaa käyttäjien välisten yhteyksien muodostamisen ja ylläpidon. Tässä projektissa signalointiprosessi on toteutettu Firebase Firestore -tietokannan avulla, joka toimii reaaliaikaisena tiedonvaihtokanavana puheluiden hallintaan. Firestore tarjoaa joustavan ja tehokkaan tavan hallita puheluiden kutsuja, hyväksyntöjä ja muita yhteydenmuodostuksen vaiheita reaaliaikaisesti.

Kun käyttäjä haluaa aloittaa videopuhelun, hän lähettää puhelukutsun Firestoreen, jossa se tallennetaan ja on sen jälkeen saatavilla vastaanottajalle. Kutsun mukana lähetetään tarvittavat tiedot yhteyden muodostamiseksi, kuten SDP (Session Description Protocol) -tarjous, joka sisältää yhteyden muodostamiseen tarvittavat tiedot. Kun vastaanottaja saa kutsun, hän lukee sen Firestoresta ja vastaa siihen, lähettämällä takaisin SDP-vastauksen ja ICE-tiedot, jotka auttavat varmistamaan suoran yhteyden.

Kun yhteys on onnistuneesti muodostettu, WebRTC RTCPeerConnection-objekti vastaa yhteyden ylläpidosta, ja medialiikenteestä, kuten äänen ja videon siirtymisestä laitteiden välillä. Tämä yhteys kuitenkin voi katketa tai jäädä kesken, jos toinen käyttäjä päättää hylätä puhelun. Puhelun hylkääminen luo haasteita, sillä se voi jättää toisen käyttäjän odottamaan yhteyttä tai aiheuttaa virhetilanteita sovelluksessa, erityisesti jos sovellus ei käsittele puhelun hylkäämistä oikein.

Puhelun hylkäämisen tilanteessa on tärkeää, että molemmat käyttäjät saavat tiedon puhelun päättymisestä ja että sovellus sulkee yhteyden turvallisesti. Jos puhelun hylkääminen ei ole oikein käsitelty, voi seurauksena olla tilanne, jossa toinen käyttäjä jää "jumiin" odottamaan yhteyttä, tai sovellus saattaa kaatua tai jäädä odottamaan signalointia, jota ei koskaan tule. Tällöin sovelluksessa tulee olla kunnollinen virheenkäsittelymekanismi, joka tunnistaa tilanteet, joissa yhteys on keskeytynyt ja reagoi niihin asianmukaisesti, esimerkiksi näyttämällä käyttäjälle ilmoituksen yhteyden katkeamisesta.

Firestore-pohjainen signalointiprosessi ei pelkästään hallitse puheluiden aloituksia ja hyväksyntöjä, vaan se myös päivittää puhelun tilan reaaliaikaisesti kaikille osapuolille. Tämä takaa sen, että kaikki käyttäjät ovat aina tietoisia puhelun tilasta, ja että tilan muutokset, kuten puhelun hylkääminen, tulevat selvästi esiin. Esimerkiksi, kun toinen käyttäjä päättää puhelun, Firestore päivittää puhelun tilan "päätetty"-tilaksi, ja molemmat osapuolet saavat tästä tiedon lähes välittömästi. Kuvasta 3 löytyy koodista esimerkki puhelun päättymisestä.

```
const unsubscribe = onSnapshot(roomRef, (doc) => {
  const data = doc.data();
  if (!data) {
    console.log("Room document is missing or was deleted. Ending call.");
    unsubscribe();
    endCall();
    return;
  }

  if (data.status === "ended") {
    endCall();
    unsubscribe();
  }
```

Kuva 3. Puhelun päättäminen

Koodiesimerkissä kuva 3 hallitaan WebRTC-puhelun elinkaarta seuraamalla Firestore-tietokannan muutoksia reaaliajassa. Tämä toteutetaan käyttämällä Firestoren onSnapshot-metodia, joka kuuntelee tietokantadokumenttiin liittyviä muutoksia. onSnapshot-metodi ottaa kaksi argumenttia: viitteen dokumenttiin (roomRef) ja palautteena toimivan funktion, joka suoritetaan aina, kun dokumentin tila muuttuu.

Ensimmäiseksi dokumentin tiedot haetaan käyttämällä doc.data()-metodia, ja nämä tiedot tallennetaan data-muuttujaan. Jos dokumentin dataa ei löydy (esimerkiksi dokumentti on poistettu), suoritetaan komento, joka keskeyttää puhelun. Tällöin kutsutaan unsubscribe-funktiota lopettamaan kuuntelu, ja suoritetaan endCall-funktio puhelun päättämiseksi. Tämän jälkeen funktiosta poistutaan return-komennolla.

Seuraavaksi tarkistetaan, onko dokumentin status-kentän arvo "ended". Jos näin on, suoritetaan myös endCall-funktio puhelun lopettamiseksi. Tämän jälkeen kutsutaan uudelleen unsubscribe-funktiota, jotta Firestoren kuuntelu voidaan lopettaa, koska puhelu on jo päättynyt.

