

Janne Kolhonen

# MOBIILILAITTEIDEN VÄLISEN TIEDON- SIIRRON SUUNNITTELU JA TOTEUTUS

Tieto- ja sähkötekniikan tiedekunta  
Diplomityö  
Maaliskuu 2019

## TIIVISTELMÄ

**JANNE KOLHONEN:** Mobiililaitteiden välisen tiedonsiirron suunnittelu ja toteutus  
Tampereen teknillinen yliopisto  
Diplomityö, 49 sivua  
Maaliskuu 2019  
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma  
Pääaine: Ohjelmistotuotanto  
Tarkastajat: Professori Kari Systä, Yliopistonlehtori Terhi Kilamo

Avainsanat: tiedonsiirto, mobiilisovellus, Android, SDK, API

Mobiililaitteiden ollessa nykypäivänä lähes välttämättömyys monille ja uusien mallien tullessa jatkuvasti markkinoille, monet käyttäjät ovatkin tottuneet vaihtamaan laitteensa uuteen jopa parin vuoden välein. Mobiililaitteen vaihtamiseen liittyy oleellisesti tietojen saanti vanhasta laitteesta uuteen ja monet yritykset kuten mobiililaitteiden jälleenmyyjät ja teleoperaattorit myyvätkin tietojensiirtoa palveluna loppukäyttäjille. Tiedonsiirron toteuttavat sovellukset ovat tyypillisesti toimineet tietokoneissa, joiden kautta tiedonsiirto on tehty. Markkinoilla on kuitenkin mahdollisesti kysyntää myös mobiililaitteissa toimiville suoraan mobiililaitteissa toiseen tiedonsiirron toteuttaville sovelluksille.

Tämän diplomityön aiheena on Piceasoftille suunniteltu ja toteutettu mobiililaitteiden välisen tiedonsiirron toteuttava mobiilisovellus. Ensimmäisenä diplomityössä esitellään sen taustat ja tarpeet, jonka jälkeen esitellään niistä muodostetut, toteutettavan sovelluksen vaatimukset. Lisäksi esitellään työn toteutukseen valitut teknologiat sekä niiden avulla toteutettu sovellus. Lopuksi arvioidaan toteutettua sovellusta vertaamalla sitä sille asetettuihin vaatimuksiin sekä suorituskykymittauksilla.

Työssä toteutettiin soveltuvuusselvityksenä toimiva mobiilisovellus Androidille. Toteutus muodostuu sovellusprojektin lisäksi sovelluksen päätoiminnallisuuden toteutettavasta kirjastoprojektista. Sovellus ei sellaisenaan ole valmis kaupalliseen käyttöön, mutta sen avulla voidaan esitellä suoraan mobiililaitteesta toiseen tapahtuvan tiedonsiirron toimivuus. Jos mobiililaitteesta toiseen tapahtuvalle siirrolle löydetään kaupallinen käyttötausa, on toteutuksen jatkokehittäminen helppoa. Kirjasto-toteutuksen ansiosta myös toiminnallisuuden käyttöönotto onnistuu helposti Piceasoftin omaan PiceaOne-mobiilisovellukseen.

## ABSTRACT

**JANNE KOLHONEN:** Design and implementation of a data transfer between mobile devices

Tampere University of Technology

Master of Science Thesis, 49 pages

March 2019

Master's Degree Programme in Information Technology

Major: Software Engineering

Examiner: Professor Kari Systä, University Lecturer Terhi Kilamo

Keywords: data transfer, mobile application, Android, SDK, API

Mobile devices being almost necessity for many these days and as new models are continuously released to the markets, many users have gotten used to change their devices to new one even every two years. Transferring data from old device to the new one is essential part of changing the mobile device and many companies like mobile device resellers and teleoperators have started selling data transfer as a service for the end users. The data transfer has been typically done with an application running on a computer, through which the data transfer is made. On the markets there is possibly demand also for the mobile applications implementing the data transfer directly from the mobile device to other.

The topic of this master's thesis is a mobile application designed and implemented for Piceasoft to do the data transfer between mobile devices. The background and needs of the thesis are introduced first, after which the requirements of the implemented application are presented. Further the chosen technologies and the implementation based on those are presented. Finally, the implemented application is evaluated by comparing it with the requirements set for it and with the performance measurements.

In this work, a proof-of-concept mobile application was implemented for Android. The implementation consists of an application project and a library project implementing the main functionality of the application. The application as such is not ready for commercial use, but it can be used to demonstrate the functionality of data transfer from the mobile device to another. If a commercial use case is found for the direct transfer between mobile devices, further development of the implementation is easy. Library implementation also makes it easy for the functionality to be taken into use in Piceasoft's own PiceaOne mobile application.

## ALKUSANAT

Tämän diplomityön aiheena on Piceasoftille kehitetty Android-sovellus, joka mahdollistaa mobiililaitteiden välisen tiedonsiirron suoraan laitteesta toiseen tai pilvipalvelun välityksellä. Haluan kiittää Piceasoftia mahdollisuudesta työn tekemiseen sekä erityisesti työn tarkastajia Kari Systää ja Terhi Kilamoa heidän antamastaan ohjauksesta ja rakentavasta palautteesta. Lisäksi haluan kiittää työkavereitani ja läheisiäni heidän antamastaan tuesta diplomityön teon aikana.

Tampereella, 15.3.2019

Janne Kolhonen

## SISÄLLYSLUETTELO

1.	JOHDANTO .....	1
2.	TAUSTAA .....	2
2.1	Piceasoftin nykyinen ratkaisu.....	2
2.2	Kilpailevat ohjelmistot .....	4
3.	VAATIMUKSET .....	6
3.1	Toiminnalliset vaatimukset .....	6
3.2	Ei-toiminnalliset vaatimukset.....	8
3.2.1	Siirrettävyys ja laajennettavuus .....	8
3.2.2	Käytettävyys.....	9
3.2.3	Tietoturva .....	10
4.	KÄYTETYT TEKNOLOGIAT .....	12
4.1	Android.....	12
4.2	Tiedonsiirtotekniikat .....	15
4.2.1	Bluetooth.....	15
4.2.2	Wi-Fi Direct .....	16
4.2.3	USB On-The-Go .....	19
4.3	Tiedonsiirtoprotokollat.....	20
4.3.1	TCP .....	20
4.3.2	HTTP ja HTTPS.....	22
4.4	Salaus .....	22
4.4.1	SSL/TLS.....	23
4.4.2	AES .....	24
4.5	Eheys .....	24
5.	TOTEUTETTU TIEDONSIIRTOSOVELLUS.....	26
5.1	Tiedonsiirtokirjasto .....	27
5.1.1	Arkkitehtuuri.....	27
5.1.2	Ohjelmointirajapinta .....	34
5.2	Mobiilisovellus.....	37
5.2.1	Käyttöliittymä .....	38
5.2.2	Toteutus.....	42
5.3	Pilvitalennuspalvelu .....	43
6.	ARVIOINTI.....	46
6.1	Vaatimuksien täyttyminen.....	46
6.2	Suorituskyky.....	47
7.	YHTEENVETO .....	49
	LÄHTEET .....	50

## LYHENTEET JA MERKINNÄT

AAR	Android Archive on pakettiformaatti, joka mahdollistaa kirjastomoduulien mukaan ottamisen Android-sovelluksiin.
ADB	Android Debug Bridge on komentorivityökalu, joka mahdollistaa kommunikoinnin Android-laitteiden kanssa. Työkalun avulla voidaan esimerkiksi asentaa ja poistaa sovelluksia Android-laitteesta.
AES	Advanced Encryption Standard on symmetrinen lohkosalausmenetelmä, jota nykyään suositellaan käytettäväksi tiedon salauksessa.
API	Application Programming Interface eli ohjelmointirajapinta määrittelee, kuinka ohjelmisto tarjoaa tietoja tai palveluita muille ohjelmistoille.
APK	Android Application Package on pakettiformaatti, jota käytetään Android-käyttöjärjestelmässä sovelluksien asentamiseen.
HTTP	Hypertext Transfer Protocol on siirtoprotokolla, jolla selaimet ja WWW-palvelimet kommunikoivat keskenään.
HTTPS	Hypertext Transfer Protocol Secure on HTTP-protokollan ja SSL/TLS-salausprotokollan yhdistelmä. Mahdollistaa suojatun tiedonsiirron selaimien ja WWW-palvelimien välillä.
IDE	Integrated Development Environment eli ohjelmointiympäristö on ohjelmisto, jolla ohjelmoija kehittää ohjelmistoa.
JSON	JavaScript Object Notation, tiedon tallennukseen ja siirtämiseen kehitetty avoimen standardin tiedostoformaatti.
MD5	MD5 message-digest -algoritmi on kryptografinen tiivistefunktio, jonka avulla laskettuja tarkistussummia käytetään usein tiedon eheyden varmistamiseen. Ei enää kryptografisesti turvallinen, seuraaja SHA-1 ja SHA-2.
SDK	Software Development Kit eli ohjelmistokehityspaketti on joukko työkaluja, kuten ohjelmistokirjastoja ja dokumentaatiota, joiden avulla voidaan kehittää sovelluksia tiettyyn tarkoitukseen.
SHA	Secure Hash Algorithm 1 ja 2 ovat kryptografisia tiivistefunktioita, joiden avulla laskettuja tarkistussummia käytetään tiedon eheyden varmistamiseen. Kryptografisesti turvallisempia kuin MD5.
SSL	Secure Socket Layer on nykyään jo vanhentunut salausprotokolla, joka mahdollistaa salatun tiedonsiirron. Seuraaja TLS.
TCP	Transmission Control Protocol on yhteydellinen tietoliikenneprotokolla, jolla tietoverkon laitteet voivat lähettää luotettavasti tietoa toisilleen.
TLS	Transport Layer Security on SSL-protokollan seuraaja ja nykyisin salattuun tiedonsiirtoon verkossa käytettävä protokolla.
XML	Extensible Markup Language on rakenteellinen kuvauskieli ja tiedostoformaatti.

# 1. JOHDANTO

Mobiililaitte- ja erityisesti älypuhelinmarkkinoiden kasvettua valtavasti viime vuosikymmenen aikana, on myös käytettyjen mobiililaitteiden markkinat kasvaneet valtavasti. Käytettyjen älypuhelimien markkinoiden ennustetaan kasvavan jopa 4-5 kertaa nopeammin kuin uusien laitteiden, vuosittain myytävien käytettyjen älypuhelimien määrän ollessa jo yli 100 miljoonaa [24]. Niin käytettyjen mobiililaitteiden luopumiseen kuin uusienkin mobiililaitteiden hankkimiseen liittyy oleellisesti niiden elinkaaren hallinta ja mobiililaitemarkkinoiden valtavan kasvun takia monet teknologiayritykset ovatkin alkaneet tarjota tätä tarkoitusta varten kehitettyjä tuotteita ja palveluita. Yksi näistä yrityksistä on suomalainen Piceasoft.

Piceasoft on vuonna 2012 Tampereella perustettu ohjelmistoyritys, joka on erikoistunut mobiililaitteiden elinkaaren hallintaan liittyvien sovelluksien kehitykseen. Piceasoftin tuotteiden mahdollistaessa mm. sisällön siirtämisen mobiililaitteesta toiseen ja vanhan mobiililaitteen täydellisen ja tietoturvallisen tyhjentämisen, on mobiililaitteiden käyttäjien helpompi siirtyä uuteen laitteeseen ja esimerkiksi myydä vanha.

Tämän diplomityön aiheena on Piceasoftille kehitetty mobiililaitteiden välisen tiedonsiirron suoraan laitteesta toiseen tai mahdollisesti pilvitallennuspalvelun avulla toteutettava mobiilisovellus. Kehitettävän sovelluksen on tarkoitus toimia soveltuvuusselvityksenä (engl. proof-of-concept) eikä sen tarkoituksena ole vielä olla lopullinen tuote. Työssä sovellus toteutettiin Android-mobiilikäyttöjärjestelmälle ottaen kuitenkin huomioon ratkaisun siirrettävyys iOS-käyttöjärjestelmälle. Toteutuksessa tuli myös ottaa huomioon eri teknologioiden mahdollinen käyttö tiedonsiirrossa sekä mahdollisuus laajentaa toteutettu ratkaisu käyttämään uusia tiedonsiirtotapoja.

Dokumentin toisessa luvussa kerrotaan diplomityön toteutukseen ja sen tarpeisiin liittyviä taustoja sekä käydään läpi toteutetun työn keskeisimpiä tavoitteita. Kolmannessa luvussa kerrotaan tarkemmin Piceasoftin ja sen asiakastahojen ohjelmistolle asettamista toiminnallisista ja ei-toiminnallisista vaatimuksista. Neljännessä luvussa esitellään työn toteuttamisessa käytettyjä teknologioita. Viidennessä luvussa kerrotaan tarkemmin työn teknisestä toteutuksesta ja sen rakenteesta. Lopuksi arvioidaan käytettyjen teknologioiden toimivuutta ja verrataan toteutettua kokonaisuutta työn alussa sille asetettuihin tavoitteisiin.

## 2. TAUSTAA

Mobiililaitteiden elinkaaren hallintaan ja samalla mobiililaitteesta toiseen siirtymiseen liittyy olennaisesti vanhan laitteen tyhjentäminen sekä sisällön siirto vanhasta laitteesta uuteen. Piceasoft tarjoaakin tuotteet näiden toimenpiteiden suorittamiseen, kuten myös laitteen analysointiin mahdollisten vikojen varalta. Laitteen diagnosointi onkin hyvin oleellinen toimenpide laitteen jälleenmyyntikelpoisuuden ja hinnan määrittämiseksi.

Piceasoft ei toimi suoraan kuluttajaliiketoiminnassa vaan sen ensisijaisia asiakkaita ovat toiset yritykset kuten teleoperaattorit, mobiililaitteiden jälleenmyyntiä ja käytettyjen mobiililaitteiden kierrätystä harjoittavat yritykset, jotka myyvät omia palveluitaan Piceasoftin ohjelmistojen avulla. Käytettyjen laitteiden markkinoiden kasvaessa on kilpailu näistä asiakkaista sekä elinkaaren hallintaan käytettyjen ohjelmistojen välillä kuitenkin kasvanut.

Loppukäyttäjille palveluna myytävää sisällön siirtoa on perinteisesti tehty ohjelmistolla, jota suoritetaan tietokoneella, johon mobiililaitteet ovat yhdistettynä USB-kaapeleilla. Piceasoft on vakiinnuttanut asemaansa näillä markkinoilla omalla PiceaSwitch-ohjelmistollaan, mutta vain mobiililaitteella suoritettavien siirtosovellusten lisääntyminen on kuitenkin luonut myös Piceasoftille kysyntää laajentaa näille markkinoille. Piceasoftilla mahdollisuutta sisällön siirron toteuttamiseksi suoraan mobiililaitteesta toiseen ilman tietokonetta päätettiin siinä alkaa selvittämään ensisijaisesti asiakastahoilta jo pidempään tulleen kysynnän vuoksi.

Keskeisimmät perustelut ominaisuuden tarpeelle ovat olleet se, että tiedonsiirron toteuttaminen mobiililaitteilla nopeuttaisi esimerkiksi Piceasoftin asiakkaiden sekä heidän loppuasiakkaidensa palvelutiskeillä kuluttamaa aikaa. Myös mobiililaitteiden tallennustilan kasvaessa on joillekin Piceasoftin asiakkaille muodostunut ongelmaksi heidän käyttämiensä tietokoneiden liian pieni tallennustila suhteessa mobiililaitteissa olevan tiedon määrään. Mobiililaitteilla toteutettu tiedonsiirto vastaisi myös kysyntään sellaisilta mahdollisilta asiakkailta, joilla ei tietoturvasyistä ole mahdollisuutta käyttää USB-portteja laitteiden yhdistämiseksi tietokoneisiin tai niille, joiden toimipisteissä ei ole käytössä lainkaan tietokoneita, vaan esimerkiksi tabletit.

### 2.1 Piceasoftin nykyinen ratkaisu

Piceasoftin kehittämä PiceaServices-ohjelmistokokonaisuus muodostuu useammasta yrityksen tuoteportfolion ohjelmistosta. Näitä tuotteita ovat esimerkiksi mobiililaitteiden diagnosointiin käytetty PiceaDiagnostics, laitteiden tyhjennykseen käytetty PiceaEraser sekä laitteiden väliseen sisällön siirtoon käytetty PiceaSwitch.

PiceaSwitch on Piceasoftin nykyinen Windows ja Mac käyttöjärjestelmillä toimiva tiedonsiirto-ohjelmisto, joka tukee niin Android, iOS kuin Windows Phone käyttöjärjestelmän mobiililaitteita. Se tukee laitteesta toiseen siirtämisen lisäksi myös varmuuskopion tekemistä USB-muistitikulle tai pilvitalennuspalveluun [35]. Siirrettäessä sisältöä mobiililaitteesta toiseen, ne kytketään tietokoneeseen USB-väylien avulla. Mobiilialustojen erilaisuuksien takia sisällön lähteenä ja sen kohteena toimivien laitteiden käyttöjärjestelmät vaikuttavat siihen mitä sisältöä niiden välillä voidaan siirtää ja siihen, kuinka sisällön käsittely ja siirto on teknisesti toteutettu.

Siirrettäessä sisältöä Android-laitteesta toiseen, PiceaSwitch tukee seuraavia sisältötyyppejä: kontaktit, kalenteri, viestit, kirjanmerkit, musiikki, kuvat, videot sekä dokumentit. Myös laitteeseen asennettujen sovellusten siirto onnistuu, mutta niiden siirtämiseksi käytetään Google-tiliä. Musiikin, kuvien, videoiden ja dokumenttien siirtäminen onnistuu yksinkertaisesti asettamalla laitteet USB tiedonsiirto -moodiin, jolloin PiceaSwitch voi lukea tiedostot ensin lähdelaitteen tiedostojärjestelmästä tietokoneelle ja kirjoittaa ne sitten kohdelaitteen tiedostojärjestelmään. Kontaktien, kalenterin, viestien ja kirjanmerkien siirtämiseksi laitteisiin tulee asentaa sovellus, joka huolehtii näiden tietojen keräämisestä Android-järjestelmästä.

Piceasoftin ratkaisu tähän on PiceaOne-mobiilisovellus, jonka PiceaSwitch asentaa laitteisiin. Ennen kuin tietokone voi kuitenkaan asentaa mitään Android-laitteeseen, tulee Androidissa olla USB-vianetsintä (engl. USB debugging) kytkettynä päälle. USB-vianetsintäasetus on piilotetuissa Androidin kehittäjäasetuksissa, jolloin sen kytkeminen päälle vaatii ensin myös Androidin kehittäjäasetuksien päälle laittamisen. Kun USB-vianetsintä on päällä, PiceaSwitch voi asentaa PiceaOne-sovelluksen ADB (Android Debug Bridge) -työkalua käyttäen. ADB-työkalu mahdollistaa TCP-yhteyden avaamisen tietokoneen ja mobiililaitteen välille, jolloin PiceaSwitch ja PiceaOne voivat kommunikoida keskenään [9].