Tämä koodi varmistaa, että WebRTC-puhelun tila synkronoidaan Firestore-tietokannan kanssa. Se huolehtii siitä, että puhelu päätetään oikein, jos dokumentti poistetaan tai sen tila muutetaan päättyneeksi. Tämä mahdollistaa sen, että molemmat osapuolet saavat ajantasaisen tiedon puhelun tilasta, mikä on erityisen tärkeää reaaliaikaisessa viestinnässä.

Tämän signalointiprosessin ja yhteydenhallinnan avulla sovelluksen käyttäjät voivat kokea sujuvan ja luotettavan videopuhelukokemuksen, jossa puhelut voidaan aloittaa, hallita ja lopettaa ilman merkittäviä viiveitä tai häiriöitä, ja jossa virhetilanteet, kuten puhelun hylkääminen, käsitellään tehokkaasti ja turvallisesti.

3.2.4 Käyttäjävuorovaikutus ja siirtymät

Käyttäjävuorovaikutus on osa videopuhelutoiminnallisuutta, sillä se määrittää, kuinka käyttäjät voivat ohjata ja hallita puheluja sovelluksessa. Tämän projektin videopuhelutoiminnallisuudessa käyttäjävuorovaikutus on pyritty tekemään mahdollisimman sujuvaksi. Käyttäjälle tarjotaan selkeät painikkeet puhelun aloittamiseen, vastaanottamiseen, keskeyttämiseen ja lopettamiseen, ja koko käyttöliittymän suunnittelussa on otettu huomioon käyttäjäystävällisyys ja esteettömyys.

Sovelluksen tärkeimmät vuorovaikutuselementit, kuten puhelun aloitus- ja lopetusnappulat, sekä kameran ja mikrofonin hallinta, on integroitu React Native Expo -ympäristöön siten, että ne reagoivat käyttäjän toimiin nopeasti ja selkeästi. Puhelun aikana käyttäjä voi hallita videon ja äänen asetuksia, esimerkiksi mykistää mikrofonin tai vaihtaa kameran, käyttäen yksinkertaisia painikkeita. Käytettävyys ja navigoinnin selkeys on varmistettu myös responsiivisella suunnittelulla, joka mukautuu eri laitteiden näyttökokoihin ja orientaatioihin.

Siirtymien ja vuorovaikutusten sujuvuus videopuheluissa on kuitenkin haastavaa erityisesti verkon vaihtelevien olosuhteiden ja puhelun hallinnan osalta. Yksi merkittävä haaste on ollut puhelujen keskeyttäminen ja niiden palauttaminen. Esimerkiksi tilanteet, joissa puhelut keskeytyvät verkkoongelmien tai käyttäjän toimenpiteiden, kuten puhelun hylkäämisen vuoksi, voivat aiheuttaa sekavuutta.

Toinen haaste on videopuheluiden siirtymätilanteet, kuten puhelun alkamisen yhteydessä videon siirtäminen asettelu näytölle & sekä selkeä näkymä vastaanotetusta videosta ja omasta lähetettävästä videosta (esim. puhelun aikana tapahtuva siirtyminen päänäytöltä mininäkymään). Näiden siirtymien hallinta on tärkeää, jotta puhelukokemus ei keskeydy, ja käyttäjä voi jatkaa keskustelua ilman häiriöitä. Näihin haasteisiin on pyritty vastaamaan mukautetuilla siirtymillä, jotka suorittavat nämä siirtymät automaattisesti.

Yksi keskeinen vuorovaikutuksen osa-alue on myös videopuhelun tilan jatkuva päivitys. Puhelun aikana on tärkeää, että käyttäjä saa tietoa muun käyttäjän tilasta (esimerkiksi, onko hän hyväksynyt puhelun tai onko yhteys katkennut). Tämä reaaliaikainen vuorovaikutus voidaan toteuttaa Firestore-pohjaisella signalointiprosessilla, joka päivittyy jatkuvasti, mutta se asettaa vaatimuksia myös käyttöliittymälle, jotta tilan muutokset voidaan esittää käyttäjälle selkeästi ja nopeasti.

Käyttäjävuorovaikutuksen suunnittelussa ja siirtymillä on suuri merkitys videopuhelun sujuvuuden ja luotettavuuden kannalta. Vaikka sovelluksessa on onnistuttu luomaan intuitiivinen ja käyttäjäystävällinen käyttöliittymä, haasteita tulee edelleen etenkin verkon tilan vaihteluiden, puhelun hylkäämisten ja yhteyksien katkeamisen osalta. Näihin ongelmiin on kuitenkin löydetty ratkaisuja, jotka tekevät videopuhelukokemuksesta mahdollisimman vakaan ja käyttäjäystävällisen.

3.2.5 Push-notifikaatiot

Sovelluksen push-notifikaatioiden logiikka on suunniteltu erityisesti tukemaan ryhmätoiminnallisuutta, kuten videopuheluiden aloittamista. Notifikaatiot toimivat siten, että jokaisen ryhmän jäsenen push-notifikaatio tieto tallennetaan Firebase-tietokantaan, ja niiden avulla viestit voidaan lähettää kaikille ryhmän käyttäjille tehokkaasti ja luotettavasti.