PiceaSwitchillä ja tietokoneella ylipäätään tehtävässä tiedonsiirrossa on niin hyvät kuin huonot puolensa. Tietokoneella tehtävän tiedonsiirron eduksi voidaan katsoa sen käyttämien USB-väylien nopeus, sillä mobiililaitteiden jo vanhempaa USB 2.0 -standardia ja nykyään jo uusimmissa laitteissa normaaliksi muodostuneiden USB 3 -standardia käyttävien USB-väylien nopeudet ovat useita langattomia verkkotekniikoita nopeampia. Tämän lisäksi tietokoneen toimiessa mobiililaitteiden välillä, on virhetilanteiden hallinta helppompaa.

Tiedonsiirron tapahtuessa mobiililaitteiden välillä toimivan tietokoneen kautta, vaikuttaa se kuitenkin negatiivisesti tiedonsiirron nopeuteen, koska data siirretään ensin tietokoneeseen ja siitä vasta toiseen laitteeseen. Tietokoneella suoritettavan siirron huonona puoleena verrattuna suoraan mobiililaitteissa toimivien sovelluksien avulla tapahtuvaan tiedonsiirtoon on myös se, että laitteissa tulee kuitenkin olla mobiilisovellus asennettuna, jotta niistä saadaan mahdollisimman paljon tietoja siirrettyä. Mobiilisovelluksen asennus

itsessään vaatii myös erinäisten asetuksien päälle laittamisen, jolloin siirtoa edeltävä asetustyö vie aikaa.

PiceaSwitch tuottaa tekemistään siirroistaan myös raportin, joka kertoo siirron onnistumisesta. Sovelluksen on helppo tuottaa raportti, sillä kaikki sisältö on siirretty sen kautta, jolloin se on myös tietoinen siirtojen onnistumisesta tai epäonnistumisesta. Näissä tietokoneissa on usein myös verkkoyhteys, jolloin raportti saadaan välittömästi välitettyä Piceasoftin raportointipalvelimelle. Raportissa on kerrottu siirrettyjen tiedostojen ja muun datan määrä sisältötyypeittäin eriteltynä. Tätä raportin dataa voidaan käyttää niin Piceasoftin sisäiseen kuin siirron tehneen Piceasoftin asiakkaankin analytiikkaan.

## 2.2 Kilpailevat ohjelmistot

Markkinoilla on olemassa useita erilaisia tiedonsiirtosovelluksia niin kaupalliseen kuin ei-kaupalliseenkin käyttöön. Kaupalliset sovellukset ovat pääasiassa Piceasoftin tavoin mobiililaitteiden elinkaarenhallintaan ohjelmistoja kehittävien yritysten, mutta ei-kaupallisia sovelluksia löytyy niin yksittäisten kehittäjien kuin laitevalmistajienkin kehittämisenä. Merkittävimpänä erona näiden sovellusten välillä voidaan pitää sitä, että kaupallisten sovelluksien avulla yritykset myyvät tiedonsiirtoa palveluna mobiililaitteiden loppukäyttäjille, kun taas ei-kaupalliset sovellukset ovat pääasiassa kotikäyttöön tarkoitettuja, joiden käytöstä vastaa loppukäyttäjä itse.

Markkinoilta löytyvien erilaisten tiedonsiirtosovelluksien perusteella sovellukset voidaan jakaa myös sen mukaan millaisissa järjestelmissä ne toimivat. Piceasoftin PiceaSwitch-ohjelmiston tavoin tietokoneissa, tai vastaavissa tiedonsiirtoon dedikoiduissa erillisissä järjestelmissä toimivat ohjelmistot, ovat yleisimpiä kaupallisten toimijoiden kuten teleoperaattoreiden tai mobiililaitteiden jälleenmyyjien käytössä. Mutta kuten luvussa 2 todettiin, kaupallisilla toimijoilla on käyttöä myös itse mobiililaitteissa toimiville tiedonsiirtosovelluksille.

Tässä työssä toteutetun ohjelmiston tavoin mobiililaitteissa itsessään ajettavat sovellukset, jotka suorittavat tiedonsiirron suoraan laitteesta toiseen tai mahdollisesti pilvitallennuspalvelun välityksellä, ovat yleisimpiä laitteen loppukäyttäjän käytössä. Näitä sovelluksia löytyy jo useilta laitevalmistajilta kuten Samsungilta, Sonylta ja Applelta, joiden sovellukset ovatkin usein esiasennettuina niiden valmistamiin laitteisiin. Android-alustalla nämä sovellukset ovat usein myös ladattavissa muiden valmistajien laitteisiin, mutta ne mahdollistavat näiden laitteiden toimivan sisällön siirrossa vain lähdelaitteena, jolloin ne tukevat siirtoa vain valmistajan omiin laitteisiin. Android-laitevalmistajien omien sovelluksien lisäksi Androidille löytyy Googlen PlayStoresta useita ilmaisia tiedonsiirtosovelluksia, jotka ovat kuitenkin täysin kotikäyttöön tarkoitettuja.

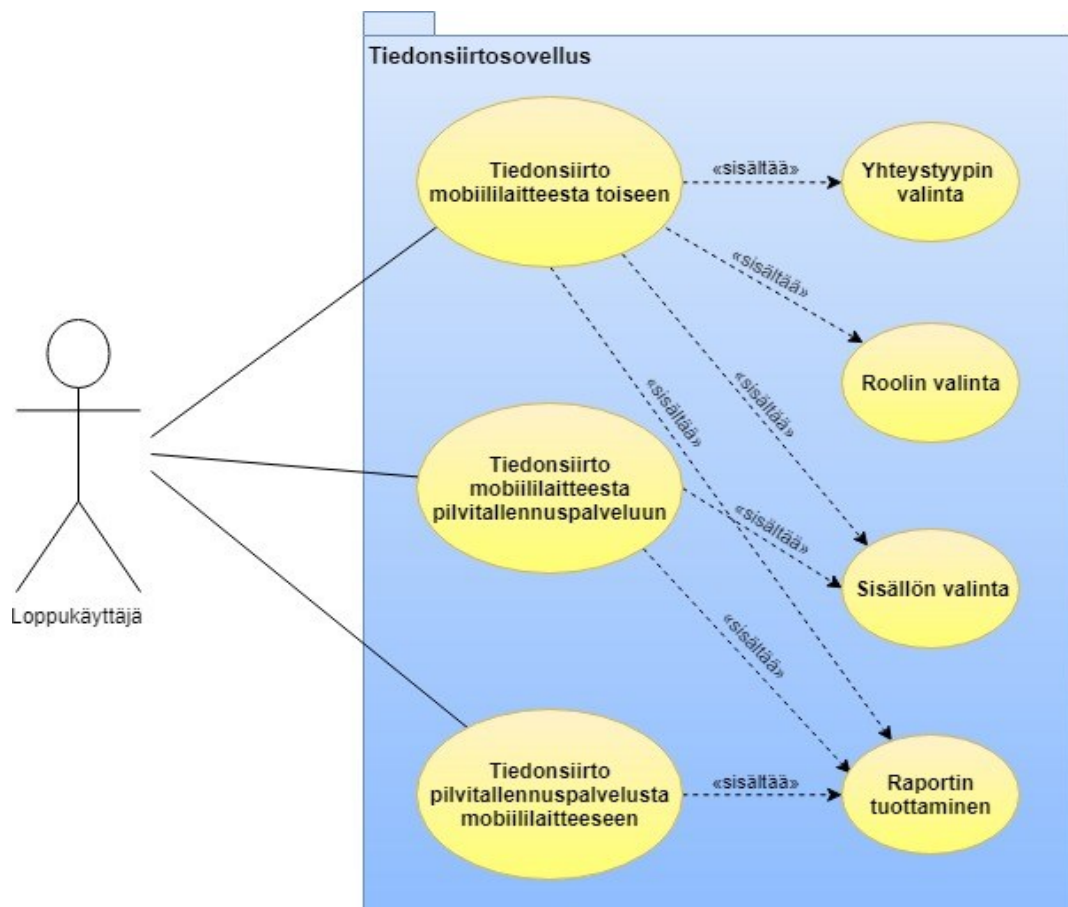
Yleisimmät kriteerit näitä kaikkia tiedonsiirto-sovelluksia verratessa ovat tiedonsiirron nopeus, toimintavarmuus sekä tuotteen käytettävyys. Myös ominaisuudet kuten tuetut tiedonsiirtotavat, sisältötyypit, tiedostoformaatit, laitetypit ja niiden käyttöjärjestelmät ovat ratkaisevia tekijöitä siirto-sovellusta valittaessa. Tällaisten tuotteiden kanssa kilpaileminen vaatii erottumista juurikin erilaisilla ominaisuuksilla sekä käytettävyydellä, joihin tässäkin diplomityössä on pyritty kiinnittämään huomiota.

### 3. VAATIMUKSET

Tässä luvussa kerrotaan työlle asetetuista toiminnallisista ja ei-toiminnallisista vaatimuksista. Osa toiminnallisista vaatimuksista on peräisin vanhoilta ja mahdollisesti tulevilta asiakkailta tulleesta kysynnästä, mutta osa on myös Piceasoftilla itse asetettuja tavoitteita. Ei-toiminnalliset ja niistä erityisesti tekniset vaatimukset olivat pääasiassa Piceasoftin asettamia.

#### 3.1 Toiminnalliset vaatimukset

Työlle asetetut toiminnalliset vaatimukset olivat jo alusta asti selvillä, niin asiakkailta tulleen kysynnän kuin kilpailevienkin tuotteiden ominaisuuksien perusteella. Työssä toteutettavan tiedonsiirtosovelluksen tulee tarjota loppukäyttäjälle toiminnallisuudet suoraan mobiililaitteesta toiseen tapahtuvalle siirrolle sekä mahdollistaa laitteen sisällön siirtämisen pilvitalennuspalveluun ja pilvipalveluun jo siirretyn sisällön lataamisen laitteeseen. Tiedonsiirtosovelluksen toiminnalliset vaatimukset loppukäyttäjän kannalta on esitetty käyttötapauskaaviossa kuvassa 1.



*Kuva 1. Toiminnallisten vaatimuksien käyttötapauskaavio*

Työssä haluttiin myös hyödyntää Piceasoftin uusimman OnTheFly-tuotteen myötä käyttöön tullutta QR-koodeja käyttävää teknologiaa. OnTheFly-tuotteissa PiceaOne-mobiilisoluvelluksella luetaan verkkosivulla esitetty QR-koodi, joka sisältää kaiken tarvittavan informaation esimerkiksi laitteen diagnosoinnin aloittamiseksi [34]. Tällöin mobiililaitetta ei tarvitse kytkeä lainkaan tietokoneeseen. Samaa teknologiaa hyödyntäen tavoitteena on, että myös sisällön siirto laitteesta toiseen voitaisiin aloittaa vain QR-koodi lukemalla. Kuvassa 1 esitettyihin kolmeen päätoiminnallisuuteen sisältyy siis myös QR-koodien lukeminen ja tarvittaessa generointi.

Kuvassa 1 esitetyllä tiedonsiirrolla mobiililaitteesta toiseen tarkoitetaan suoraan kahden laitteen välistä tiedonsiirtoa ilman, että dataa välitetään minkään siirtoa kontrolloivan kolmannen laitteen kautta. Tiedonsiirron aloittamisen perustuessa QR-koodeihin, käyttäjä voi aloittaa siirron omatoimisesti näyttämällä toteutetun sovelluksen generoimaa QR-koodia siirron toisena osapuolena olevasta laitteesta ja lukea sen toisella. Vaihtoehtoisesti QR-koodit voitaisiin näyttää myös kolmannella laitteella esimerkiksi tilanteissa, joissa Piceasoftin asiakastahoilla olisi toimipisteissään tietokoneiden sijaan tabletit ja tiedonsiirto haluttaisiin aloittaa nimenomaan tämän kolmannen laitteen avulla.

Kuvassa 1 esitettyjä pilvitalennuspalveluja käyttäviä siirtoja voitaisiin käyttää tilanteissa, joissa käyttäjällä ei ole vielä uutta laitetta hallussaan. Tällöin käyttäjä voi aloittaa vanhan laitteen sisällön siirron pilvipalveluun lukemalla sitä varten tehdyn QR-koodin esimerkiksi Piceasoftin asiakastahon laitteesta tai verkkosivulta. Vastaavasti käyttäjä voi uuden laitteen saatuaan aloittaa tietojen lataamisen sitä varten tehdyn QR-koodin avulla.

Työssä toteutettavassa sovelluksessa käyttäjän valitessa suoraan mobiililaitteesta toiseen tapahtuvan siirron, tulee sen jälkeen olla valittavissa laitteen rooli, siirrossa käytettävä yhteystyyppi sekä siirrettävä sisältö. Laitteen roolin valinnalla määritellään, kumpi laite toimii tiedonsiirrossa tietojen lähettäjänä ja kumpi vastaanottajana. Käytettävä yhteystyyppi voi olla mikä tahansa tuetuista tiedonsiirtotekniikoista kuten Bluetooth, Wi-Fi Direct tai USB. Siirrettävä sisältö tulee olla mahdollista valita eri sisältötyypeittäin kuten ääni, video tai dokumentti. Lähdelaitteena toimivan laitteen tulee voida näyttää QR-koodi, joka sisältää tiedot yhteyden muodostamiseksi ja siirron aloittamiseksi. Roolin valinnan jälkeen kohdelaitteen tulee pystyä lukemaan lähdelaitteesta esitetty QR-koodi.

Käyttäjän valitessa tiedonsiirron mobiililaitteesta pilvitalennuspalveluun, pitää käyttäjän voida valita siirrettävä sisältö myös silloin. Varmuuskopiointiin käytettävä pilvitalennuspalvelu on Piceasoftin aikaisemmin kehittämä konseptipalvelu, jolla on demottu tietojen siirtämistä pilveen PiceaSwitchillä. Palvelun ollessa Piceasoftin oma, siirto sinne voitaisiin periaatteessa aloittaa automaattisesti. Loppukäyttäjä tarvitsee kuitenkin tiedot myös tietojen palauttamiseksi ja tätä varten varmuuskopiointia aloittaessa Piceasoftin palvelu tarvitsee sähköpostin, jonne se lähettää QR-koodin, josta sisällön lataus takaisin laitteeseen voidaan aloittaa. Sähköposti voitaisiin syöttää mobiilisoluvelluksessa ennen siirron aloittamista pilvipalveluun, mutta koska työssä on haluttu ottaa huomioon mahdollisuus

käyttää toisia Piceasoftin palvelua vastaavan rajapinnan toteuttavia palvelimia, olisi siirron tietojen ja käytettävän pilvipalvelun valitsemiseksi helpointa käyttää QR-koodeja jonkin käyttäjän jo identifioineen verkkosivun tai muun sovelluksen kautta.

Myös tietojen lataamiseksi pilvipalvelusta käytetään työssä Piceasoftin omaa pilvipalvelua, jolloin sovelluksen tulee mahdollistaa käyttäjän itse sinne tekemän varmuuskopion palauttaminen. Varmuuskopion tekemisen tavoin sen palauttamisessa tulee ottaa huomioon mahdollisuus laajentaa sovellus käyttämään muita pilvipalveluita. Piceasoftin palvelu toteuttaa jo QR-koodin lähettämisen sähköpostiin käyttäjän siirrettyä tiedot pilvipalveluun, jolloin ensisijainen ratkaisu tietojen latauksen aloittamiseksi on sähköpostiin saadun QR-koodin lukeminen.

Tehtäessä sisällönsiirto Piceasoftin PiceaSwitch-ohjelmistolla, siirrosta tuotetaan sen onnistumisesta kertova raportti. Tällöin, jos työssä toteutettava mobiilisovellus otettaisiin käyttöön Piceasoftin omassa tuotteessa, tulee myös sillä tehdystä siirrosta saada vastaava raportti. Mobiilisovelluksessa tulee olla siis valmius tuottaa siirrosta määrittelyn mukainen JSON-muotoinen raportti, jonka se voi lähettää Piceasoftin tuotteiden tuottamat raportit vastaanottavalle palvelimelle.

## **3.2 Ei-toiminnalliset vaatimukset**

Työllä asetettiin myös ei-toiminnallisia vaatimuksia mm. siirrettävyyden, tietoturvan sekä käytettävyyden suhteen. Ei-toiminnalliset vaatimukset ohjasivat paljon työn toteutuksessa käytettyjä teknologioita, joita käsitellään tarkemmin luvussa 4 sekä itse arkkitehtuurin toteutusta, josta kerrotaan tarkemmin luvussa 5.

### **3.2.1 Siirrettävyys ja laajennettavuus**

Yleisesti siirrettävyydellä tarkoitetaan jo olemassa olevan sovelluksen muokkaamista siten, että se toimii myös toisessa ympäristössä. Tähän liittyy oleellisesti termi järjestelmäriippumattomuus, joka tarkoittaa sovelluksen toimivuutta useammalla alustalla lähes samalla tavalla. Alustalla voidaan tarkoittaa mm. käyttöjärjestelmää. [26]

Vaikka olemassa on useita ohjelmistokehyksiä, jotka mahdollistavat alustariippumattomien sovelluksien kehittämisen sekä Androidille että iOS-mobiilikäyttöjärjestelmälle, päädyttiin työssä toteuttamaan ohjelmisto käyttöjärjestelmäkohtaisesti, jolloin tässä työssä siirrettävyysvaatimuksella tarkoitetaan erityisesti Android-mobiilialustalle toteutettavan ratkaisun tekemistä siten, että se on mahdollisimman pienin muutoksin toteutettavissa myös iOS-käyttöjärjestelmässä.

Siirrettävyysvaatimus liittyy oleellisesti myös työn tavoitteeseen ottaa huomioon asiakastahoilta tullut kysyntä Piceasoftin tuotteiden saamisesta ohjelmistokehityspakettina

(SDK, Software Development Kit), joka haluttiin myös ottaa huomioon työn arkkitehtuuria ja rajapintoja miettiessä. SDK:lla tarkoitetaan joukkoa ohjelmistokehitystyökaluja, kuten ohjelmistokirjastoja, sen dokumentaatiota ja koodinäytteitä, joiden avulla voidaan kehittää sovelluksia tiettyyn tarkoitukseen [40]. SDK mahdollistaisi myös työn oleellisen toiminnallisuuden siirrettävyyden ja sen avulla Piceasoftin asiakkaat voivat käyttää Piceasoftin tuotteiden tarjoamia ominaisuuksia haluamallaan tavalla omissa sovelluksissaan. Tämä vähentäisi myös Piceasoftilla tarvittavia kehitysresursseja, joita menee esimerkiksi tuotteiden brändäykseen asiakkaille.

Määritelmän mukaan siirrettävyyteen voi kuulua myös vaatimus siitä, että siirrettävän toteutuksen tulisi olla halvempaa kuin täysin uuden sovelluksen tekeminen [26]. Teknisten siirrettävyytsvaatimusten lisäksi työssä pyritäänkin myös vähentämään uuden toteutuksen suunnitteluun ja kehittämiseen menevää aikaa ja näin ollen kustannuksia.

Toteutuksen tulee olla myös helposti laajennettavissa ja konfiguroitavissa, sillä mobiilialustojen eroavaisuuksien takia ne voivat sisältää paljon erilaisia teknologioita ja muita ominaisuuksia, joita toisessa alustassa ei ole. Mobiilialustojen tarjoamat ohjelmointirajapinnat eroavat myös niin paljon, että samojen ominaisuuksien toteuttaminen ei välttämättä onnistu molemmissa. Tällöin jonkin ominaisuuden toteuttamista ja käyttöönottoa ei haluta kuitenkaan rajoittaa toisella alustalla, vaikka sen toteuttaminen toisella alustalla ei olisikaan mahdollista. Tällöin näiden eri ominaisuuksien ja teknologioiden tuki sekä käyttöönotto on oltava toteutuksessa erikseen määriteltävissä.

Laajennettavuus on oleellinen vaatimus myös tulevaisuutta silmällä pitäen, sillä tekniikan kehittyessä voi mobiililaitteisiin ja niiden käyttöjärjestelmiin tulla uusia ominaisuuksia, joita tulee voida myös tukea, samalla joitain tekniikoita voi vanheta, niin ettei niitä enää tueta järjestelmätasolla tai niiden käyttämistä ei enää suositella.

Työn kannalta oleellisia laajennettavia tai konfiguroitavissa olevia ominaisuuksia voivat olla mm. laitteistotasolla tulevat uudet tiedonsiirtotekniikat, nykyisten uudet standardit, ohjelmointirajapintoihin tulevat muutokset ja uudet salausmenetelmät. Myös tuki uusien, erilaisien pilvitalennuspalveluiden käyttöönotolle tulee ottaa huomioon. Edellä mainittujen ominaisuuksien lisäämisen pitää olla mahdollisimman helppoa ja mahdollista ilman suuria ohjelman rakenteellisia muutoksia.

Niin siirrettävyys kuin laajennettavuuskin asettavat vaatimuksia toteutuksen arkkitehtuurille ja sen rajapinnoille, joten ne ovat erityisen tärkeää ottaa huomioon jo toteutuksen suunnitteluvaiheessa.

### **3.2.2 Käytettävyys**

Toiminnallisten vaatimusten lisäksi työssä tulee ottaa huomioon toteutettavan ohjelmiston käytettävyys. Käytettävyydellä tarkoitetaan yleisesti jonkin tuotteen kuten ohjelmiston käyttäjien saavuttamaa tarkoituksenmukaisuutta, tehokkuutta ja tyytyväisyyttä, käy-

tettäessä tuotetta johonkin haluttuun tarkoitukseen [29]. Käytettävyys liittyy siis oleellisesti esimerkiksi järjestelmän toiminnallisiin vaatimuksiin määrittelemällä, kuinka hyvin ja tehokkaasti niillä saavutetaan haluttu tavoite ja kuinka tyytyväinen käyttäjä on niiden käyttöön.

Käytettävyydelle voidaan antaa seuraavia ominaisuuksia:

- **Opittavuus:** Kuinka helposti ensimmäistä kertaa tuotetta käyttävä käyttäjä oppii tuotteen perustoiminnot?
- **Tehokkuus:** Kuinka nopeasti tuotteen käytön oppineet käyttäjät pystyvät käyttämään sitä tiettyihin tehtäviin?
- **Muistettavuus:** Kuinka kauan tuotteen käytön oppineilta käyttäjiltä menee palauttaa mieleen sen käyttäminen ja toiminnot.
- **Virheettömyys:** Kuinka paljon ja kuinka vakavia virheitä käyttäjät tekevät ja kuinka helposti niistä toivutaan?
- **Miellyttävyys:** Kuinka miellyttävää tuotteen käyttö on?

Näitä laatuksikomponentteja ja käytettävyyttä yleensäkin arvioidaan usein ohjelmiston käyttöliittymää tarkastellessa, mutta samoja vaatimuksia voidaan arvioida mitä tahansa käyttäjärajapintoja tarkastellessa. [29]

Työssä toteutettavan Android-sovelluksen käyttöliittymän tulee olla mahdollisimman helppokäyttöinen, jonka avulla käyttäjän on mahdollista suorittaa toiminnallisissa vaatimuksissa kuvattuja toimintoja mahdollisimman yksinkertaisesti. Käyttöliittymän ollessa kuitenkin helposti muutettavissa ja toimiessa vain loppukäyttäjän rajapintana tiedonsiirron toteuttavalle ohjelmistolle, on ohjelmiston rajapintojen käytettävyys ja käyttöönoton helppous työn mahdollisessa SDK-käyttötapauksessa sitäkin tärkeämpää.

Otettaessa ohjelmisto käyttöön SDK:na, tulee sen tarjota rajapinnat kaikkien toiminnallisten vaatimuksien toteuttamiseksi, oltava kattavasti ja selkeästi dokumentoitu, ja sen tulee myös toimia dokumentaation mukaisesti. Hyvän käytettävyuden saavuttamiseksi on rajapintojen toteutuksessa myös pyrittävä yhdenmukaisuuteen Piceasoftin muiden mobiilisovellusten rajapintojen kanssa, jolloin sen käyttö on helpommin opittavissa ja muistettavissa.

### 3.2.3 Tietoturva

Tietoturva tai tietoturvallisuus on yksi tärkeimmistä työlle asetetuista ei-toiminnallisia vaatimuksia, sillä nykyään mobiililaitteet voivat sisältää todella paljon käyttäjän henkilökohtaista ja mahdollisesti arkaluontoista dataa kuten videoita, kuvia sekä dokumentteja. Tällaisen data siirtäminen tekee tietoturvallisuudesta yhden työn keskeisimmistä ja jous-tamattomimmista vaatimuksista.

Tietoturvallisuus voidaan jakaa seuraaviin osiin:

- Luottamuksellisuus (engl. confidentiality)
- Saatavuus (engl. availability)
- Eheyys (engl. integrity)

Luottamuksellisuudella tarkoitetaan sitä, että tietoa voi käsitellä ja lukea vain ne tahot, joilla on siihen oikeus. Tässä työssä käyttäjän yksityistä dataa siirrettäessä onkin erityisen tärkeää varmistaa sen luottamuksellisuus. Tiedon luottamuksellisuutta voidaan parantaa mm. salauksella ja pääsynhallinnalla. [1]

Saatavuudella tarkoitetaan sitä, että käytettävä järjestelmä tai palvelu on käytettävissä silloin kun sitä tarvitaan [1]. Tässä työssä saatavuus liittyy erityisesti laitteen sisällön varmuuskopioinnin ja varmuuskopion lataamisen mahdollistavaan pilvipalveluun.

Eheydellä tarkoitetaan tiedon oikeellisuutta eli sitä, että käsiteltävä tai säilytettävä tieto pysyy virheettömänä ja muuttumattomana niin, että mikään ulkopuolinen tekijä ei ole tahallisesti tai vahingossa muuttanut sen sisältöä [1]. Tiedon luottamuksellisuuden tavoin, myös sen eheyden varmistaminen on erityisen tärkeää käyttäjän tietoja siirtäessä. Tiedon eheyttä voidaan edesauttaa mm. salausmenetelmillä ja käyttämällä sellaisia tiedonsiirto-protokollia, joissa on virheen korjaus- ja tunnistusmekanismeja. Lopullinen tiedon eheys voidaan varmistaa laskemalla siitä tiiviste eli tarkistussumma. Tiedon eheyden varmistamisesta ja tarkistussumman laskemisesta on kerrottu tarkemmin luvussa 4.5.

Jo olemassa olevan pilvipalvelun saatavuus riippuu erityisesti palvelun infrastruktuuriin, johon ei tässä työssä oteta kantaa. Tässä työssä tietoturvan tärkeimpiä osa-alueista ovatkin tiedon luottamuksellisuus ja eheys, niiden liittyessä erityisesti mobiililaitteesta toiseen tehtävään tiedonsiirtoon. Näiden osalta työssä vaaditaan, että kaikki tiedonsiirto tehdään oletusarvoisesti nykyiset tietoturvasuosittukset täyttäviä salattuja yhteystyyppejä ja tiedonsiirto-protokollia käyttäen, sekä tarvittaessa tietojen tulee olla vielä erikseen salattavissa. Tiedon luottamuksellisuuden lisäksi sen eheys tulee oletusarvoisesti varmistaa käyttäen suosittujen mukaisia tarkistussummia. Nämä tietoturva-vaatimukset on voitu varmistaa erilaisilla turvallisuuteen liittyvillä teknologiavalinnoilla, joista kerrotaan tarkemmin luvussa 4.

## 4. KÄYTETYT TEKNOLOGIAT

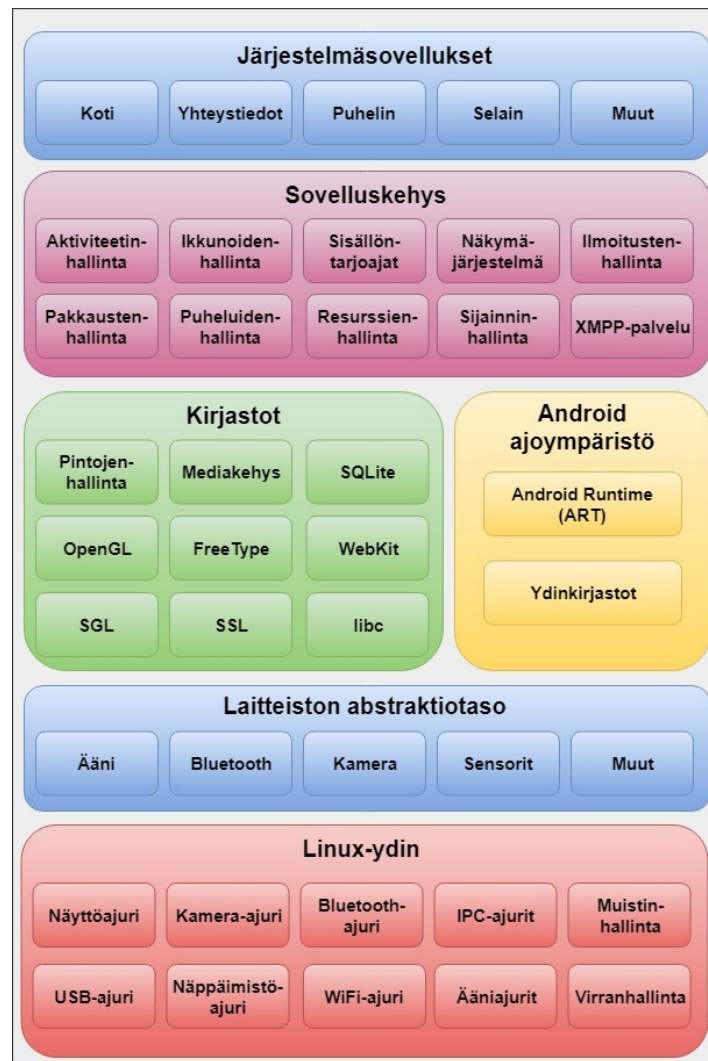
Tässä luvussa kerrotaan tarkemmin diplomityön toteutukseen valituista teknologioista, kuten mobiililaitteiden välille tiedonsiirron suoraan laitteesta toiseen mahdollistavista tiedonsiirtomenetelmistä ja -protokollista, sekä siirron turvallisuuteen ja tiedon eheyteen liittyvistä teknologioista.

### 4.1 Android

Android on alun perin Android Inc.:n puhelimille ja muille mobiililaitteille kehittämä avoimeen lähdekoodiin perustuva ohjelmistopino, johon sisältyy käyttöjärjestelmän lisäksi eri tasoilla toimivia väliohjelmistoja sekä loppukäyttäjän perusohjelmia. Androidin kehitys siirtyi Googlelle sen ostettua Android Inc.:n vuonna 2005, mutta nykyisin sen kehittämisestä vastaa Open Handset Alliance (OHA), johon kuuluu Google mukaan lukien kaikkiaan 84 laitteisto- ja ohjelmistovalmistajaa sekä teleoperaattoria [30].

Androidin perustuessa avoimeen lähdekoodiin sen perustana on avoimen lähdekoodin GPLv2-lisensioitua Linux-käyttöjärjestelmäydin sekä muita avoimen lähdekoodin ohjelmistoja [2][10]. Google on myös julkaissut suurimman osa Androidin lähdekoodista avoimen lähdekoodin Apache 2.0 -lisenssillä [2]. Vaikka Android kehitettiin alkujaan ensisijaisesti kosketusnäytöllisille mobiililaitteille, on Google kehittänyt siitä myös erikseen televisioille, autoille ja älykelloille räätälöidyt versiot.

Android järjestelmän kokonaisuudessaan muodostava ohjelmistopino ja sen eri tasojen sisältämät pääkomponentit ovat esitelty kuvassa 2. Ohjelmistopino koostuu kaikkiaan viidestä tasosta, joista jokaisella on oma ensisijainen käyttötarkoituksensa.



**Kuva 2.** Android-ohjelmistopinon rakenne [10]

Ylimmällä eli niin sanotulla sovellustasolla sijaitsevat perustoiminnallisuuksia tarjoavat sovellukset kuten yhteystiedot, SMS-viestit, puhelin ja Internet-selain. Näiden sovelluksen avulla ohjelmistokehittäjien ei tarvitse itse toteuttaa toiminnallisuuksia kuten SMS-viestien lähetystä omiin sovelluksiinsa. Nämä järjestelmäsovellukset toimivat poikkeuksia, kuten esimerkiksi järjestelmän Asetukset-sovellusta, lukuun ottamatta, kuin käyttäjän asentamat kolmannen osapuolen sovellukset. Tämä sovelluksista muodostuva taso onkin se millä käyttäjät pääosin toimivat. [10]

Sovelluskehitys Androidille tapahtuu pääasiassa Java-kielellä ja edellä mainitut sovellukset ovatkin suurimmaksi osaksi sillä kirjoitettu. Vuonna 2017 Android sai täyden tuen myös Kotlin-ohjelmointikielille. Kotlin on täysin yhteensopiva Javan tavukoodia suorittavan Java-virtuaalikoneen kanssa. Tällöin sillä kirjoitettu ohjelma toimii Androidissa täysin Javalla kirjoitetun ohjelman tavoin [14].

Toiseksi ylimmällä eli sovelluskehityksen tasolla sijaitsevat Android-käyttöjärjestelmän tarjoamat Java-ohjelmointikielellä kirjoitetut ohjelmointirajapinnat (API). Nämä rajapinnat ja niiden avulla käytettävät järjestelmäkomponentit ja palvelut määrittelevät sovelluksen perusrakenteen. Android-sovellusten kehittämisen mahdollistavia komponentteja ovat esimerkiksi näkymäjärjestelmä, joka mahdollistaa sovellusten käyttöliittymien rakentamisen sekä sovelluksen resurssien, elinkaaren ja ilmoitusten hallinnan tarjoavat komponentit. [10]

Ohjelmistopinin kirjastotasolla sijaitsee C/C++-kielellä kirjoitettuja natiiveja kirjastoja esimerkiksi ääneen ja grafiikkaan liittyen. Useita näistä alemman tason kirjastoista voidaan käyttää sovelluskehityksen tasolla olevien Java-luokkien läpi, mutta joitain niistä voidaan käyttää myös suoraan Androidin natiivikehityspaketin (NDK) avulla. [10]

Natiivikirjastojen kanssa samalla tasolla sijaitsee Android ajoympäristöön liittyvät komponentit. Nämä komponentit ovat ydinkirjastot ja Android Runtime (ART) -ympäristö. Ydinkirjastot tarjoavat Androidille Java-kielen toiminnallisuudet, joita sovelluskehityksenkin tarjoama API käyttää. Android Runtime toteuttaa esimerkiksi Java-lähdekoodista käännetyn tavukoodin kääntämisen laitteistokohtaiseksi natiivikoodiksi sovelluksen asennusvaiheessa. [10]

Laitteiston abstraktiotaso (HAL) sisältää niin ikään C/C++-kielellä kirjoitettuja kirjastoja, jotka mahdollistavat laitteistokomponenttien, kuten kameran tai Bluetooth-moduulin, käyttämisen korkeamman tason Java-rajapintojen avulla. [10]

Alimmalla tasolla toimii Linux-käyttöjärjestelmäydin, johon Androidin ajoympäristökin nojaa. Linux-ydin tarjoaa esimerkiksi laitteistokomponenttien käyttämiseen tarvittavat ajurit sekä toiminnallisuudet säikeistykselle ja muistinhallinnalle. [10]

### **Käyttäjän tietojen tallentaminen ja käyttäminen**

Tiedonsiirtosovelluksen toimintaan liittyy oleellisesti, kuinka Android tallentaa käyttäjän tietoja ja kuinka niitä voidaan sovelluksessa käsitellä. Android-järjestelmässä kaikki käyttäjän tiedot ovat tallennettuna laitteen tiedostojärjestelmään. Tiedonsiirtosovelluksen kannalta tiedot voivat olla tallennettu järjestelmään yksittäisinä tiedostoina tai tietokantojen muodossa, joko sisäiseen tai ulkoiseen muistiin. [13]

Sisäisellä muistilla tarkoitetaan sovelluksen täysin yksityistä muistia, jota vain se voi käyttää. Ulkoiseen muistiin tallennetut tiedostot voivat olla julkisia tai yksityisiä. Julkiset tiedostot ovat kaikkien sovelluksien sekä laitteen käyttäjän käytettävissä. Yksityiset tiedostot ovat myös kaikkien nähtävissä, mutta jonkin sovelluksen omistamia, ja ne poistetaan tiedostojärjestelmästä, kun sovellus poistetaan. Käyttäjän musiikki-, kuva-, video- ja dokumenttiedostojen ollessa pääasiassa tallennettuna ulkoiseen muistiin, voidaan niitä käyttää Androidin tarjoaman File-ohjelmointirajapinnan kautta kuin mitä tahansa muitakin tiedostoja. [17]

Sisältötyypeistä kontaktit, kalenteri, viestit ja kirjanmerkit ovat Android-järjestelmässä tallennettuna tietokantoihin. Näistä tietokannoista vastaa oletusarvoisesti Androidin omat järjestelmäsovellukset. Android tarjoaa sovelluskehityksen tasolla sisällöntarjoaja-ohjelmointirajapinnan (Content provider), joka mahdollistaa sovelluksien hallinnoida pääsyä omiin tai muiden sovelluksien tietoihin. Sisällöntarjoaja muodostaa yhtenäisen ja turvallisen rajapinnan sovelluksen tietojen käsittelemiseksi, olivat tiedot järjestelmässä sitten yksittäisinä tiedostoina tai tietokantojen muodossa. Sisällöntarjoaja on siis aina kunkin sisältöä tallentavan sovelluksen itse toteuttama. Androidin järjestelmäsovelluksien toteuttaessa omat sisällöntarjoajansa, voivat muut sovellukset käsitellä niiden avulla esimerkiksi järjestelmässä olevia kontakteja ja viestejä, kunhan käyttäjä on myöntänyt niihin tarvittavat oikeudet. [11] Multimediatiedostoihin liittyvää metadataa voidaan käsitellä myös sisällöntarjoajien avulla, mutta tiedostojen raakadatan käsittely vaatii File-rajapinnan käyttöä.

## 4.2 Tiedonsiirtotekniikat

Tässä luvussa kerrotaan työssä käytetyistä tiedonsiirtotekniikoista, jotka mahdollistavat mobiililaitteiden välisen yhteyden muodostamisen tiedonsiirtoa varten.