ExpoPushTokenien hallinta

Jokaisella käyttäjällä on yksilöllinen expoPushToken, joka generoidaan sovelluksen käyttöönoton yhteydessä. Tämä arvo tallennetaan Firebase-tietokantaan yhdessä käyttäjän muiden tietojen, kuten sähköpostiosoitteen, kanssa. Tällä tavoin sovellus pystyy kohdentamaan notifikaatiot tarkasti oikeille vastaanottajille.

Notifikaatioiden luonti puheluiden yhteydessä

Kun käyttäjä aloittaa ryhmäpuhelun, sovellus luo automaattisesti uuden notifikaation jokaiselle ryhmän jäsenelle. Notifikaatio sisältää seuraavat tiedot:

- **callGroupCode**: Tunniste, joka liittyy aloitettuun puheluun. Tämän avulla pystytään suodattamaan käyttäjät kenelle notifikaatio tulee lähettää.
- **message**: Viesti, joka ilmoittaa käyttäjille puhelun alkamisesta. Esimerkkinä: "Jami Äijänen has started a call. Join now!"
- **timestamp**: Ajankohta, jolloin notifikaatio luotiin.
- **type**: Notifikaation tyyppi, joka tässä tapauksessa on "call".

Notifikaatiot tallennetaan Firebase-tietokantaan käyttäjän notifications-hakemistoon. Tämä rakenne mahdollistaa notifikaatioiden tehokkaan käsittelyn ja näyttää reaaliaikaisesti käyttäjälle heidän vastaanottamansa ilmoitukset. Kuvassa 4 esimerkkikuva tietokantaan tallennetusta notifikaatiosta.

Kuva 4. Tietokantaan tallennettu puhelu notifikaatio

Notifikaatioiden lähetys ja vastaanotto

Kun notifikaatio luodaan tietokantaan, sovellus käyttää expo-notifications-kirjastoa lähettääkseen viestin jokaiselle käyttäjälle, jonka expoPushToken on tallennettuna. Tämä viesti näkyy käyttäjän mobiililaitteen ilmoituspaneelissa ja kutsuu heidät liittymään aloitettuun puheluun.

Tämä toteutus hyödyntää Firebase-tietokantaa ja Expo-kirjastoja notifikaatioiden tallentamiseen, lähettämiseen ja hallintaan. Ratkaisu mahdollistaa nopeat ja luotettavat ryhmän sisäiset ilmoitukset, erityisesti videopuheluiden aloituksen yhteydessä, tarjoten käyttäjille hyvän käyttökokemuksen.

3.3 Kehityksen aikana kohdatut haasteet ja ratkaisut

Sovelluskehityksen aikana kohdattiin useita teknisiä ja toiminnallisia haasteita, jotka vaativat perusteellista ongelmanratkaisua ja menetelmien hienosäätöä. Näistä haasteista merkittävimpiä olivat Firebase-kirjastojen yhteensopivuusongelmat, reaaliaikaisen yhteyden vakauden varmistaminen heikossa verkkoympäristössä sekä käyttäjävuorovaikutuksen optimointi videopuheluissa. Tässä luvussa käsitellään näitä ongelmia ja niiden ratkaisuja yksityiskohtaisesti, aloittaen Firebase-luokkavirheiden ratkaisusta.

3.3.1 WebRTC:n yhteensopivuusongelmat Expo-ympäristössä

Projektin alkuvaiheessa yksi merkittävimmistä haasteista oli saada WebRTC toimimaan Expo-ympäristössä. Expo on suosittu kehitystyökalu React Native -sovelluksille, mutta sen valmiit ominaisuudet eivät aluksi tukeneet suoraan WebRTC:n käyttöä, joka on keskeinen osa videopuhelutoiminnallisuutta. Tämä johtui Expon rajoituksista tietyissä alhaisen tason natiivimoduuleissa, joita WebRTC vaatii toimiakseen, kuten pääsy kameran ja mikrofonin laitteistoon sekä suorien verkkoyhteyksien hallinta.

Ensimmäisessä yrityksessä käyttää WebRTC:tä Expon oletusympäristössä törmättiin jatkuviin virheisiin ja yhteensopivuusongelmiin. Ongelmat liittyivät muun muassa puuttuvaan tukeen natiivimoduuleille ja siihen, että WebRTC:n vaatimat kirjastot eivät toimineet suoraan Expo ympäristössä. Tämä esti projektia etenemästä, sillä videopuheluiden keskeistä osaa ei saatu testattua tai toteutettua.