### 4.2.1 Bluetooth

Bluetooth on avoimen standardin langaton tiedonsiirtotekniikka, joka mahdollistaa laitteiden välisen tiedonsiirron lähietäisyydellä. Bluetoothin toimii 2,4 GHz taajuusalueella, jossa myös suuri osa Wi-Fi -standardin laitteista toimii. Tarkemmin Bluetooth kuitenkin vaihtelee lähetystaajuutta 2,4000 - 2,4835 GHz välillä, jossa taajuuden hyppyalue on  $2\,402 + k$  MHz ja  $k = 0 \dots 78$  [47]. Taajuuden vaihtelulla pyritään vähentämään häiriöitä tiloissa, joissa on käytössä useita Bluetooth tai muita 2,4 GHz taajuudella toimivia laitteita. Bluetooth-yhteyden käyttöetäisyyden määrittelee siinä käytettävän lähettimen teholuokka. Mahdolliset teholuokat ja niiden käyttöetäisyydet on esitelty taulukossa 1. Vähävirtaisuutta suosivissa mobiililaitteissa on päädytty käyttämään yleisintä teholuokkaa 2, jolloin mobiililaitteiden välinen käyttöetäisyys jää yleisesti noin 10 metriin [32].

*Taulukko 1. Bluetoothin teholuokat ja niiden käyttöetäisyydet [32]*

Luokka	Suurin sallittu teho (mW)	Käyttöetäisyys (m)
1	100	~ 100
2	2,5	~ 10
3	1	~ 1

Bluetoothin eduiksi voidaankin katsoa sen virrankulutus sekä yleisyys, sillä se löytyy nykyään käytännössä jokaisesta älypuhelimesta sekä tabletista käyttöjärjestelmään katsomatta. Vaikka Bluetoothia käytetään perinteisesti kahden laitteen väliseen tiedonsiirtoon, mahdollistaa tekniikka jopa kahdeksan laitteen piconet-verkon muodostamisen, jossa yksi laite toimii isäntänä loppujen toimiessa orjina. [47]

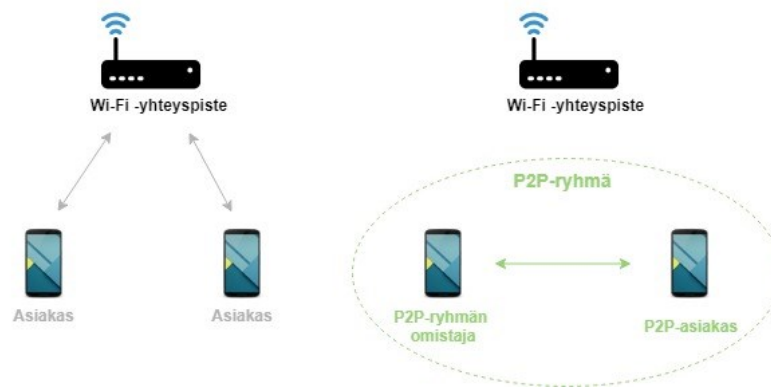
Versiosta riippuen perinteisen Bluetooth-standardin määrittelemä tiedonsiirtonopeus on noin 1Mbps - 3Mbps. Vuonna 2009 julkaistu Bluetooth 3.0 + HS (High Speed) standardi toi kuitenkin mukanaan 802.11 PAL (Protocol Adaption Layer) ominaisuuden, joka mahdollistaa 802.11-standardin, eli tutummin WLAN-yhteyden, käyttämisen tiedonsiirron apuna. WLAN-yhteyden käyttäminen mahdollistaa jopa 24Mbps:n siirtonopeuden. Vuonna 2016 julkaistu Bluetooth 5 standardi tuplasi aiemmat tiedonsiirtonopeudet, jolloin WLAN-yhteyttä hyödyntävän Bluetooth-yhteyden siirtonopeus on jopa 48Mbps. [47]

Vuonna 2010 julkaistu Bluetooth 4.0 toi perinteisen ja suuren nopeuden (HS) -protokollien lisäksi uuden matalaenergian LE (Low Energy) -protokollan, joka mahdollistaa entistäkin pienemmän virrankulutuksen, kuitenkin tiedonsiirtonopeuden kustannuksella. Bluetooth LE standardia ei olekaan ensisijaisesti tarkoitettu käytettäväksi suurien, esimerkiksi mobiililaitteista toisiin siirrettävien datamäärien siirtämiseen vaan enemmänkin pienien datamäärien siirtämiseen vähävirtaisissa, kuten paristolla toimivissa laitteissa.

Bluetooth-yhteydellä siirrettävä data on salattu käyttäen joko E0-jonosalausta, tai 128-bittistä AES-lohkosalausta. Tiedonsiirtoon riittävän turvalliseksi todettua 128-bittistä AES-salausta käytetään Bluetooth 4.0 standardia uudemmissa, perinteistä protokollaa käyttävissä yhteyksissä sekä kaikissa Bluetooth LE yhteyksissä [32]. Vanhemmissa standardeissa käytettävä E0-salaus on todettu tiedonsiirtoon riittämättömäksi, jolloin siirrettävä tieto on suositeltavaa salata ensin sovellus-tasolla [32]. Sovellus-tason salaukseen soveltuukin jo mainittu AES-salausalgoritmi, josta kerrotaan tarkemmin luvussa 4.4.2.

## 4.2.2 Wi-Fi Direct

Wi-Fi Direct, tunnetaan myös nimellä Wi-Fi P2P, on Wi-Fi -tekniikkaan perustuva langaton verkkoteknologia, joka mahdollistaa suoran laitteiden välisen yhteyden muodostamisen ilman erillistä Wi-Fi -yhteyspistettä (engl. Access Point). Tekniikan avulla voidaan luoda kahden laitteen välinen yhteys tai yhdestä laitteesta useampaan laitteeseen suuntautuva yhteys. [45] Kuvassa 3 on esitetty perinteisen Wi-Fi -verkon, jossa laitteet kommunikoiivat aina yhteyspisteen kautta, ero verrattuna Wi-Fi Direct -verkkoon.



**Kuva 3.** Perinteinen Wi-Fi verkko verrattuna Wi-Fi Direct verkkoon [46]

Kuten kuvasta 3 nähdään, WiFi Direct -verkon laitteet, jotka tunnetaan paremmin P2P-laitteina, muodostavat P2P-ryhmän (engl. P2P Group). Tämä P2P-laitteiden muodostama ryhmä on toiminnallisuudeltaan verrattavissa perinteiseen Wi-Fi -yhteyspisteen muodostamaan verkkoon. Ryhmä muodostuu yhteyspistettä vastaavassa roolissa toimivasta P2P-ryhmän omistajasta (engl. Group Owner) sekä asiakaslaitteina toimivista P2P-asiakkaista. [46]

P2P-ryhmässä laitteiden roolit eivät ole mitenkään ennalta määritellyt, vaan ne voivat ryhmää muodostaessaan neuvotella kumpi toimii ryhmän omistajana ja kumpi asiakkaana. Ryhmän muodostamisen jälkeen muutkin laitteet voivat liittyä ryhmään asiakaslaitteina aivan kuten perinteiseen Wi-Fi -verkkoon yhdistäessä. Wi-Fi Direct tukee myös niin sanottujen *legacy*-laitteiden eli laitteiden, jotka eivät tue Wi-Fi Directiä, liittymistä ryhmään, mutta tällöin ryhmän omistaja toimii niiden kannalta perinteisenä Wi-Fi -yhteyspisteenä, eivätkä ne voi käyttää Wi-Fi Directille määriteltyjä ominaisuuksia. [46]

P2P-ryhmän muodostaminen voi tapahtua eri menetelmin riippuen siitä tarvitseeko laitteiden neuvotella ryhmän omistajuudesta tai että onko laitteilla jotain yhteyden turvallisuuden liittyviä määrittelyjä jo tiedossa [3]. Normaalissa tapauksessa, missä kumpikaan laite ei ole jo muodostanut P2P-ryhmää eivätkä ne ole vaihtaneet suojaukseen liittyviä tietoja, yhteyden muodostus lähtee siitä, että laitteet suorittavat perinteisen Wi-Fi -verkkojen skannauksen, jolla voidaan löytää jo olemassa olevia P2P-ryhmiä ja perinteisiä Wi-Fi -verkkoja. Skannauksen jälkeen laitteet aloittava löytämävaiheen (engl. Discovery) käyttäen Wi-Fi taajuusalueen kanavia 1, 6 ja 11. Tässä vaiheessa kumpikin laite vuorottelee kahden tilan välillä: etsintätilan (engl. Search state), jossa laite etsii aktiivisesti toista laitetta näiltä kanavilta ja kuuntelutilan (engl. Listen state), jossa laite taas kuuntelee niihin kanaviin tulevia pyyntöjä [3][46].

Kun laitteet ovat löytäneet toisensa, ne aloittavan kolmitiekättelyllä tapahtuvan P2P-ryhmän omistajan neuvottelun (engl. Group Owner Negotiation phase). Tässä vaiheessa laitteet päättävät kumpi niistä toimii ryhmän omistajana ja millä 2,4 tai 5 Ghz taajuusalueella

toimivalla kanavalla ryhmä tulee toimimaan. Lopuksi laitteet vielä varmistavat turvallisen kommunikoinnin WPS-toimintoa (Wi-Fi Protected Setup) käyttäen, jonka jälkeen ne vaihtavat DHCP-tietoja IP-asetusten asettamiseksi. [3][46]

WPS-toiminnon avulla turvallisen yhteyden muodostaminen onnistuu vähäisellä käyttäjältä vaadittavalla vuorovaikutuksella, sillä se mahdollistaa suojatun yhteyden muodostamisen esimerkiksi vain P2P-asiakaslaitteeseen tulevaan PIN-kyselyyn vastaamalla. WPS muodostuu kahdesta vaiheesta: ensimmäisessä vaiheessa P2P-ryhmän omistaja luo ja välittää salausavaimet asiakaslaitteelle, jonka jälkeen toisessa vaiheessa P2P-asiakas eroaa ja liittyy uudelleen ryhmään saamiaan tietoja käyttäen. Tällöin jos asiakaslaite olisi jo tiennyt käytetyt salausavaimet, olisi WPS-vaiheessa riittänyt vain toisen vaiheen suorittaminen. WPS:n käyttäessä langattomien lähiverkkojen viimeisintä tietoturvastandardia, WPA2 (Wi-Fi Protected Access II) -salausprotokollaa, ja AES-salausta, voidaan Wi-Fi Direct yhteyttä pitää pääsääntöisesti turvallisena. [3][45][46]

Wi-Fi Direct -sertifioitujen laitteiden välinen käyttöetäisyys ja tiedonsiirtonopeus määräytyvät täysin käytettävän Wi-Fi, eli IEEE 802.11 -standardin mukaan ja ne toimivat kuin mikä tahansa Wi-Fi -laite vastaavassa verkossa [45]. Käytetyn standardin mukaan yhteyden teoreettinen käyttöetäisyys voi olla joistakin kymmenistä metreistä jopa yli 200 metriin [44][45]. Taulukossa 2 on esitetty yleisimmät mobiililaitteiden tukemat standardit sekä niiden teoreettiset tiedonsiirtonopeudet.

**Taulukko 2.** *Mobiililaitteissa yleisimpien 802.11-standardien tiedonsiirtonopeudet [44]*

IEEE-standardi	Taajuus (GHz)	Teoreettinen enimmäisiirtonopeus
<b>802.11a</b>	5	54 Mbps
<b>802.11b</b>	2,4	11 Mbps
<b>802.11g</b>	2,4	54 Mbps
<b>802.11n</b>	2,4 / 5	150 Mbps
<b>802.11ac</b>	5	450 Mbps

Taulukossa esitetyt nopeudet ovat teoreettisia enimmäisnopeuksia, kun lähetukseen että vastaanottoon käytetään samanaikaisesti vain yhtä antennia. Esimerkiksi 802.11n ja 802.11ac standardit voivat toimia käyttäen jopa kolmea antennia yhtä aikaa, jolloin teoreettiset enimmäisnopeudet olisivat 450 Mbps ja 1,3 Gbps [4][44]. Taulukosta nähdään myös Wi-Fi -verkkojen toimivan pääasiassa joko Bluetoothin kanssa samalla 2,4 GHz

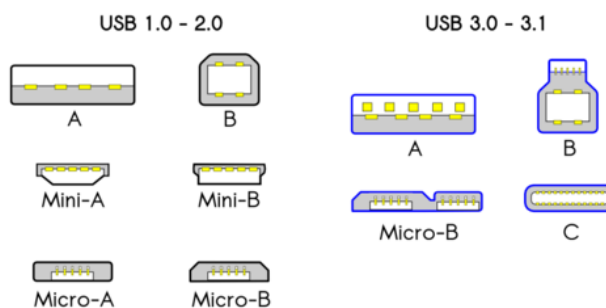
tai uudemmissa standardeissa yleistyneellä 5 GHz taajuudella. Teoreettiset arvot määräävän käytetyn standardin lisäksi yhteyden lopulliseen käyttöetäisyyteen ja tiedonsiirtonopeuteen vaikuttavat yleisesti käytetyn lähetystaajuuden lisäksi käytetty kaistanleveys, laitteisto, antennien lukumäärä sekä ympäristö ja sen mahdolliset häiriötekijät [45].

Wi-Fi Direct on ollut tuettuna Android-käyttöjärjestelmän 4.0 versiosta lähtien ja nykyään se kilpaileekin ominaisuuksiensa puolesta mobiililaitteiden väliseen langattomaan tiedonsiirtoon jo lähes standardiksi muodostuneen Bluetooth-tiedonsiirtotekniikan kanssa [19].

### 4.2.3 USB On-The-Go

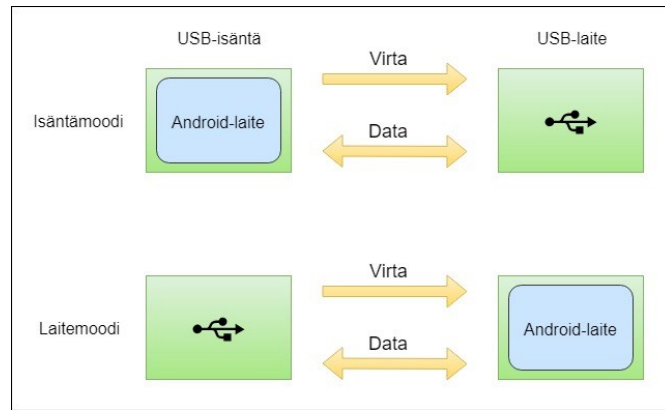
USB OTG (On-The-Go) on vuonna 2001 silloiseen USB 2.0 -standardiin tehty lisäys, joka mahdollistaa USB-laitteiden toimivan sekä isännän että oheislaitteen roolissa. Perinteinen USB-arkkitehtuuri on toiminut isäntä/orja -periaatteella (engl. master/slave), jossa toinen laite on toiminut aina isännän ja toinen orjan eli oheislaitteen roolissa. Yleisesti tietokoneet ovat toimineet aina isäntinä, kun niihin kytkettävät laitteet kuten tulostimet ja puhelimet ovat toimineet orjina. Mobiililaitteiden toteuttaessa aina orjan roolin, ei USB-väylän käyttäminen kahden mobiililaitteen väliseen tiedonsiirtoon ollut mahdollista ennen OTG-standardia. [20]

USB OTG -standardissa määritellään USB-laitteelle kaksi roolia; A-laite ja B-laite. Perinteisesti rooli on määrittynyt sen mukaan, kumpaan päähän USB-kaapelia laite on kytketty. Kuvassa 4 esitetyt USB-liitintyyppien nimet kertovat kummassa roolissa kyseiseen liittimeen kytketty laite oletuksena toimii. Oletuksena A-laite toimii isäntänä ja antaa virtaa USB-väylälle, kun B-laite toimii oheislaitteen roolissa [20].



**Kuva 4.** USB-liitintyyppit [6][21]

OTG-standardissa määritellyn Host Negotiation Protocol:n (HNP) avulla laitteet voivat vaihtaa isännän ja oheislaitteen rooleja keskenään. Laitteiden vaihtaessa roolejaan, virtaa antava laite kuitenkin pysyy samana [20]. Androidissa molemmat roolit ja niiden vaihtaminen laitteiden välillä ovat olleet tuettuna jo käyttöjärjestelmän 3.1 versiosta lähtien [18]. Kuvassa 5 on esitetty isäntä- ja laitemoodien toiminta Android-laitteen ja oheislaitteen välillä, mutta tilanne toimii vastaavasti kahden Android-laitteen välillä rooleista riippumatta.



**Kuva 5.** Androidin USB-moodit [18]

USB OTG -standardin ollessa vain lisäys perusstandardiin, noudattaa se täysin USB-standardien mukaisia tiedonsiirtonopeuksia [20]. Taulukossa 3 on esitetty joitakin USB-standardeja ja niiden siirtonopeuksia. Nopeuksia vertaillaessa tulee kuitenkin ottaa huomioon niiden olevan teoreettisia ja riippuvan paljon käytettävästä laitteistosta ja USB-kaapelista.

**Taulukko 3.** Yleisimmät USB-standardit ja niiden teoreettiset siirtonopeudet [6][21]

USB-standardi	Teoreettinen enimmäissiirtonopeus
USB 1.1	12 Mbps
USB 2.0	480 Mbps
USB 3.1	5 Gbps

Taulukon standardeista USB 2.0 ja USB 3.1 ovat yleisimmät mobiililaitteissakin tuetut standardit, 2.0:n ollessa kuitenkin vielä paljon yleisempi 3.1:n ollessa tuettu pääasiassa vain uusissa korkeamman hintaluokan mobiililaitteissa.

### 4.3 Tiedonsiirtoprotokollat

Tässä luvussa kerrotaan työssä käytetyistä tiedonsiirtoprotokollista, jotka toteuttavat tiedonsiirron sen jälkeen, kun mobiililaitteiden välille on luotu yhteys jollakin 4.2 luvussa esitellyistä tekniikoista tai kun tietoa halutaan siirtää mobiililaitteen ja pilvipalvelun välillä.

#### 4.3.1 TCP

Transmission Control Protocol (TCP) on tietoliikenneprotokolla, jolla tietokoneet voivat lähettää luotettavasti tietoa toisilleen. Suurin osa Internetin liikenteestä perustuu TCP-protokollaan sen ollessa käytössä sovelluksissa kuten WWW, sähköposti ja FTP. [33]

TCP on yhteydellinen protokolla, joka tarkoittaa sitä, että lähettävän ja vastaanottavan sovellusten välille avataan looginen yhteys ennen varsinaisen tiedonsiirron aloittamista, jonka jälkeen tietoa käsitellään vain näissä verkon päätepisteissä. Sovellustasolla luotua yhteyttä käsitellään sokettien (engl. socket) avulla, jotka toimivat verkkokommunikaation päätepisteinä abstrahoiden varsinaisen kommunikaation toteutuksen. [33]

TCP-yhteys laitteiden välillä koostuu kolmesta vaiheesta: yhteyden muodostamisesta, tiedonsiirrosta ja yhteyden katkaisemisesta. Yhteyden muodostaminen tapahtuu kolmitiekättelyllä, jossa yhteyden aloittaja lähettää ensiksi yhteydenmuodostuspyynnön SYN-paketilla (engl. synchronization). Tämän saatuaan vastaanottaja vastaa aloittajalle SYN/ACK-paketilla (engl. acknowledgement). SYN/ACK-paketin vastaanotettuaan aloittajalaite vastaa vielä ACK-paketilla merkiksi paketin vastaanottamisesta. Kolmitiekättelyn jälkeen laitteet voivat aloittaa tiedonsiirron. [33]

Tiedonsiirtovaiheessa monet TCP-protokollan mekanismit varmistavat luotettavan tiedonsiirron protokollatasolla eikä sovellusten ohjelmoijien tarvitse kiinnittää huomiota siellä esiintyviin ongelmiin. TCP-paketit mm. sisältävät järjestysnumeron, jonka avulla voidaan varmistaa vastaanotettujen pakettien käsittely oikeassa järjestyksessä ja mahdollisten kaksoiskappaleiden hylkääminen. Protokollassa on mekanismit myös hävinneiden TCP-pakettien tunnistamiseksi ja niiden uudelleenlähettämiseksi, sillä jokaisesta vastaanotetusta paketista lähetetään kuittaus (ACK) lähettäjälle. Jos kuittausta ei tule tietyssä ajassa, paketti lähetetään uudestaan. [33]

Siirrettävän tiedon oikeellisuuden varmistamiseksi TCP-paketti sisältää myös lähettäjän koko paketille laskeman tarkistussumman. Jos vastaanottajan laskema tarkistussumma eroaa paketissa ilmoitetusta, paketti hylätään eikä kuittausta lähetetä. Tämän seurauksena lähettäjä lähettää paketin uudestaan tietyn odotusajan jälkeen. TCP-protokollan käyttämä tarkistussumma on kuitenkin riittämätön kokonaisten tiedostojen oikeellisuuden varmistamiseksi. Laskemalla tarkistussummat kuitenkin sovellustasolla voidaan varmistaa kaiken tiedon siirtyneen lopulta oikein. Tarkistussummien laskennasta sovellustasolla on kerrottu tarkemmin luvussa 4.5.