Ratkaisua haettiin siirtymällä Expon ”Development Build” -työskentelymalliin. Tämä malli mahdollisti natiivimoduulien, kuten WebRTC:n vaatiman ”react-native-webrtc”-kirjaston, integroimisen projektiin. Development Build eroaa Expon Managed Workflow -tilasta siinä, että se tarjoaa kehittäjälle mahdollisuuden mukauttaa sovelluksen rakennetta ja lisätä natiivikirjastoja, joita ei ole oletuksena saatavilla Expon ympäristössä. (Create a development build)

Development Build -ympäristön käyttöönotto vaati muutoksia sovelluksen kehitysmalliin. Aluksi projektin ”app.json”-tiedostoon lisättiin WebRTC:n vaatimat konfiguraatiot, ja ”expo-dev-client” otettiin käyttöön mahdollistamaan mukautettujen buildien luominen. Lisäksi WebRTC:n toimivuuden varmistamiseksi sovelluksen natiivikomponentteja muokattiin lisäämällä tarvittavat oikeudet kameran ja mikrofonin käyttöön.

Näiden muutosten myötä WebRTC:n toiminnallisuus saatiin lopulta integroitua Expo-ympäristöön. Tämä ratkaisu osoittautui paitsi tehokkaaksi myös pitkäjänteiseksi, sillä se mahdollisti tulevaisuudessa muidenkin natiivimoduulien integroimisen sovellukseen ilman rajoituksia. WebRTC:n käyttöönotto Expo Development Buildissä tarjosi myös syvempää ymmärrystä siitä, kuinka Expon Managed Workflow -tilan rajoitukset voidaan kiertää, mikä on arvokasta tietoa sovelluksen jatkokehityksen kannalta.

Tämä haaste opetti, kuinka tärkeää on valita joustava kehitysympäristö, joka tukee projektin vaatimuksia, erityisesti kun kyseessä ovat teknisesti vaativat ominaisuudet, kuten WebRTC:n kaltaiset reaaliaikaiset viestintäprotokollat.

3.3.2 Duplicate Firebase-luokkavirheet

Firebase-kirjastojen integrointi React Native Expo -ympäristöön aiheutti odottamattomia ongelmia, kun kehitysympäristö raportoi "duplicate class" -virheitä. Tämä ongelma liittyi erityisesti Firebase Timestamp -luokkiin, joita eri Firebase-kirjastojen versiot käsittelivät eri tavoin. Tilanne syntyi, koska projektissa käytetyt Firebase-paketit, kuten auth, firestore ja database, eivät olleet keskenään täysin yhteensopivia johtuen niiden eri versioista. Virheet estivät kehityskokoonpanon kääntymisen, mikä hidasti projektin etenemistä merkittävästi.

Ongelman ratkaiseminen vaati järjestelmällistä lähestymistapaa. Ensimmäiseksi selvitettiin tarkasti, mitkä Firebase-paketit ja niiden versiot olivat käytössä projektissa. Havaittiin, että eri kirjastojen riippuvuudet aiheuttivat ristiriitoja, jotka johtivat luokkien kaksoismäärittelyyn. Ratkaisuksi kaikki Firebase-kirjastot päivitettiin samalle versiotasolle, joka oli yhteensopiva React Native Expo -ympäristön kanssa. Tämä päivitys vaati myös projektin app.json-konfiguraation tarkistamista ja muokkaamista niin, että se vastasi päivitettyjä Firebase-riippuvuuksia. Lisäksi varmistettiin, että Expo-ympäristö tukee näitä kirjastoja, jotta sovellus voi toimia odotetusti.

Vaikka Expo ei käytä perinteistä build.gradle-tiedostoa, oli tarpeen tehdä joitakin mukautuksia Metro Bundlerin asetuksiin. Näillä muutoksilla estettiin ristiriitaiset viittaukset Firebase-luokkiin, jotka olivat olleet ongelman ytimessä. Lopuksi suoritettiin kattava testaus, joka varmisti, että duplicate-luokkavirheet oli ratkaistu, eikä uusia konflikteja syntynyt projektissa.

Ongelman ratkaiseminen korosti huolellisen versiohallinnan merkitystä laajoissa projekteissa. Tämän lisäksi se opetti tärkeitä periaatteita liittyen kirjastoihin ja niiden riippuvuuksien hallintaan, erityisesti silloin, kun käytössä on monimutkainen teknologiaekosysteemi, kuten React Native Expo ja Firebase. Tämä prosessi varmisti, että projekti pystyi jatkumaan ilman teknisiä esteitä, ja antoi samalla arvokasta oppia tulevia kehityshaasteita varten.

3.3.3 Kutsunhallinnan virheet ja ratkaisut

Videopuhelutoiminnon kehityksessä yksi merkittävimmistä haasteista liittyi kutsunhallintaan ja erityisesti tilanteisiin, joissa toinen osapuoli hylkäsi saapuvan puhelun. Alkuperäisessä toteutuksessa hylätyn puhelun käsittelyssä ilmeni useita ongelmia, jotka johtivat epävakaaseen käyttäjäkokemukseen. Näistä yleisin oli tilanne, jossa seuraava soittoyritys epäonnistui kokonaan, tai sovellus kaatui odottamattoman virheen vuoksi. Tämä johtui siitä, että kutsulogiikka ei osannut käsitellä puhelun tilatietojen muutoksia oikein Firestoressa, mikä jätti sovelluksen virhe tilaan.