TCP-yhteyden päättäminen tapahtuu pääsääntöisesti nelitiekättelyllä, jossa molemmat osapuolet katkaisevat yhteyden lähettämällä lopetus eli FIN-paketin (engl. finish) ja kuittaamalla toistensa paketit ACK-paketilla. Yhteyden päättäminen onnistuu myös kolmitiekättelyllä kuten yhteyden avaaminen, päättäessä SYN- ja SYN/ACK-paketit on vain korvattu FIN- ja FIN/ACK-paketeilla.

Luotettavan tiedonsiirron ja tiedon eheyden varmistavista mekanismeista huolimatta TCP-protokolla ei ole sellaisenaan täysin tietoturvallinen, sillä yhteydellä siirrettävä tieto on täysin salaamatonta. Tämä ongelma voidaan kuitenkin ratkaista esimerkiksi käyttämällä TLS-protokollaa, josta on kerrottu tarkemmin luvussa 4.4.1.

### 4.3.2 HTTP ja HTTPS

Hypertext Transfer Protocol (HTTP) on sovellustasolla toimiva protokolla, joka luotiin alun perin hypertekstin siirtämiseen selaimien ja WWW-palvelimien välillä. Nimestään huolimatta HTTP-protokollalla voidaan kuitenkin siirtää mitä vain tietoa. [7]

HTTP-protokollaa käytetään asiakas-palvelin -mallissa, jossa asiakas ja palvelin keskustelvat pyyntö-vastaus -periaatteella. Esimerkiksi selaimen ja WWW-palvelimen välisessä kommunikaatiossa selain voi lähettää pyynnön palvelimelle, joko lähettääkseen sille tietoja tai hakeakseen siltä joitakin resursseja, kuten HTML-sivun tai binääridataa. Käsiteltävään pyynnön palvelin lähettää selaimelle vastauksen tietojen onnistuneesta vastaanottamisesta tai pyydetyn resurssin. [7]

HTTP toimii pääasiallisesti TCP-protokollan avulla eli asiakassovellus avaa aina ensin TCP-yhteyden palvelimelle, jonka jälkeen se voi lähettää sille pyynnön HTTP-protokollaa käyttäen. HTTP-protokolla tuo siis jälleen uuden abstraktiotason, joka sisäisesti toteuttaa TCP-yhteyden hallitsemisen, jolloin sovellusohjelmoijan tarvitse sitä tehdä. HTTP-protokollassa asiakkaan ja palvelimen välinen yhteys on voimassa vain siirtotahtuman ajan. [7]

HTTP-protokollassa tiedonsiirron tapahtuessa salaamattomasti, on se haavoittuvainen esimerkiksi salakuuntelulle ja mies välissä -hyökkäykselle (engl. man-in-the-middle attack). Hypertext Transfer Protocol Secure (HTTPS) on tiedon suojattuun siirtoon tarkoitettu HTTP-protokollan ja TLS-salausprotokollan yhdistelmä, jota käytettäessä verkkosivu tarvitsee varmenteen (engl. certificate), jolla se voi todistaa olevansa juuri se palvelin, jonka kanssa halutaan asioida. [38]

HTTPS-protokollassa asiakassovelluksen ja palvelimen välille avataan ensin TCP-yhteys, jonka jälkeen luodaan suojattu yhteys TLS-salausprotokollaa käyttäen. Salausprotokolla toimii siis TCP-yhteyden päällä. [31] Salatun yhteyden luonnin aikana sovitaan sen aikana käytettävistä salausavaimista. Salatun yhteyden luonnin jälkeen sen sisällä HTTP-protokollalla lähetettävät tiedot asiakkaan ja palvelimen välillä on salattu. TLS-salausprotokollasta on kerrottu tarkemmin luvussa 4.4.1.

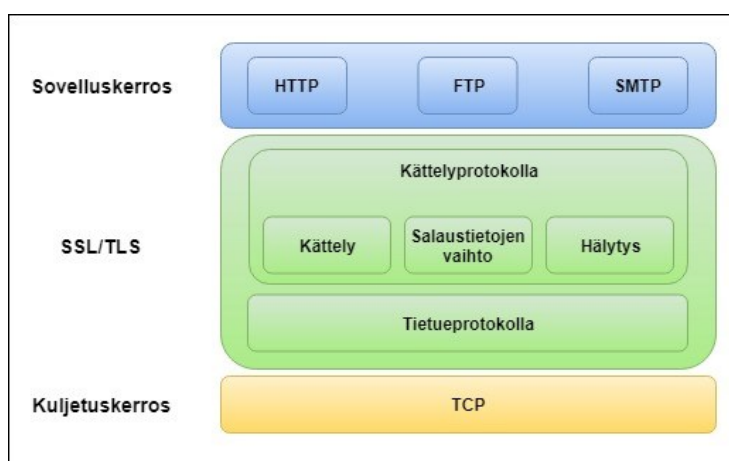
## 4.4 Salaus

Työn turvallisuuteen liittyvistä teknologioista keskeisimpiä ovat siirrettävän sisällön salaukseen käytetyt menetelmät, joilla pyritään varmistamaan tiedonsiirron luottamuksellisuus. Työssä käytetyt salausmenetelmät ovat protokollatasolla toimiva TLS -salausprotokolla sekä sovellustasolla käytetty AES-salausalgoritmi.

#### 4.4.1 SSL/TLS

Transport Layer Security (TLS) ja sen edeltäjänä toiminut Secure Sockets Layer (SSL) ovat salausprotokollia, jotka mahdollistavat salatun tiedonsiirron verkon päätelaitteiden välillä. Netscape aloitti edeltäjänä toimineen SSL-protokollan kehityksen, mutta luovutti sen IETF-standardointiorganisaatiolle, joka kehitti siitä oman Transport Layer Security (TLS) protokollaksi nimetyn versionsa. Näin ollen TLS-protokollan 1.0 versiosta tuli vuonna 1996 julkaistun SSL 3.0:n seuraaja. [31]

SSL-protokolla kehitettiin alkujaan HTTP-yhteyksien suojaamiseksi. Kuvassa 6 on esitetty kuinka sovelluskerroksella toimivat protokollat toimivat protokollapinossa SSL/TLS-protokollien päällä, salausprotokollien sijaitessa kuljetuskerroksella toimivan TCP-protokollan päällä. SSL- ja TLS-protokollien toimiessa kuljetuskerroksen päällä, niitä voidaan käyttää kaikissa HTTP:n tavoin sovelluskerroksella toimivien ja TCP-yhteyttä käyttävien protokollien suojaamiseksi [31]. Salausprotokollaa käytettäessä normaalisti TCP-pakettien sisällä salaamattomasti siirrettävä tieto on salattuna salausprotokollan pakettien sisällä, joita TCP-protokolla kuljettaa aivan kuten salaamattomakin tietoa.



**Kuva 6.** SSL/TLS-protokollan rakenne protokollapinossa [43]

Kuvassa 6 on esitetty myös SSL/TLS-protokollaan sisältyvät aliprotokollat; kättely- (engl. Handshake Protocol) ja tietueprotokolla (engl. Record Protocol). Kättelyprotokollan tehtävänä on yhteyden muodostamiseen ja ylläpitämiseen liittyvät toiminnot, kuten käytettävän salausalgoritmin ja salausavainten sopiminen, sekä virheiden käsittely. [43]

Tietueprotokollan tehtävänä on sovelluskerrokselta tulevan, siirrettävän tiedon fragmentointi, tiedon eheyden tarkistaminen tiivistefunktioiden avulla sekä itse tiedon salaaminen kättelyprotokollalla sovittuja salausalgoritmeja ja avaimia käyttäen. [43]

Nykyään mitään SSL-protokollan tai TLS-protokollan ensimmäisiä versioita ei suositella käytettäväksi, sillä jokaiseen niistä on olemassa eri haavoittuvuuksia hyödyntäviä hyökkäyksiä. TLS-protokollasta suositellaankin nykyään käytettäväksi vain kahta uusinta;

vuonna 2008 julkaistua 1.2 tai vuonna 2018 julkaistua 1.3 versiota. TLS 1.2 versiosta on myös löydetty haavoittuvuuksia, mutta sen tarjotessa nykyaikaisempia salaus- ja todennusmenetelmiä kuin sen edeltäjät, voi sitä vielä käyttää. [36]

#### 4.4.2 AES

Advanced Encryption Standard (AES) on symmetrinen lohkosalausmenetelmä, josta tuli vuonna 2001 Yhdysvaltain standardoimisviraston (NIST) standardoimana DES-salausalgoritmin seuraaja. AES perustuu NIST:in järjestämän kansainvälisen kilpailun voittaneeseen Rijndael-salaukseen, jonka ovat kehittäneet belgialaiset kryptograafit Vincent Rijmen ja Joan Daemen. Teknisesti AES on Rijndael-salausalgoritmin osajoukko sisältäen vain osan sen ominaisuuksista. [28]

Symmetrisenä salausmenetelmänä AES-salauksessa tiedon salaamiseen ja salauksen purkamiseen käytetään samaa salausavainta, jolloin sekä tiedon lähettäjän että vastaanottajan täytyy tietää käytettävä salainen avain. Lohkosalausmenetelmällä tarkoitetaan sitä, että tiedot salataan tietyn kokoinen lohko kerrallaan. AES-salauksessa lohkot ovat 128-bittisiä [28]. Lohkosalausmenetelmä on jonosalausmenetelmän lisäksi toinen yleisimmistä symmetrisistä salausmenetelmistä.

AES-salauksesta tukee kolmea eri salaustasoa eli avainkokoja; AES-128, EAS-192 ja AES-256, joissa nimen lopussa ilmoitettu luku kertoo käytettävän avaimen koon bitteinä. Salausavaimen pituuden määrittää salauksessa tehtyjen muutoskierrosten määrä, jota käytetään ns. selkotekstin (engl. plaintext) muuttamiseksi lopulliseksi salatuksi tekstiksi (engl. ciphertext). 128-bittinen salaus vaatii 10, 192-bittinen 12 ja 256-bittinen 14 kierrosta. Jokainen kierros muodostuu useammasta prosessointivaiheesta. [28]

AES-salausta pidetään yhä käytännössä murtamattomana, sillä kaikki sen murtamiseksi kehitetyt menetelmät vaativat valtavasti aikaa myös supertietokoneita käytettäessä [41]. Salauksen suurimmat riskit muodostuvatkin salassa pidettävän avaimen hallitsemiseen. AES-128 käyttöä voidaan suositella yleiseen käyttöön, mutta esimerkiksi erittäin salaisen ja kriittisen tiedon salaukseen suositellaan käytettäväksi 192- tai 256-bittistä salausta. [5][41]

AES-salauksen käyttöä voidaan perustella erityisesti sen turvallisuuden lisäksi myös sen nopeudella, sillä nykyaikaisissa prosessoreissa on erityinen AES-salauksen käsittelyyn tarkoitettu käskykanta, joka mahdollistaa paljon nopeamman salauksen ja sen purkamisen laitteistotasolla, lisäten samalla salauksen käytön turvallisuutta.

#### 4.5 Eheys

Työssä tiedonsiirron turvallisuuteen liittyviä teknologioita salaukseen käytettyjen algoritmien lisäksi ovat erityisesti siirretyn tiedon eheyden varmistamiseen käytettyjen

tarkistussummien laskemiseen käytettävät tiivistefunktiot. Tiedonsiirrossa välitettävän tiedon eheys varmistetaan laskemalla siitä ensin tiiviste eli tarkistussumma, joka välitetään siirrettävän tiedon yhteydessä sen vastaanottajalle. Vastaanottaja laskee tiedosta tarkistussumman ja vertaa sitä lähettäjän laskemaan tarkistussummaan. Jos tarkistussummat täsmäävät, on tieto saapunut muuttumattomana vastaanottajalle. Jos tarkistussummat taas eivät täsmää, tiedon sisältö on muuttunut matkalla lähettäjältä vastaanottajalle, jolloin tiedot tulee lähettää uudelleen.

Kuten salausalgoritmeja, tiivistefunktioita on olemassa useita erilaisia, näistä käytetyimpien ollessa MD5, SHA-1 ja SHA-2. MD5 (Message-Digest algorithm) on MD4 tiivistefunktion seuraajaksi vuonna 1992 julkaistu algoritmi, joka tuottaa 128-bittisen eli 32 heksadesimaalimerkkiä pitkän tiivisteeseen [39]. Yleisesti MD5 algoritmin seuraajana pidetty SHA-1 (Secure Hash algorithm 1) on NSA:n (National Security Agency) kehittämä ja NIST:n vuonna 1995 standardoima kryptografinen tiivistefunktio, joka tuottaa 160 bittiä eli 40 merkkiä pitkän tiivisteeseen [27].

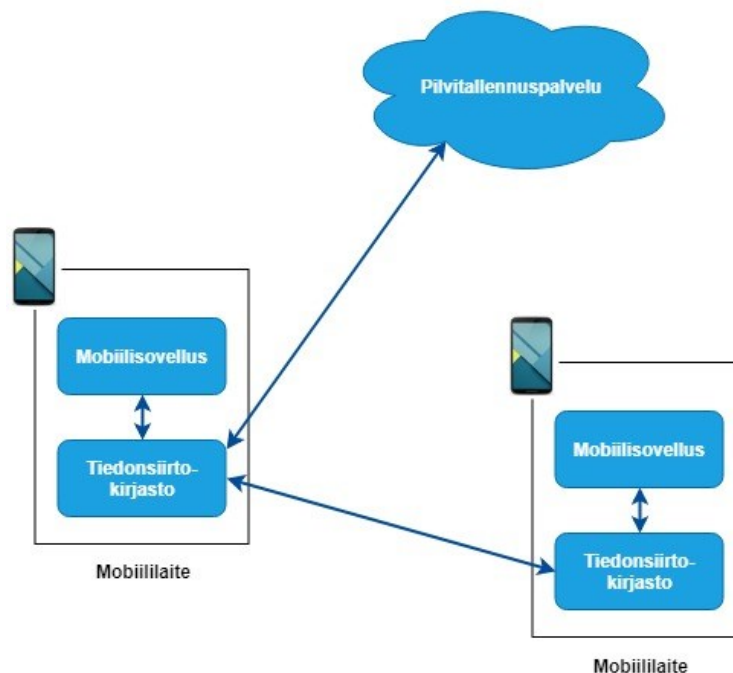
SHA-1 algoritmin syrjäytti NIST:n vuonna 2002 standardoima SHA-2 (Secure Hash algorithm 2). SHA-2 on yhteisnimitys neljällä uudella algoritmilla, jotka ovat SHA-224, SHA-256, SHA-384 ja SHA-512, joissa nimien lopussa oleva luku kertoo tiivistealgoritmin pituuden bitteinä. Näin ollen esimerkiksi SHA-256 tuottaa 256-bittisen tiivisteeseen, joka heksadesimaalimerkkeinä esitettynä on 64 merkkiä pitkä. [27]

Nykyään MD5 ja SHA-1 tiivistealgoritmit ovat todettu kryptografisesti heikoiksi, sillä ne ovat alttiita tiivisteiden yhteentörmäyksille, jolloin kaksi eri syötettä voivat tuottaa saman tiivisteeseen [22]. Tästä syystä niitä ei suositella lainkaan esimerkiksi salasanojen tiivisteiden laskemiseen, vaan niiden sijaan tulisi käyttää vahvempaa tiivistefunktiota kuten jotain SHA-2 algoritmeista.

Kryptografisista heikkouksistaan huolimatta MD5 ja SHA-1 tiivistefunktiot ovat edelleen laajalti käytössä tarkistussummia laskettaessa, sillä haavoittuvuuksien merkitys tiedon eheyden varmistamisessa ei toistaiseksi ole kovin merkittävä [22]. Näiden tiivistealgoritmien käyttöä puoltaa myös tarkistussummien laskemisen nopeus verrattuna SHA-2 algoritmeihin [37].

## 5. TOTEUTETTU TIEDONSIIRTOSOVELLUS

Työlle asetettujen toiminnallisten ja ei-toiminnallisten vaatimuksien pohjalta toteutettu järjestelmä muodostuu mobiililaitteisiin asennettavasta tiedonsiirto-sovelluksesta ja sen käyttämästä Piceasoftin pilvitalennuspalvelusta. Kuvassa 7 on esitetty kokonaisuusien muodostama järjestelmä ja sen eri osien välinen yhteys. Kuvassa on esitetty tarkemmin myös mobiililaitteessa toimivan tiedonsiirto-sovelluksen sisältämät kaksi keskeistä kokonaisuutta; Android-sovellusmoduuli ja sen käyttämä Android-kirjastomodulaali.



*Kuva 7. Tiedonsiirtojärjestelmän osat*

Kuvassa 7 esitetty Android-sovellus on laitteeseen asennettava sovellus, jonka päätehtävänä on tarjota tiedonsiirto-sovelluksen käyttäjälle etualalla toimiva graafinen käyttöliittymä ja toimia rajapintana käyttäjän ja tiedonsiirron toteuttavan kirjaston välillä. Sovellusmoduuli kutsuu tiedonsiirtokirjaston rajapintoja käyttäjän käyttöliittymän kautta määrittelemällä tavalla ja esittää käyttäjällä tiedonsiirron etenemisestä kertovia tietoja siihen tarkoitettussa näkymässä.

Android-kirjasto on Android-sovellukseen sisällytetty moduuli, joka toimii sovelluksen suorituksessa taustalla tarjoten sovellukselle rajapinnat kaiken tiedonsiirrolle oleellisen toiminnallisuuden suorittamiseksi. Kirjastomodulaalin päätehtävänä on siis toteuttaa yhteyden muodostaminen laitteiden välille eri teknologioita käyttäen. Yhteyden muodostamisen jälkeen se vastaa kaikesta sisällönhallinnasta, kuten sisällön käsittelemisestä sisältötyypeittäin sekä sen lähettämisestä ja vastaanottamisesta joko kahden laitteen tai lait-

teen ja pilvitalennuspalvelun välillä. Tiedonsiirron suorittamisen mahdollistavien rajapintojen lisäksi kirjasto tarjoaa kuuntelijarajapinnat tiedonsiirron etenemisen seuraamiseksi.

Koska Piceasoftin pilvipalvelu on kehitetty vain demotarkoitukseen ja sen käyttöä ei ole tarkoitus mahdollistaa asiakkaille, kuvassa 7 esitetyn tiedonsiirtosovelluksen käyttämää pilvitalennuspalvelua ei käsitellä tässä diplomityössä sen käyttämiseksi toteutettua rajapintaa tarkemmin.

## 5.1 Tiedonsiirtokirjasto

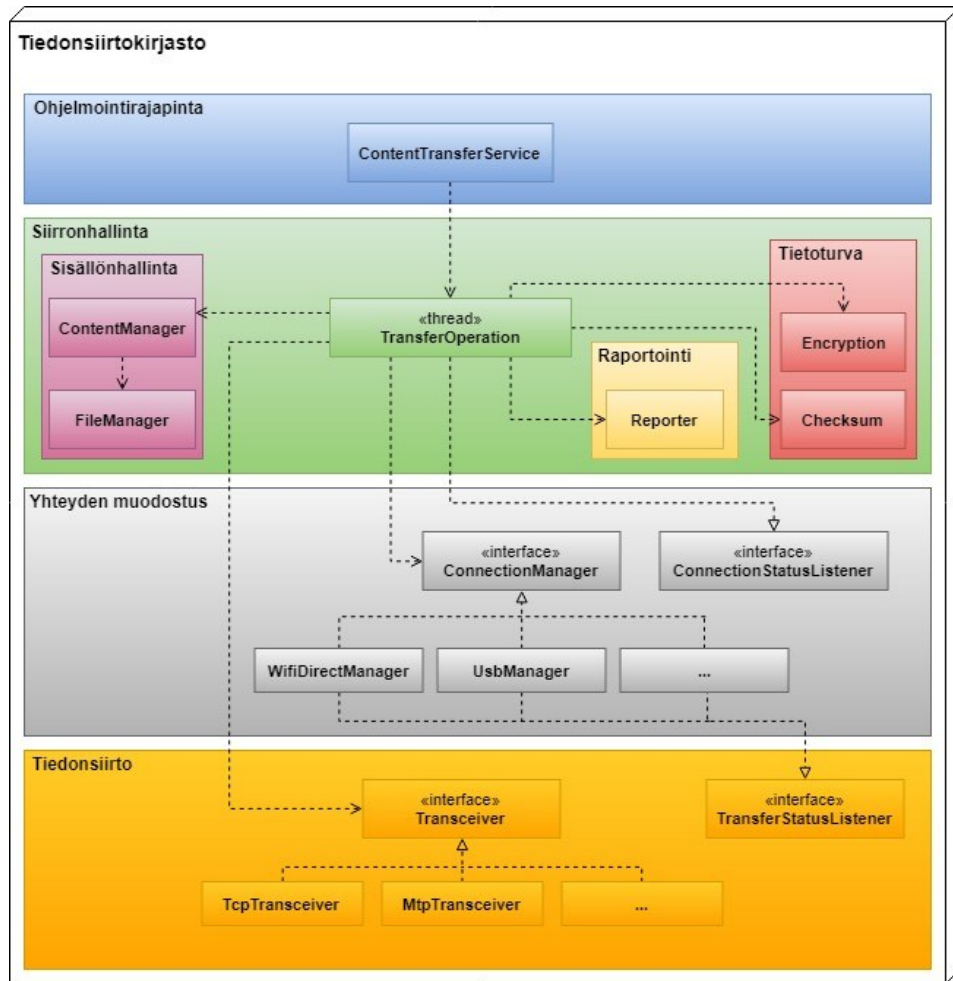
Tiedonsiirtosovelluksen päätoiminnallisuus päätettiin toteuttaa kirjastona muiden Piceasoftin mobiilisovelluksiin tehtyjen toiminnallisuuksien, kuten laitteen diagnosoinnin ja tyhjentämisen, tavoin. Tämän ansiosta tiedonsiirron toteuttavan ohjelmiston siirrettävyys Android-sovelluksesta toiseen on myös mahdollisimman vaivatonta ja samalla se mahdollistaa työssä toteutetun oleellisen toiminnallisuuden jakamisen osana SDK:ta.

Android-kirjasto on rakenteellisesti kuin Android-sovellus eli se sisältää kaiken sovelluksen tekemiseksi tarvittavan lähdekoodista muihin resursseihin kuten käyttöliittymän asettelu- ja kuvatiedostoihin [12]. Toisin kuin sovellukset, jotka käännetään Androidiin asennettaviksi APK (Android Application Package) -tiedostoiksi, kirjasto käännetään AAR (Android Archive) -tiedostoksi, joka voidaan ottaa käyttöön missä tahansa Android-sovellusprojektissa [12].

### 5.1.1 Arkkitehtuuri

Tiedonsiirtokirjaston toteuttamisessa päädyttiin kerrosarkkitehtuuriin, koska se on lähes aina sovellettavissa ja mahdollistaa ohjelmiston rakenteen määrittelyn eri käsitetasoilla, missä järjestelmä koostuu omat käyttötarkoituksensa omaavista kerroksista. Yleisesti kerrosarkkitehtuurissa ylempänä olevat kerrokset abstrahoivat niiden alapuolella olevien kerrosten toiminnallisuuden ja tavoitteena on, että abstraktiotasoltaan korkeammat, ylempänä toimivat kerrokset käyttävät niiden alapuolella olevia abstraktiotasoltaan matalampia palveluita. [23]

Kuvassa 8 on esitetty tiedonsiirtokirjaston korkean tason arkkitehtuuri ja sen keskeisimmät eri käyttötarkoitusten mukaan muodostetut loogiset kokonaisuudet. Arkkitehtuuri ei ole täysin tiukka, vaan kerrokset voivat niiden sisällä toimivien kokonaisuuksien tavoin olla osittain sulautuneita toisiinsa.



*Kuva 8. Kirjaston korkean tason arkkitehtuurin kuvaus.*

Ohjelmiston arkkitehtuurissa on pyritty kerrosarkkitehtuurille tyypilliseen periaatteeseen, jossa lähimpänä ihmistä olevat tasot ovat ylhäällä ja alimpana sijaitsevat tasot ovat tyypillisesti lähempänä laitteistoa sekä käyttöjärjestelmän rajapintoja [23]. Kirjastossa lähimpänä ihmistä on sen ohjelmointirajapinta, jonka kautta ohjelmoija ja sitä kautta asiakassovellus voivat käyttää kirjaston tarjoamia palveluita.

Rajapintakerros käyttää alapuolellaan sijaitsevaa siirronhallintakerrosta, joka vastaa sisällön siirron kontrolloinnista rajapintakutsujen määrittämällä tavalla. Tämä kerros pitää sisällään myös sisällönhallintaan, siirron raportointiin ja tietoturvaan liittyvät kokonaisuudet. Arkkitehtuurissa alimpana ovat yhteyksien muodostamisesta ja fyysisestä tiedonsiirrosta vastuussa olevat kokonaisuudet, niiden toimiessa lähimpänä laitteistoa ja käyttöjärjestelmän tarjoamia rajapintoja.

Arkkitehtuurissa jokaisella kerroksella on määritelty omat rajapinnat, joiden avulla niitä käytetään, jolloin rakenne ei aseta rajoituksia muiden kerrosten toteutuksille. Tämä myös mahdollistaa yksittäisten kerrosten muuttamisen ilman, että muita kerroksia tarvitsee lainkaan muuttaa. Rajapinnan muuttuessa se vaikuttaa pahimmillaan vain kerroksen tarjoamia palveluita käyttäviin lähimpiin kerroksiin. [8]

Kuvassakin esiintyvälle ylhäältä alaspäin tapahtuvalle kerrosten ohittamiselle on usein myös perusteita, eikä sitä pidetä yleisesti pahana poikkeamisena kerrosarkkitehtuurista. Kerrosten ohittamista voidaan perustella esimerkiksi suorituskykyisyyllä, sillä palvelukutsujen välittäminen aina välissä olevan kerroksen kautta voi olla täysin turhaa. Sen sijaan ylemmän kerroksen palveluiden kutsuminen alemmalla kerrokselta on ongelma ja se rikkoo kerrosarkkitehtuurin periaatteita, sillä silloin alemmalle kerrokselle muodostuu riippuvuus ylemmästä kerroksesta. Jotta alempi kerros pysyisi riippumattomana ylemmästä, on kutsujenvälitys ylempään kerrokseen toteutettava takaisinkutsuja (engl. callback) käyttäen. [23]

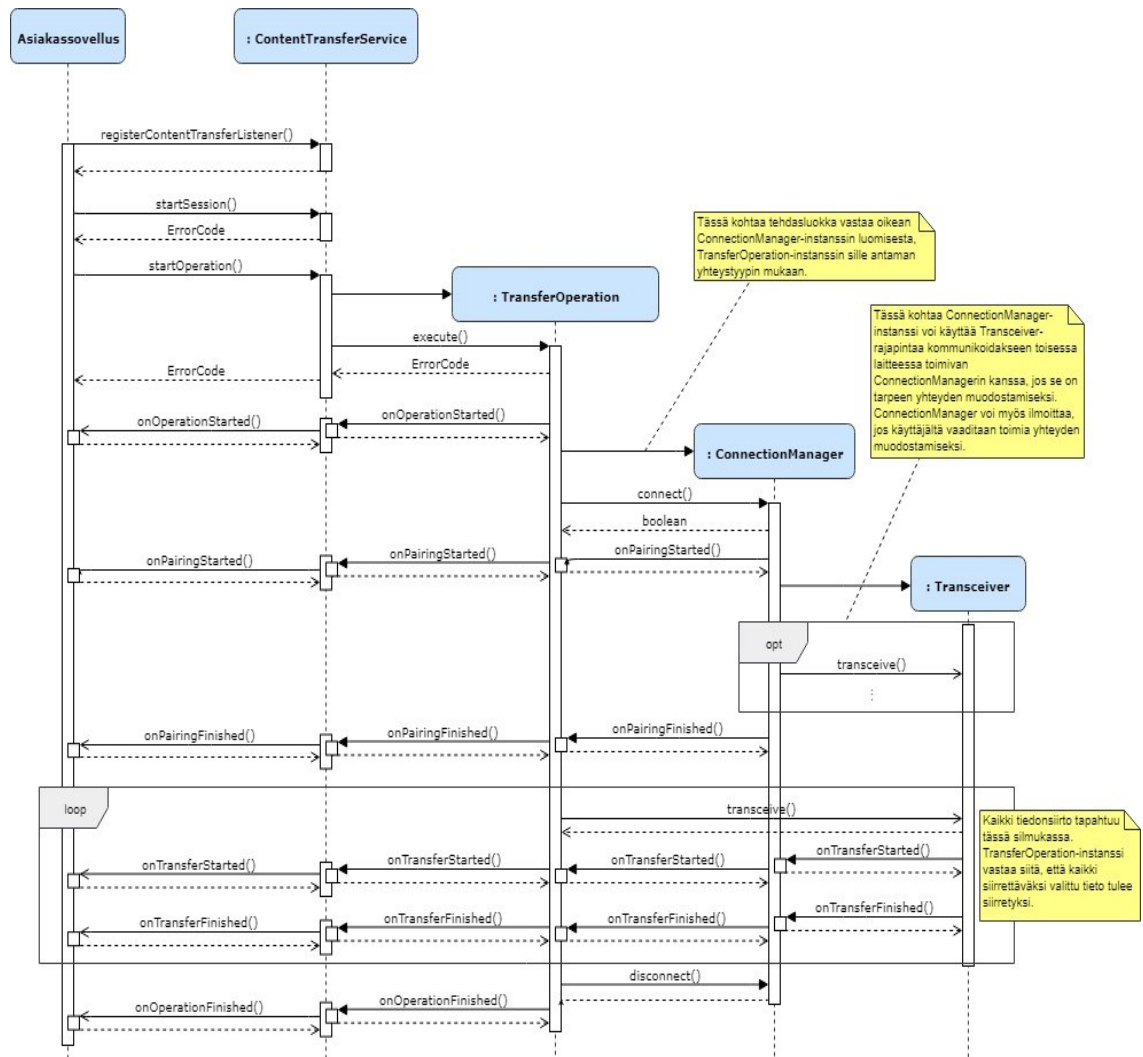
Kuvassa 8 on myös esitetty kunkin arkkitehtuurisen kokonaisuuden keskeisimmät luokat, jotka on kuvattu tarkemmin taulukossa 4.

*Taulukko 4. Tiedonsiirtokirjaston keskeisimmät luokat*

<b>Luokka</b>	<b>Kuvaus</b>
ContentTransferService	Toimii tiedonsiirtokirjaston ohjelmointirajapinta, jonka metodeja kutsumalla asiakassovellus voi käyttää kaikkia kirjaston tarjoamia toimintoja. Kutsuu arkkitehtuurissa alemmalla abstraktiotasolla toimivia rajapintoja. Luokalle rekisteröitävien kuuntelijoiden avulla se ilmoittaa siirto-operaation tilasta asiakassovellukselle.
TransferOperation	Toimii kirjaston tiedonsiirtoa kontrolloivana instanssina, joka suorittaa siirron kirjaston ohjelmointirajapinnan kautta annetun konfiguraation mukaan. Luokan instanssia suoritetaan omassa säikeessään, joka toimii sen aloittamisen jälkeen itsenäisesti. Sen tehtävänä on mm. valitun tiedonsiirtoyhteyden avaaminen yhteyden muodostus-kerroksen tarjoamien rajapintojen avulla, valittujen sisältötyyppien välittäminen salattuna tiedonsiirtokerrokselle siirrettäväksi ja vastaanotetun sisällön salauksen purkaminen ja eheyden varmistaminen.
ContentManager	Toimii sisällönhallintakomponentin rajapintana. Kutsuu eri sisältöä käsitteleviä luokkia siltä pyydettyjen sisältötyyppien mukaan.
FileManager	Toimii tiedostonhallinnan rajapintana. Vastaa sisällönhallinnassa tiedostojärjestelmässä olevien tiedostojen käsittelystä.

Reporter	Toteuttaa tiedonsiirrosta raportoinnin. Saa raportin muodostamiseen tarvittavat tiedot sitä käyttävältä siirto-operaatiolta.
Encryption	Toteuttaa tiedon salauksen ja salauksen purkamisen, käyttäen sille annettuja salausalgoritmeja ja avaimia.
Checksum	Toteuttaa siirrettyjen tiedostojen tarkistussumman laskemisen käyttäen sille annettuja tiivistefunktioita.
ConnectionManager	Toimii rajapintana yhteyden muodostamiselle. Toteutettu rajapintaluokkana, joka määrittelee yhteyden muodostamiseksi vaadittavat rajapintametodit, abstrahoiden näin ollen yhteyden muodostamisen konkreettisen toteutuksen. Yhteyden muodostamisesta vastaa siis rajapinnan toteuttavat konkreettiset luokat, jotka toimivat yhteistyypin määrittelemällä tavalla.
ConnectionStatusListener	Kuuntelijarajapinta, jonka avulla yhteyden muodostus -kerrokselta välitetään yhteyden tilaan liittyviä tietoja arkkitehtuurissa ylempänä toimiville kerroksille.
Transceiver	Toimii rajapintana fyysiselle tiedonsiirtämiselle. Toteutettu rajapintaluokkana, joka määrittelee tiedonsiirtämiseksi vaadittavat rajapintametodit, abstrahoiden näin ollen tiedonsiirron konkreettisen toteutuksen. Tiedonsiirrosta vastaa siis rajapinnan toteuttavat konkreettiset luokat, jotka toimivat valitun yhteistyypin edellyttämällä tavalla.
TransferStatusListener	Kuuntelijarajapinta, jonka avulla tiedonsiirtokerrokselta välitetään tiedonsiirtoon liittyviä tietoja arkkitehtuurissa ylempänä toimiville kerroksille.

Kuvassa 9 on esitetty sekvenssikaavio tiedonsiirtoon osallistuvien instanssien välisestä viestien kulusta ja operaatioiden tapahtumajärjestyksestä. Sekvenssikaavio on hyvin yksinkertaistettu ja siinä on kuvattu vain kuvassa 8 esitetyn arkkitehtuurin kannalta tiedonsiirron keskeisimmät instanssit ja niiden väliset operaatiot, tietoja lähettävässä laitteessa. Siitä on jätetty myös kuvan selkeyden vuoksi metodikutsujen parametrit esittämättä, sillä ne selitetään myöhemmin. Sekvenssikaaviossa vasemmalta oikealle etenevät kutsut vastaavat arkkitehtuurissa matalammille abstraktiotasojen meneviä kutsuja. Näin ollen kerrosarkkitehtuurin periaatteita noudattaessa vasemmalle päin menevät ovat aina takaisin-kutsuja.



**Kuva 9.** Tiedonsiirron keskeisimpien instanssien välinen kommunikaatio lähdelaitteessa

Käyttäjän tehtyä luvussa 5.2.1 esitetyt tiedonsiirtoa edeltävät valinnat asiakassovelluksen käyttöliittymässä, asiakassovellus muodostaa valinnoista konfiguraatio-objektit ja aloittaa tiedonsiirron kutsumalla ContentTransferService:n `startSession()`- ja `startOperation()`-metodeja välittäen niille konfiguraatio-objektit parametreina. Saadakseen tietoja käynnissä olevasta siirto-operaatiosta asiakassovelluksen tulee myös rekisteröidä ContentTransferListener-kuuntelijarajapinnan toteuttava instanssi `registerContentTransferListener()`-metodin avulla. Näistä tiedonsiirtokirjaston tarjoamista rajapintameteodeista ja niiden parametreista kerrotaan tarkemmin luvussa 5.1.2.

Aloittaessaan uuden tiedonsiirto-operaation, ContentTransferService luo uuden TransferOperation-instanssin asiakassovellukselta saamallaan konfiguraatiolla. Tämän jälkeen kutsutaan siirto-operaation `execute()`-metodia, joka käynnistää sen omassa säikeessään. Operaatio ilmoittaa operaation onnistuneesta alkamisesta sen käynnistäneelle ContentTransferService:lle, joka lähettää saman ilmoituksen rekisteröityneille kuuntelijoille kirjaston ulkopuolella. Tämän jälkeen operaatio rakennuttaa ConnectionManager-rajapinnan toteuttavan instanssin.

Kuten kuvassa 8 on esitetty, ConnectionManager-rajapinnan toteuttavat konkreettiset luokat ovat sidottu aina johonkin tiettyyn tiedonsiirtotekniikkaan. Tällöin ne toteuttavat rajapinnassa määriteltyjen metodien toiminnan tietyn yhteystyyppin ja yhteyden käytön mahdollistavien Android-järjestelmän rajapintojen edellyttämällä tavalla.

Konkreettisten luokkien käyttöä varten on toteutettu tehdasluokka (engl. Factory), joka luo instanssin sille annetun yhteystyyppin mukaisesta luokasta. Tehdas palauttaa luodun instanssin aina ConnectionManager-tyyppisenä. Tällöin rajapintaa käyttävän, tiedonsiirtoa kontrolloivan operaatio-instanssin, tarvitsee osata kutsua vain rajapinnassa määriteltyjä metodeja, välittämättä sen konkreettisesti toteutuksesta.