Ongelman juurisyy löytyi puutteellisesta siivouslogiikasta sekä puhelun tilan hallinnasta. Kun puhelu hylättiin, Firestore-tietokantaan tallennettu kutsun tila jäi määrittelemättömäksi tai sitä ei päivitetty ollenkaan, mikä aiheutti ristiriitoja seuraavan puheluyrityksen yhteydessä. Lisäksi reaktiot käyttäjän tekemään hylkäyspäätökseen eivät heijastuneet kaikkien sovelluksen komponenttien välillä yhdenmukaisesti, mikä lisäsi ongelmien monimutkaisuutta.

Ratkaisun ensimmäinen vaihe oli analysoida kutsuprosessin tilanhallintaa kokonaisuutena ja kartoittaa tilanteet, joissa tila jäi päivittämättä. Päivityksen jälkeen kutsun tila merkittiin selkeästi Firestoressa joko "hylätyksi", "hyväksytyksi" tai "päättyneeksi". Tämä muutos mahdollisti sen, että kaikki sovelluksen komponentit pystyivät reagoimaan yhtenäisesti puhelun tilan muutoksiin.

Toinen vaihe keskittyi sovelluksen siivouslogiikkaan. Kun käyttäjä hylkäsi puhelun, varmistettiin, että kaikki siihen liittyvät resurssit – kuten WebRTC-yhteydet, mediarajapinnat ja Firestore-tiedot – tyhjennettiin oikein. Tämä toteutettiin lisäämällä selkeät siivouskutsut CallScreen.js- ja JoinScreen.js-komponentteihin. Siivouskutsut päivittivät myös Firestoressa olevat tiedot, jotta muut laitteet ja käyttäjät pysyivät ajan tasalla puhelun tilasta.

Lopuksi toteutettiin kattava testaus eri skenaarioilla, kuten tilanteilla, joissa toinen osapuoli ei vastannut tai hylkäsi puhelun. Näiden testien perusteella havaittiin, että parannettu kutsulogiikka poisti suurimman osan virhetilanteista. Kutsujen hallinta oli nyt yhtenäisempi ja reagoi tehokkaasti eri käyttötilanteisiin, mikä paransi sekä luotettavuutta että käyttäjäkokemusta.

Näiden ratkaisujen ansiosta kutsunhallinta saatiin vakaammaksi, ja oppimiskokemus korosti tilanhallinnan sekä resurssien vapauttamisen merkitystä erityisesti monimutkaisissa reaaliaikaisissa sovelluksissa.

3.3.4 Ryhmäkoodin automaattinen luonti ja käyttö

Yksi projektin kehitysvaiheessa kohdatuista haasteista liittyi ryhmäkoodien hallintaan videopuheluissa. Aluksi suunnittelumalli edellytti, että käyttäjät syöttävät ryhmäkoodin manuaalisesti liittyäkseen videopuheluun. Tämä lähestymistapa osoittautui kuitenkin käyttäjäkokemuksen kannalta hankalaksi, sillä manuaalinen syöttäminen lisäsi käyttäjän työmäärää ja oli altis virheille, erityisesti teknisesti vähemmän kokeneiden käyttäjien kohdalla.

Ratkaisuksi kehitettiin mekanismi, jossa ryhmäkoodi luotiin automaattisesti ja sen syöttäminen tehtiin käyttäjälle näkymättömäksi. Järjestelmä tuottaa jokaiselle puhelulle uniikin koodin, joka toimii yhteyden identifiointitunnisteena. Kun käyttäjä aloittaa puhelun, koodi tallennetaan automaattisesti Firestore-tietokantaan, ja muut osallistujat voivat liittyä siihen suoraan kutsun kautta ilman manuaalista koodin syöttämistä.

Teknisesti tämä toteutettiin hyödyntämällä Firestorea sekä uniikkien tunnisteiden luontiin että niiden hallintaan. Jokaisen puhelun alkaessa luotiin uusi dokumentti Firestore-tietokantaan, johon sisältyi automaattisesti generoitu ryhmäkoodi. Tämä koodi liitettiin myös käyttäjien kutsulogiikkaan, jolloin kutsut ja liittymiset puheluun synkronoituvat taustalla Firestore-pohjaisen signalointiprosessin avulla.

Käyttäjän kannalta kokemus yksinkertaistui merkittävästi. Puhelun aloittaja ei enää nähnyt koodia tai joutunut jakamaan sitä manuaalisesti, ja liittyvä osapuoli voi vastata kutsuun yhdellä painalluksella, jolloin sovellus huolehti koodin hakemisesta ja yhteyden muodostamisesta automaattisesti. Tämän toimintamallin ansiosta puhelun aloitus ja liittyminen sujuvat vaivattomasti ja käyttäjäkokemus parani merkittävästi.

Ratkaisu myös vahvisti sovelluksen tietoturvaa, sillä uniikkien tunnisteiden luonti teki ryhmäkoodeista vaikeasti arvattavia ulkopuolisille. Tämä estää ei-toivottuja liittymisiä puheluihin ja parantaa yksityisyyden suojaa. Lisäksi kehitysprosessissa opittiin, kuinka tärkeää on minimoida käyttäjän työmäärä ja automatisoida toistuvia tehtäviä, mikä on erityisen tärkeää käyttäjäystävällisyyden kannalta mobiilisovelluksissa.

4 Pohdinta

Pohdintaosiossa käsitellään projektin aikana saavutettuja tuloksia, onnistumisia ja haasteita sekä tarkastellaan kehitystyön lopputulosta suhteessa asetettuihin tavoitteisiin. Lisäksi arvioidaan, mitä olisi voitu tehdä toisin ja millaisia kehitysmahdollisuuksia projekti tarjoaa tulevaisuudessa. Pohdinta antaa myös tilaisuuden tarkastella projektin vaikutuksia laajemmin sekä reflektoida oppimisprosessia.

4.1 Projektin onnistuminen ja tavoitteiden toteutuminen

Projektin päätavoitteena oli toteuttaa toimiva videopuhelutoiminnallisuus mobiilisovellukseen hyödyntäen WebRTC-teknologiaa ja Firebase Firestorea. Tavoitteet saavutettiin pääosin, ja videopuhelutoiminnallisuus saatiin toteutettua onnistuneesti React Native Expo -ympäristössä. Projektin tärkeimmät osa-alueet, kuten yhteyksien luominen, kutsujen hallinta ja reaaliaikainen tietojen siirto, toimivat odotetulla tavalla.

Erityisen hyvin onnistui WebRTC:n integrointi Firebase Firestore -signalointiprosessiin, mikä mahdollisti vakaat yhteydet käyttäjien välillä. Kehitystyön aikana toteutetut ratkaisut, kuten siirtyminen Expon Development Buildiin ja STUN/TURN-palvelimien lisääminen, paransivat merkittävästi videopuhelutoiminnallisuuden laatua. Lisäksi käyttöliittymän navigointi ja ryhmäkoodien automaattinen hallinta helpottivat käyttäjäkokemusta, mikä oli yksi projektin keskeisistä tavoitteista.

Yhteistyö eri teknologioiden välillä, kuten WebRTC:n ja Firebase Firestoren, osoittautui onnistuneeksi ratkaisuksi, sillä se yhdisti tehokkaan reaaliaikaisen viestinnän ja joustavan tietokantapalvelun. Näiden teknologioiden hyödyntäminen mahdollisti myös projektin laajennettavuuden tulevaisuudessa, mikä oli tärkeä lisäarvo. Kokonaisuudessaan projekti vastasi sille asetettuja odotuksia, ja tavoitteiden saavuttaminen osoitti projektin suunnittelun ja teknisen toteutuksen vahvuudet.

4.2 Haasteet ja kehityskohteet

Projektin aikana kohdatut haasteet liittyivät erityisesti WebRTC:n ja Firebase Firestore -teknologian yhteensopivuuteen sekä sovelluksen vakauden varmistamiseen erilaisissa käyttötilanteissa. WebRTC:n integrointi React Native Expo -ympäristöön osoittautui aluksi haastavaksi, sillä Expo ei tue natiivikomponentteja suoraan. Tämä rajoitus ratkaistiin siirtymällä Development Buildiin, mutta siirtymä toi mukanaan uusia vaatimuksia ympäristön konfiguroinnille ja lisäkirjastojen hallinnalle.

Toinen merkittävä haaste liittyi Firebasen käyttöön signalointiprosessissa. Vaikka Firestore mahdollisti tehokkaan reaaliaikaisen tiedonvaihdon, sen synkronointimekanismi toi ajoittain viivettä kutsujen tilan päivittymiseen. Tämä viive vaikutti erityisesti puheluiden hallintaan, kuten puhelun

hylkäämiseen ja seuraavan puhelun käynnistämiseen, mikä johti epäjohdonmukaisuuksiin sovelluksen toiminnassa. Näiden ongelmien korjaaminen vaati tarkkaa signalointilogiikan uudelleensuunnittelua ja perusteellista testaamista eri skenaarioissa.

Sovelluksen vakauden varmistaminen haastavissa verkkoympäristöissä oli myös keskeinen kehityskohde. Videopuhelut vaativat jatkuvaa ja matalaviiveistä yhteyttä, mikä ei aina ollut mahdollista vaihtelevissa verkkotilanteissa. STUN- ja TURN-palvelimien lisääminen paransi yhteyksien luotettavuutta, mutta haasteita esiintyi edelleen matalan kaistanleveyden ympäristöissä. Näihin haasteisiin olisi voinut varautua paremmin kattavammalla suorituskyvyn optimoinnilla ja erilaisia verkkotilanteita huomioivalla testaamisella.

Kehityksen aikana ilmeni myös pienempiä haasteita, kuten käyttöliittymän ja navigoinnin hienosäätö, joka vaati jatkuvaa kehitystä. Vaikka nämä ongelmat saatiin pääosin ratkaistua, osa ratkaisuista, kuten modaalikomponenttien ulkoasun optimointi näppäimistön kanssa, olisi voitu suunnitella alusta alkaen tehokkaammin.