ConnectionManager-rajapintaluokka on määritelty Javassa seuraavasti:

```
public interface ConnectionManager {

    @NonNull
    ConnectionDetails getConnectionDetails();

    boolean connect(@NonNull ConnectionDetails connectionDetails,
                   @NonNull DeviceRole deviceRole,
                   @NonNull ConnectionStatusListener listener);

    void disconnect();
}
```

### ***Ohjelma 1. ConnectionManager-rajapintaluokan määrittely Javassa***

ConnectionManager-rajapinnan ja sen konkreettisesti toteuttavat instanssit luovan tehdasluokan käyttö osaltaan mahdollistavat kirjaston toteutuksen helpon laajennettavuuden. Tällöin uusien yhteystyyppien lisääminen yhteyden muodostus -kerrokselle voidaan tehdä käytännössä täysin siirronhallinta-kerroksella näkymättä. Joissain tapauksissa siellä olevan toteutuksen muokkaaminen voi kuitenkin olla tarpeeseen, jos se ei uuden yhteystyyppin käyttöön sellaisenaan sovellu.

ConnectionManager-instanssin *connect()*-metodia kutsuessaan tulee operaation antaa sille parametreiksi ConnectionDetails- ja DeviceRole-objektit, jotka määrittelevät yhteyden muodostamiseksi oleelliset tiedot ja laitteen roolin tiedonsiirrosta. Nämä tiedot operaatio on saanut omasta konfiguraatiostaan. Kuten kuvassa 8 on esitetty, TransferOperation toteuttaa ConnectionStatusListener-rajapinnan, jolloin se antaa itsensä *connect()*-metodin viimeiseksi parametriksi.

Tieto yhteyden tilasta välitetään ylemmille kerroksille takaisinkutsu-menetelmällä toimivan ConnectionStatusListener-kuuntelijarajapinnan avulla, joka on määritelty Javassa seuraavasti:

```

public interface ConnectionStatusListener {

    void onPairingStarted();

    void onPairingFinished(boolean success,
                            @Nullable Transceiver transceiver);

    void onTransferStarted();

    void onTransferProgress(int progress);

    void onTransferFinished();

    void onDataReceived(@Nullable Transferable transferable,
                        @Nullable InputStream data);

    void onPermissionsRequired(@NonNull String[] permissions);

    void onUserActionRequired(int action, @NonNull String message);
}

```

### ***Ohjelma 2. ConnectionStatusListener-rajapintaluokan määrittely Javassa***

Yhteyden muodostuttua siirto-operaatio saa siitä tiedon *onPairingFinished()*-metodin takaisinkutsulla, jossa se saa parametrina myös Transceiver-instanssin, jota se käyttää tietojen lähettämiseen.

Arkkitehtuurin tiedonsiirtokerroksen rajapinnan muodostaa Transceiver-rajapintaluokka. Transceiver-instanssin luomisesta vastaa aina jonkin yhteystyyppin ConnectionManager-instanssi, sillä tiedonsiirron toteutus on usein hyvin sidottu tiettyyn siirtotekniikkaan. Transceiver-rajapinta on määritelty Javassa yksinkertaisesti:

```

public interface Transceiver {

    void transceive(@NonNull Transferable transferable);

    void transceive(byte[] data);

    void close();
}

```

### ***Ohjelma 3. Transceiver-rajapintaluokan määrittely Javassa***

Transceiver-rajapinta mahdollistaa Transferable-tyyppisten objektien sekä raakadatan siirtämisen. Transferable-tyyppiset objektit voivat olla raakadatan identifioivaa meta-dataa tai muuhun kommunikointiin käytettäviä viestejä.

Vastaavasti kuin yhteyden muodostus -kerrokselta välitetään tietoa ylemmille kerroksille takaisinkutsu-menetelmällä, myös tiedonsiirtokerroksen tapahtumien kuuntelua varten on toteutettu oma TransferStatusListener-kuuntelijarajapinta, joka on määritelty Javassa seuraavasti:

```
public interface TransferStatusListener {

    void onTransferStarted();

    void onTransferProgress(int progress);

    void onTransferFinished();

    void onDataReceived(@Nullable Transferable transferable,
                        @Nullable InputStream data);
}
```

#### *Ohjelma 4. TransferStatusListener-rajapintaluokan määrittely Javassa*

Kuvassa 8 on esitetty kuinka kukin ConnectionManager-instanssi toteuttaa kyseisen rajapinnan, jolloin ne välittävät itsensä uudelle Transceiver-instanssille kuuntelijaksi sellaisen luodessaan.

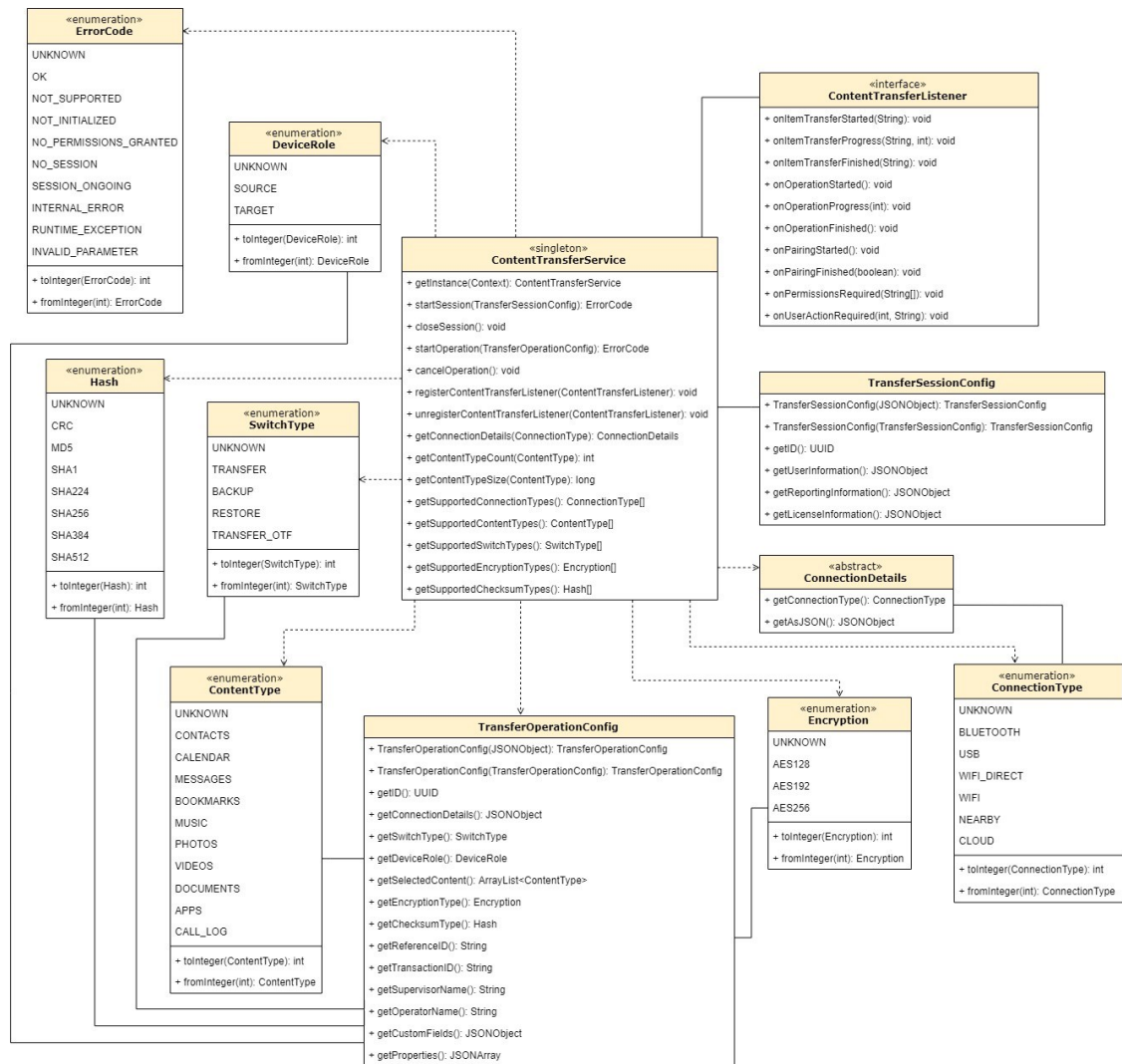
### 5.1.2 Ohjelmointirajapinta

Tiedonsiirtokirjaston julkista ohjelmointirajapintaa suunnitellessa oli otettava huomioon työlle asetetut toiminnalliset ja ei-toiminnalliset vaatimukset. Kirjaston toteuttaessa kaiken tiedonsiirron, sitä käyttävän asiakassovelluksen pitää pystyä tekemään kaikki toiminnallisissa vaatimuksissa mainitut toiminnot sen rajapinnan kautta.

Toiminnallisten vaatimuksien lisäksi kirjaston rajapinnan toteutuksessa pyrittiin noudattamaan hyvälle rajapintasuunnittelulle yleisiä käytäntöjä, jolloin se on yksinkertainen, helppo käyttää ja hyvin dokumentoitu. Lisäksi rajapinnan toteutuksessa pyrittiin yhtenäisyyteen muiden Piceasoftin mobiilikirjastojen rajapintojen kanssa sekä siirrettävyyteen mobiilialustojen välillä.

Kuten kuvassa 8 on esitetty, tiedonsiirtokirjaston arkkitehtuurissa ylimmällä ohjelmointirajapintakerroksella toimii ContentTransferService-luokka. Tämän luokan tehtävänä on toimia kirjaston rajapintana, jolloin kaikki kirjaston tarjoamien toiminnallisuuksien käyttö tapahtuu kutsumalla vain sen julkisia metodeja.

ContentTransferService-luokan lisäksi kirjasto sisältää joitain julkiseksi määriteltyjä tietotyyppejä, joita käytetään esimerkiksi ohjelmointirajapinnan tarjoamien metodien paluuarvoissa ja parametreina. Pääasiassa kaikki arkkitehtuurissa ohjelmointirajapinnan alapuolella toimivat kerrokset ja niiden luokat ovat kirjaston sisäistä toteutusta eikä näin ollen näy kirjaston ulkopuolelle. Kuvassa 10 on esitetty luokkakaavio tiedonsiirtokirjaston julkisen rajapinnan muodostavista luokista ja niiden julkisista metodeista.



**Kuva 10.** Tiedonsiirtokirjaston julkisten luokkien luokkakaavio

Julkisella rajapinnalla tarkoitetaan sitä, että vain kuvassa esitetyt luokat ja metodit ovat sellaisenaan sovellusohjelmoijan nähtävissä ja näin ollen tiedonsiirtokirjastoa käyttävän asiakassovelluksen käytettävissä.

## Tiedonsiirron suorittaminen

ContentTransferService pyrittiin pitämään mahdollisimman yksinkertaisena ja helppokäyttöisenä, jolloin sen rajapintametodien parametrien lukumäärä on pyritty minimoimaan. Tämä on mahdollistettu toteuttamalla TransferSessionConfig- ja TransferOperationConfig-luokat, jotka kokoavat tiedonsiirtosessioon ja -operaatioon liittyvät tiedot yhdeksi kokonaisuudeksi.

Sessio on operaatioit yhteen sitova ylemmän tason kokonaisuus, joka muodostuu Piceasofin tuotteen käyttöön liittyvistä asiakas-, lisenssi- ja raportointitiedoista, joita ei

tässä käsitellä sen tarkemmin. Operaation tiedot liittyvät jonkin Piceasoftin tuotteen tarjoaman toiminnon, kuten laitteen diagnosoinnin, suorittamiseen. Tässä työssä operaatiolla tarkoitetaan siis tiedonsiirtoa laitteesta toiseen.

Konfiguraatio-objektien määrittely tapahtuu JSON-objektien avulla. JSON on tiedonvälitykseen kehitetty avain-arvo -pareista muodostuva tekstiformaatti, jonka käyttö mahdollistaa työssä konfiguraatietietojen välittämisen esimerkiksi QR-koodien avulla. Toteutuksessa sovelluksessa ja Piceasoftin OnTheFly-tuotteissa QR-koodien käyttö perustuu siis niiden sisältämään JSON-informaatioon. Sessio- ja operaationkonfiguraatioiden määrittelemiseksi käytettävä JSON voi näyttää seuraavalta:

```
{
  "server_type": "",
  "session_id": "",
  "user": { ... },
  "license": { ... },
  "reporting": { ... },
  "reference_id": "",
  "transaction_id": "",
  "operator_name": "",
  "supervisor_name": "",
  "custom_fields": {},
  "properties": [],
  "switch_type": 1,
  "connection_details": {
    "connection_type": 1,
    "device": "",
    "url": ""
  }
  "content": [0, 1, 2],
  "role": 1,
  "encryption": 1,
  "checksum": 1
}
```

#### **Ohjelma 5.** Konfiguraatio-objektien määrittelemiseksi käytetyn JSON:in rakenne

Kun ohjelmassa 5 esitettyä JSON:ia kuvaava JSONObject annetaan TransferSessionConfig- ja TransferOperationConfig-luokkien rakentajille parametrina, muodostavat ne JSON:in sisältämästä tekstimuotoisesta informaatiosta ohjelmointikielen mukaiset vastineet. Esimerkiksi *role*-avainta vastaava numeroarvo on sovelluksessa määritelty DeviceRole-tyyppisenä *SOURCE*. Tämä on myös paluuarvo, jonka JSON:in avulla määritellyn TransferOperationConfig-luokan instanssi palauttaa sen *getDeviceRole()*-metodia kutsuttaessa.

Samaa JSON:ia voi siis työssä käyttää molempien konfiguraatioiden luontiin. JSON-formaatin käyttö työssä mahdollistaa myös helpon konfiguroitavuuden, sillä siihen on helppo lisätä uusia avain-arvo -pareja ja sitä käyttävä ohjelmisto voi käsitellä vain tarvitsemiaan arvoja. Kirjaston rajapinnan käsitellessä vain JSON:ia, kirjaston asiakassovellus

voi käyttää sen siirtämiseen käytännössä mitä vain tekstimuotoista dataa siirtävää teknologiaa QR-koodien sijaan.

Kuten kuvan 9 sekvenssikaaviossa on kuvattu, tiedonsiirto aloitetaan kutsumalla rajapinnan *startSession()*- ja *startOperation()*-metodeja antamalla niille niitä vastaavat konfiguraatio-objektit parametreina. Session aloittaminen on edellytys Piceasoftin tuotteen käytölle, jolloin se tulee tehdä aina ennen operaation aloittamista. Tämä tekee rajapinnasta myös yhdenmukaisen Piceasoftin muiden mobiilikirjastojen kanssa, sillä niiden käyttäminen tapahtuu vastaavasti samannimisiä metodeja kutsumalla.

Tiedonsiirron suorittamisen mahdollistavan rajapinnan lisäksi kirjasto tarjoaa *ContentTransferListener*-kuuntelijarajapinnan tiedonsiirron etenemisen seuraamiseksi. Kuuntelijoiden rekisteröinti on esitetty kuvan 9 sekvenssikaaviossa *registerContentTransferListener()*-metodia kutsumalla. Metodi ottaa parametrinaan *ContentTransferListener*-rajapinnan toteuttavan instanssin ja sitä voidaan käyttää useiden instanssien rekisteröintiin. *ContentTransferService* välittää sekvenssikaavion mukaisesti tiedon kirjaston sisäisistä tapahtumista kaikille rekisteröityneille kuuntelijoille. Tällöin kuuntelijaksi rekisteröitynyt asiakassovellus voi esittää käyttäjälle tapahtumatietoja käyttöliittymänsä kautta.

## Dokumentaatio

*ContentTransferService*in julkinen rajapinta on dokumentoitu Javadoc-dokumentaation generointi -työkalua käyttäen. Javadoc on Java-ohjelmointikieltä varten kehitetty, joten se on toimiva valinta Java-koodin ja siinä olevien rajapintojen dokumentointiin. Javadoc-dokumentaatio on nähtävissä reaaliajassa Java-ohjelmointiympäristöissä (engl. IDE, Integrated Development Environment), joka helpottaa kehittämistä, kun ohjelmoija näkee luokkien ja metodien dokumentaation niitä käyttäessä. Monet ohjelmointiympäristöt myös mahdollistavat HTML-sivun generoinnin Javadocista, joka helpottaa dokumentaation jakamista.

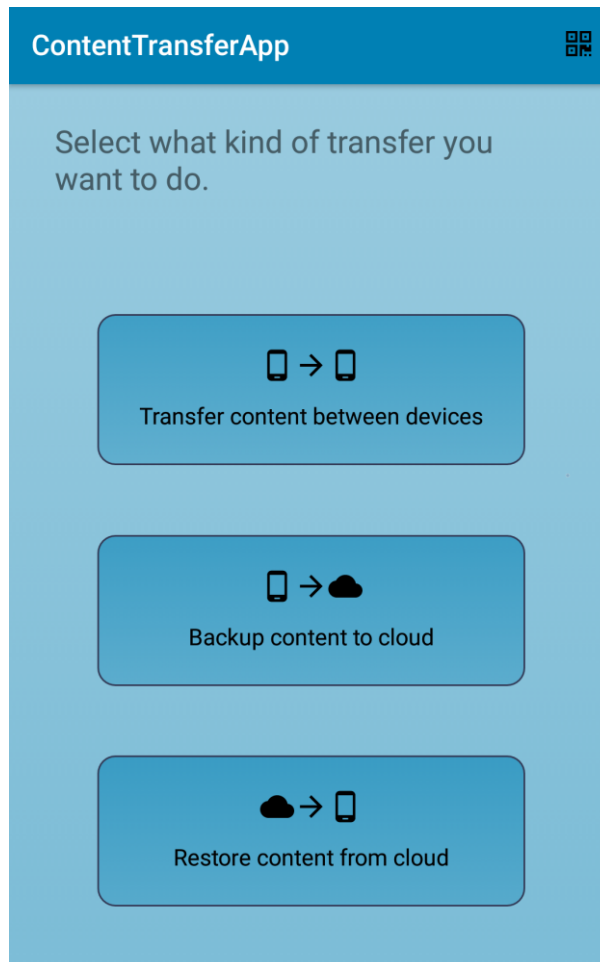
## 5.2 Mobiilisovellus

Android-sovellus toteutettiin perinteisenä sovellusprojektina, johon kirjastomoduli otettiin mukaan aliprojektina kaikkine resursseineen. Tämän ansiosta kirjastoa voitiin kehittää samassa projektissa yhdessä sovelluksen kanssa, eikä sitä tarvinnut kehittää erikseen ja kääntää AAR-tiedostoksi aina ennen kuin uusia muutoksia haluttiin käyttää sovelluksen puolella.

## 5.2.1 Käyttöliittymä

Mobiilisovelluksen käyttöliittymää toteuttaessa noudatettiin käyttöliittymäsuunnittelulle yleisiä periaatteita, jolloin käyttöliittymä pyrittiin pitämään mahdollisimman yksinkertaisena ja selkeänä, tarjoamalla käyttäjille vain tarpeelliset elementit työlle asetetuissa toiminnallisissa vaatimuksissa mainittujen toimintojen käyttämiseksi. Nämä periaatteet osaltaan edesauttoivat työlle asetetun käytettävyyksivaatimuksen täyttämistä mahdollistamalla käyttöliittymän helppokäyttöisyys. Sovelluksen toteutuksessa pyrittiin myös noudattamaan yleisiä Android-käyttöliittymien tyyliohjeita.