Näiden haasteiden myötä projektista kertyi paljon arvokasta oppia, erityisesti monimutkaisten teknologioiden integroinnista mobiilisovellukseen. Kehitystyössä ilmeni selkeä tarve panostaa entistä enemmän suunnitteluvaiheeseen sekä eri teknologioiden yhteensopivuuden arviointiin ennen toteutusta. Tämä voisi helpottaa tulevaisuudessa vastaavien haasteiden ennakointia ja ratkaisua.

4.3 Mahdollisuudet ja jatkokehitys

Sovelluksen kehitystyö tarjoaa useita mahdollisuuksia parantaa käyttökokemusta ja laajentaa sen toiminnallisuuksia. Etenkin videopuheluiden laatuun ja käyttäjävuorovaikutukseen liittyvät kehityskohteet voisivat merkittävästi lisätä sovelluksen arvoa ja käytettävyyttä.

Ensimmäinen keskeinen kehityskohde on videopuheluiden laadun parantaminen. Sovellukseen voitaisiin lisätä adaptiivinen videonlaadun säätö, joka optimoi lähetyksen resoluutiota ja kaistanleveyden käyttöä automaattisesti käyttäjän verkkoyhteyden mukaan. Tämä ominaisuus takaisi tasaisemman kokemuksen erityisesti vaihtelevissa verkkoympäristöissä, esimerkiksi heikossa mobiiliverkossa. Lisäksi puheluiden äänen ja videon synkronoinnin parantaminen voisi vähentää viiveitä ja tehdä vuorovaikutuksesta entistä sujuvampaa.

Toinen merkittävä kehityskohde liittyy parempien push-ilmoitusten implementointiin. Nykyisiä ilmoituksia voisi parantaa lisäämällä niihin interaktiiviset painikkeet, kuten "Hyväksy" ja "Hylkää" -valinnat, jolloin käyttäjä voisi vastata puheluun tai hylätä sen suoraan ilmoitusnäkymästä ilman tarvetta avata sovellusta. Tämä paitsi nopeuttaisi käyttäjän toimintaa myös parantaisi käyttökokemusta erityisesti kiireellisissä tilanteissa.

Kolmas tärkeä kehityssuunta on odottamattomien puhelukatkosten käsittely. Jos videopuhelu katkeaa yllättäen esimerkiksi verkkoyhteysongelmien vuoksi, käyttäjille tulisi tarjota selkeä ja informatiivinen ilmoitus, joka kertoo yhteyden katkeamisesta. Tämä voisi sisältää myös mahdollisuuden yrittää yhdistää puhelu uudelleen ilman, että käyttäjä joutuu aloittamaan koko prosessia alusta. Tällainen ominaisuus lisäisi sovelluksen luotettavuutta ja vähentäisi turhautumista ongelmatilanteissa.

Lisäksi käyttäjäkokemusta voitaisiin parantaa muokkaamalla puhelun aloitus- ja lopetusprosessia intuitiivisemmaksi. Esimerkiksi puheluiden tilan (soitetaan, odotetaan vastausta, päättynyt) näkyvyyttä käyttöliittymässä voisi selkeyttää. Tähän voisi yhdistää myös ääni- ja visuaalisia ilmoituksia, jotka auttavat käyttäjiä navigoimaan puheluprosessin eri vaiheissa.

Jatkon osalta voisi myös harkita lisäominaisuuksia, kuten tekstimuotoisten viestien lähettämistä puhelun aikana. Tämä mahdollistaisi helpon tavan kommunikoida tilanteissa, joissa ääni tai video ei jostain syystä toimi. Lisäksi sovellukseen voisi lisätä tilastotyökalun, jolla käyttäjä voi tarkastella aiempia puhelujaan, niiden kestoja ja mahdollisia teknisiä ongelmia.

Näiden kehityskohteiden toteuttaminen vaatii tarkkaa suunnittelua ja testausta, mutta ne tarjoaisivat merkittävää lisäarvoa sovelluksen käyttäjille.

4.4 Opit ja reflektointi

Projektin aikana on ollut mahdollista kehittää paitsi teknisiä taitoja myös projektinhallinnan ja ongelmanratkaisun osaamista. Erityisesti WebRTC-teknologian ja Firebase-sovelluksen integrointi tarjosi arvokkaita oppimiskokemuksia, joiden avulla olen syventänyt ymmärrystäni reaaliaikaisten sovellusten kehittämisestä ja niiden haasteista.

Tekniseltä puolelta projektin suurimpia oppeja olivat WebRTC:n ja Firebase Firestoren yhteensovittaminen Expo-ympäristössä, erityisesti signalointiprosessin toteuttaminen Firebaseen ja WebRTC:hen perustuvassa videopuhelusovelluksessa. WebRTC:n vaatimukset ja rajoitukset mobiiliympäristössä, erityisesti Expo-sovelluksen kontekstissa, olivat aluksi haasteellisia. Opin kuinka tärkeää on testata ja valita oikeat kehitystyökalut ja konfiguraatiot varmistamaan, että sovellukset toimivat optimaalisesti eri alustoilla. Development buildin käyttö antoi konkreettista tietoa ja ratkaisuja, jotka mahdollistivat WebRTC:n toiminnan Expo-ympäristössä.

Projektinhallinnassa olen oppinut, kuinka tärkeää on selkeä aikarajojen ja resurssien hallinta. Aikataulut ja kehitysvaiheet on suunniteltava realistisesti ottaen huomioon tekniset haasteet ja testauksen vaatimukset. Tämän projektin aikana opin myös tärkeitä taitoja kommunikoinnista ja

tiimityöstä, sillä teknisten ratkaisujen lisäksi oli tärkeää varmistaa, että kaikki osapuolet olivat tietoisia projektin tavoitteista ja aikarajoista.

Ongelmanratkaisutaidot ovat kehittyneet erityisesti puhelun signalointiprosessissa ja yhteydenhallinnassa. Puhelujen hylkäämisestä johtuvat ongelmat olivat yksi esimerkki siitä, kuinka tärkeää on ottaa huomioon kaikki mahdolliset käyttötapaukset ja niiden vaikutukset sovelluksen toimivuuteen. Opin, kuinka tärkeää on testata kaikki mahdolliset virhetilanteet ja ennakoida käyttäjien toimet, jotta sovellus toimii moitteettomasti kaikissa olosuhteissa.

Kaiken kaikkiaan tämä projekti on ollut merkittävä oppimiskokemus, jossa sain paitsi syvällistä teknistä osaamista myös käytännön kokemusta projektin suunnittelusta, toteutuksesta ja hallinnasta. Nämä taidot tulevat olemaan hyödyllisiä tulevissa projekteissani, joissa tarvitaan kykyä ratkaista monimutkaisia teknisiä ongelmia ja hallita tiimejä ja aikarajoja tehokkaasti.

Lähteet

Cloud Firestore. Documentation. Luettavissa: https://firebase.google.com/docs/firestore. Luettu 3.11.2024

Create a development build. Luettavissa: https://docs.expo.dev/develop/development-builds/create-a-build/. Luettu 23.12.2024

Digital Samba. ICE and SDP in WebRTC. Luettavissa: https://www.digitalsamba.com/blog/ice-and-sdp-in-webrtc. Luettu 2.11.2024

Expo-dev-client. Luettavissa: https://docs.expo.dev/versions/latest/sdk/dev-client/. Luettu 23.12.2024

Expo Docs. Develop an app with Expo. Luettavissa: https://docs.expo.dev/workflow/overview/. Luettu 1.11.2024

Expo Docs. Use libraries. Luettavissa: https://docs.expo.dev/workflow/using-libraries/. Luettu 1.11.2024

Expo Notifications. Luettavissa: https://docs.expo.dev/versions/latest/sdk/notifications/. Luettu 23.12.2024

Firebase. Documentation. Luettavissa: https://firebase.google.com/docs/guides. Luettu 3.11.2024

Interactive Connectivity Establishment (ICE). Luettavissa: https://www.geeksforgeeks.org/interactive-connectivity-establishment-ice/. Luettu 23.12.2024

Mary E. Shacklett. WebRTC (Web Real-Time Communications). Luettavissa: https://www.techtarget.com/searchunifiedcommunications/definition/WebRTC-Web-Real-Time-Communications. Luettu 2.11.2024

Mathew Pregasen. Deploying WebRTC on an Expo React Native app. Luettavissa: https://www.daily.co/blog/deploying-webrtc-on-an-expo-react-native-app-2/. Luettu 1.11.2024

Multilogin. WebRTC STUN. Luettavissa: https://multilogin.com/glossary/webrtc-stun/. Luettu 2.11.2024

React-native-webrtc. Luettavissa: https://github.com/react-native-webrtc/react-native-webrtc. Luettu 23.12.2024

Shoutem. A brief history of React Native. Luettavissa: https://medium.com/react-native-development/a-brief-history-of-react-native-aae11f4ca39. Luettu 1.11.2024

Signaling and video calling. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Signaling_and_video_calling. Luettu 23.12.2024

Stream. Learn STUN & TURN Servers on WebRTC. Luettavissa: https://getstream.io/resources/projects/webrtc/advanced/stun-turn/. Luettu 2.11.2024

WebRTC. Home Page. Luettavissa: https://webrtc.org/. Luettu 1.11.2024

What is SDP - Session Description Protocol?. Luettavissa: https://www.3cx.com/pbx/sdp/. Luettu 23.12.2024

What is SRTP - Secure Real-Time Transport Protocol?. Luettavissa: https://www.3cx.com/voip/srtp/. Luettu 23.12.2024

Ömür Bilgili. Exploring Expo in React Native: A Comprehensive Guide to Cross-Platform App Development. Luettavissa: https://omurbilgili.medium.com/exploring-expo-in-react-native-a-comprehensive-guide-to-cross-platform-app-development-45e6a3bfa111. Luettu 1.11.2024