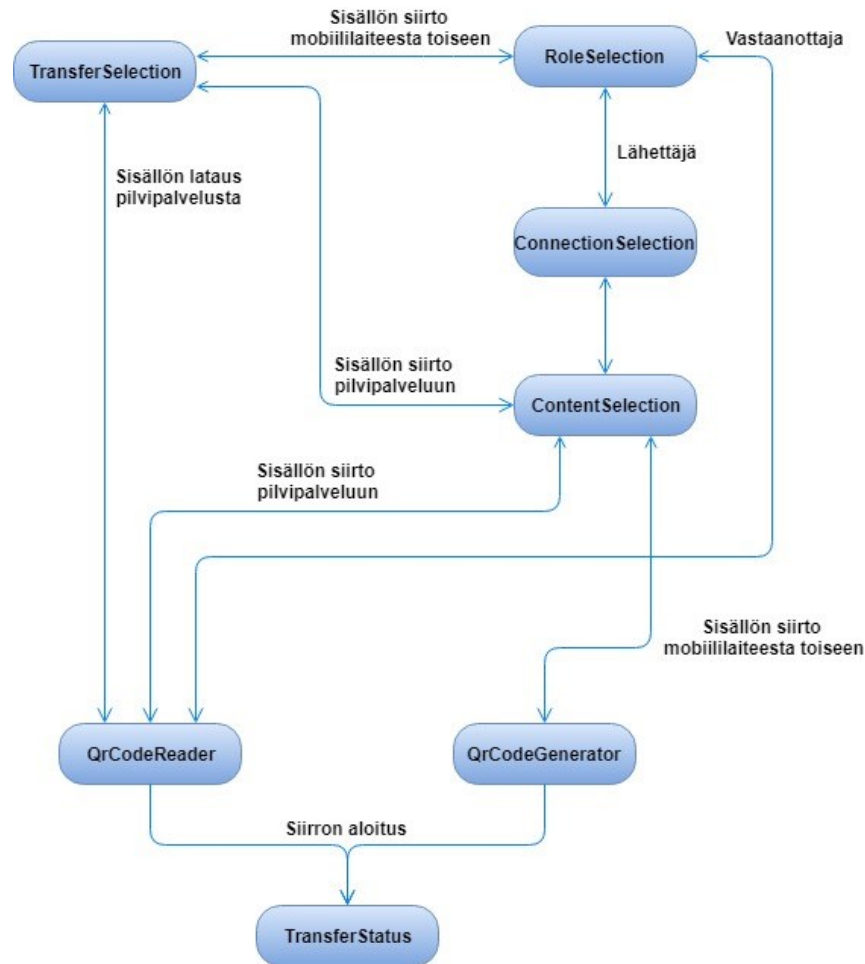
Kun sovellus käynnistetään aukeaa kuvassa 11 esitetty aloitusnäky (TransferSelection). Tässä näkymässä käyttäjä voi painikkeiden avulla valita minkä tahansa luvussa 3.1 esitellyistä tiedonsiirto-sovelluksen päätoiminnallisuuksista; suoraan laitteesta toiseen tapahtuvan tiedonsiirron, tiedonsiirron pilvitalennuspalveluun tai jo pilvipalveluun siirretyn tiedon lataamisen.



*Kuva 11. Mobiilisovelluksen aloitusnäky*

Kuvassa 12 on esitetty käyttäjän navigointi sovelluksen eri näkymien välillä. Ennen kuin tiedonsiirto on aloitettu, kaikista sovelluksen näkymistä on mahdollista navigoida myös

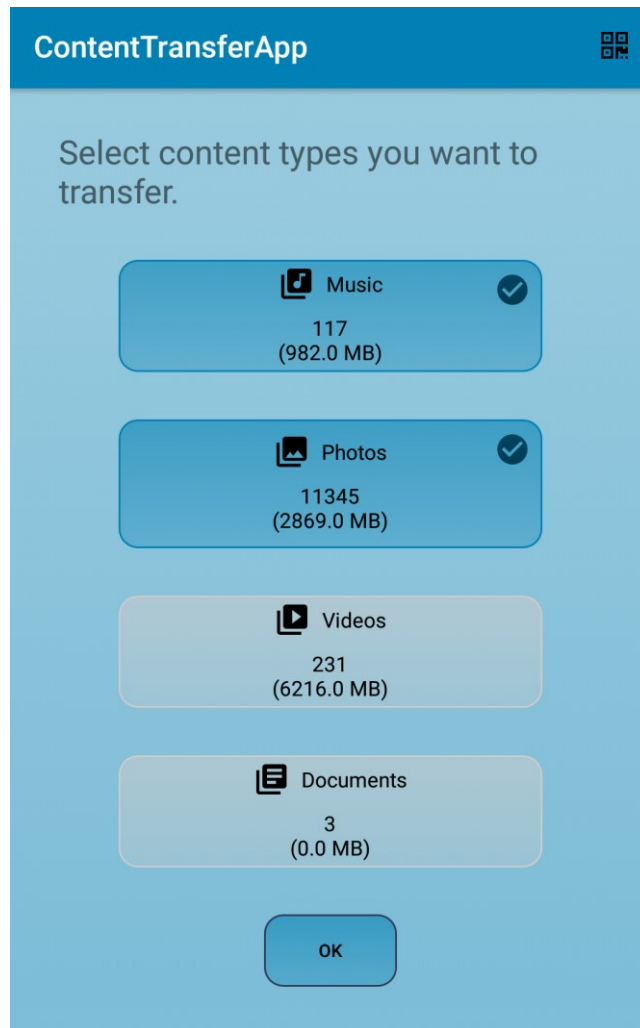
takaisin edelliseen näkymään esimerkiksi jo tehtyjen valintojen vaihtamiseksi. QR-koodin lukija -näkömään (QrCodeReader) siirtyminen on myös mahdollista mistä tahansa sitä edeltävästä näkömäästä, kuvassa 11 näkyvää toimintapalkissa sijaitsevaa QR-koodi -ikonin painamalla. Tällöin käyttäjän ei tarvitse erikseen tehdä valintoja siirron konfiguroimiseksi, kun kaikki konfiguraatio on luettavissa QR-koodista.



**Kuva 12.** Navigointi mobiilisovelluksen näkymien välillä

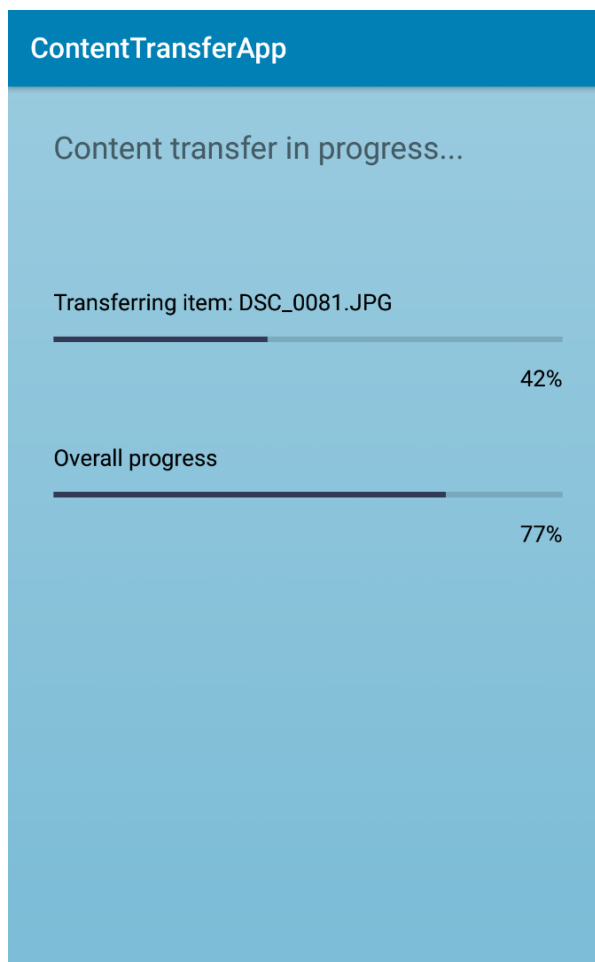
Käyttäjän valitessa siirron osapuolina toimivista laitteista suoraan mobiililaitteesta toiseen tehtävän siirron, avautuu laitteisiin uusi valintanäkymä (RoleSelection), jossa käyttäjä voi valita laitteen roolin tiedonsiirrossa. Vastaanottavaan laitteeseen aukeaa QR-koodin lukija -näkömää, jolla on tarkoitus lukea lähettävään laitteeseen lopulta aukeavan näkömään (QrCodeGenerator) esittämä QR-koodi. Lähettäjäksi valitussa laitteessa avautuu seuraavaksi uusi valintanäkymä (ConnectionSelection), jossa käyttäjä voi valita jonkin sovelluksen tukemista tiedonsiirrossa käytettävistä tiedonsiirtotekniikoista. Tiedonsiirtotekniikan valittuaan käyttäjälle avautuu kuvassa 13 esitetty siirrettävän sisällön -valintanäkymä (ContentSelection), jossa sovelluksen tukemat sisältötyypit ovat valittavissa painikkeiden avulla. Käyttäjän valittua siirrettävän sisällön, aukeaa lähdelaitteeseen QR-koodin esittävä näkömää, joka käyttäjän tulee lukea kohdelaitteella. Kun laitteiden välinen

tiedonsiirto alkaa, molemmissa laitteissa aukeaa automaattisesti kuvassa 14 esitetty näkymä (TransferStatus), josta käyttäjä voi seurata tiedonsiirron etenemistä.



**Kuva 13.** Siirrettävän sisällön -valintanäkymä

Kuvassa 13 on esitetty toteutetun tiedonsiirtokirjaston ja samalla sovelluksen toistaiseksi tukemat sisältötyypit. Tuen lisääminen uusille sisältötyypeille, kuten viesteille, on lopuksi melko triviaalia, sillä käytännössä se vaatii toteutuksen vain niiden lukemiselle ja kirjoittamiselle, tiedonsiirtokirjaston rajapintojen pysyessä sellaisenaan. Sovelluksen tehtävänä on luoda käyttöliittymä tiedonsiirtokirjaston ohjelmointirajapinnan tarjoaman *getSupportedContentTypes()*-metodin palauttaman taulukon mukaisesti. Kuvassakin esitetyt tiedot kunkin sisältötyypin lukumääristä ja niiden yhteiskoosta sovellus saa kirjaston ohjelmointirajapinnan ja *getContentItemCount()*- ja *getContentItemSize()*-metodien avulla. Lopulta se on sovelluksen vastuulla lisätä valitut sisältötyypit *content*-avaimella tiedonsiirron konfigurointiin käytettävään JSON:iin.



*Kuva 14. Tiedonsiirron tila -näkyvä*

Jos käyttäjä valitsee aloitusnäkyvässä tiedonsiirron pilvitallennuspalveluun aukeaa sen jälkeen siirrettävän sisällön -valintanäkymä. Siirrettävän sisällön valittuaan käyttäjälle aukeaa jälleen QR-koodin luku -näkyvä, jonka avulla käyttäjä voi lukea sille esimerkiksi web-sivulla esitetyn QR-koodin, joka määrittelee tarkemmin, kuinka pilvipalveluun siirto tapahtuu. Käyttäjän valitessa jo pilvipalveluun siirrettyjen tietojen lataamisen käyttäjälle aukeaa sen jälkeen suoraan QR-koodin lukija -näkyvä, koska sisältö on jo pilvipalveluun siirrettäessä päätetty. Vastaavasti tällöin käyttäjä voi lukea sille esitetyn QR-koodin, joka määrittelee, kuinka pilvipalveluun siirretyn tiedon lataaminen tapahtuu. kummassakin tapauksessa aukeaa jälleen kuvassa 14 esitetty tiedonsiirron tila -näkyvä tiedonsiirron alkaessa.

Käyttöliittymä on toteutettu Android-alustalle tyypillisellä tavalla käyttämällä käyttöliittymän arkkitehtuurin määritteleviä asettelutiedostoja (engl. layout files). Asettelutiedostot ovat Android-sovelluksen resurssitiedostoihin lukeutuvia XML-tiedostoja, joissa käyttöliittymässä näkyvien elementtien tyylit ja sijainnit on määritelty XML-merkintäkieltä käyttäen. Android mahdollistaa käyttöliittymän määrittelyn myös ohjelmallisesti koodin suorituksen yhteydessä, mutta tällöin käyttöliittymän ylläpito ja tyylien muutta-

minen on vaikeampaa ulkoasun määritysten ollessa sen logiikan määrittelevän ohjelma-koodin seassa. Käyttöliittymämäärittelyjen sijoittamisella asettelutiedostoihin pyritäänkin helpottamaan käyttöliittymän ylläpitoa ja muokattavuutta.

## 5.2.2 Toteutus

Toteutetun mobiilisovelluksen keskeisin logiikka muodostuu yhdestä aktiviteetista (engl. activity) ja sen sisältämistä fragmenteista (engl. fragment). Aktiviteetti on Android-järjestelmässä sovelluksen keskeisimpiä komponentteja. Se tarjoaa käyttöjärjestelmässä ikkunan, johon sovelluksen käyttöliittymä piirretään ja vastaa näin pääasiassa käyttäjän kanssa tapahtuvasta interaktiosta. Ensimmäinen sovelluksen käynnistyessä aukeava aktiviteetti tunnetaan tyypillisesti pääaktiviteettina (engl. main activity) [16]. Toteutetun mobiilisovelluksen käynnistyessä avautuu oletusarvoisesti nimetty MainActivity-aktiviteetti. MainActivity on sovelluksen ainoa aktiviteetti ja vastaa näin ollen sovelluksen näkymistä koko sen elinkaaren ajan.

Siirtyminen sovelluksen näkymien välillä on toteutettu aktiviteetissa fragmenttien avulla. Fragmentti on aktiviteetin kaltainen käyttöliittymän tarjoava komponentti Android-järjestelmässä, joka mahdollistaa modulaaristen käyttöliittymän osien käytön aktiviteeteissa. Toisin kuin aktiviteetit, fragmentit eivät voi siis toimia täysin itsenäisesti, vaan niiden täytyy kuulua aina johonkin aktiviteettiin [15]. Fragmenttien käyttöä näkymien vaihtamiseksi puoltaa mahdollisuus vaihtaa niitä käynnissä olevan aktiviteetin sisällä, joka on suorituskyvyn kannalta kevyempää kuin uusien aktiviteettien käynnistäminen. Niin kauan kuin fragmentti on lisättyä käynnissä olevaan aktiviteettiin se voi vastaanottaa syötteitä käyttäjältä oman käyttöliittymänsä kautta [15].

Kaikki luvussa 5.2.1 esitetyt näkymät toimivat omissa näkymäfragmenteissaan, joista jokaisen käyttöliittymän ulkoasu on määritelty erikseen kunkin omassa XML-asettelutiedostossa. MainActivity vaihtaa eri näkymiä näkyviin niissä tehtyjen valintojen mukaan kuvassa 12 esitetyn navigointikaavion mukaisesti. Androidissa aktiviteetin ja sen fragmenttien välinen kommunikaatio voidaan toteuttaa monella tavalla. Sovelluksessa päädyttiin suoraviivaiseen kuuntelijarajapintojen käyttöön aktiviteetin ja sen alaisuudessa toimivien näkymäfragmenttien välillä. Aktiviteetin toteuttaessa fragmenttien tarjoamat kuuntelijarajapinnat, näkyviin lisätyt fragmentit voivat välittää tiedot käyttäjän interaktiosta aktiviteetille näiden rajapintojen avulla.

Edellä mainittujen pääasiassa käyttöliittymään liittyvien komponenttien lisäksi sovellukseen sisältyy QR-koodien lukemisen ja generoinnin toteuttavat komponentit, joita QrCodeReader ja QrCodeGenerator näkymäfragmentit käyttävät. Android ei itsessään kuitenkaan sisällä ohjelmointirajapintoja näiden toimintojen tekemiseksi, jolloin sovellusprojektin tehtävänä on sisällyttää projektiin kolmannen osapuolen kirjastoja näitä komponentteja varten. Android-alustalle on kuitenkin olemassa useita vapaan ohjelmiston

Apache 2.0 lisenssin kirjastoja kumpaankin tarkoitukseen, jolloin niitä on ollut helppo hyödyntää.

Sovelluksen käyttöliittymän hallinnan lisäksi MainActivity:n pääasiallisena tehtävänä on myös tiedonsiirtokirjaston tarjoaman ohjelmointirajapinnan kutsuminen luvussa 5.1.2 kuvatulla tavalla. Se vastaa siis tiedonsiirron aloittamisesta, kun käyttäjä on edennyt fragmenttien avulla vaiheeseen, jossa sovelluksella on kaikki tiedot siirron konfiguroimiseksi ja TransferSessionConfig- ja TransferOperationConfig-objektien määrittelemiseksi. Aktiiviteetti toteuttaa myös ContentTransferService-luokan tarjoaman ContentTransferListener-kuuntelijarajapinnan, jolloin se voi ohjata rajapinnan kautta saamansa tiedot siirron tilasta kertovalle TransferStatus-näkymän tarjoavalle fragmentille.

### 5.3 Pilvitalennuspalvelu

Mobiilisovelluksen käyttämä pilvitalennuspalvelu on Piceasoftilla kehitetty prototyyppipalvelu, jonka avulla voidaan siirtää tietoja uuteen mobiililaitteeseen pilven kautta. Palvelu on kehitetty ensisijaisesti demotarkoitukseen Piceasoftin asiakkaita varten, eikä sitä ole tarkoitettu loppukäyttäjien käytettäväksi.

Pilvipalvelun käyttämiseksi mobiilisovelluksen käyttäjän tulee lukea siirron konfiguroiva QR-koodi. QR-koodin sisältämä JSON määrittelee, tehdäänkö tiedonsiirto mobiililaitteesta pilvipalveluun vai pilvipalvelusta mobiililaitteeseen. Tämän lisäksi se määrittelee käytettävän pilvipalvelun osoitteen.

Jos Piceasoftin asiakastaho toteuttaisi vastaavan rajapinnan toteuttavan palvelun käyttäen omaa palvelin-infrastruktuuriaan, voisi työssä kehitetty mobiilisovellus käyttää sitä käytännössä sellaisenaan, sillä ainoa mikä muuttuisi on palvelun osoite. Taulukossa 5 on kuvattu tietojen tallentamiseksi ja lataamiseksi käytettävä rajapinta.

*Taulukko 5. Pilvitalennuspalvelun rajapinta*

<i>Komento</i>	<i>Kuvaus</i>
<b>Pilvipalveluun tallentaminen</b>	
register.php	Luo palvelimelle uuden tiedonsiirto-session sille annetuilla parametreilla. Session luonti mahdollistaa tietojen lähettämisen palvelimelle.
unregister.php	Poistaa palvelimelle luodun tiedonsiirto-session. Voidaan käyttää, kun siirto perutaan tai se epäonnistuu.

activate.php	<p>Aktivoi palvelimelle luodun session ja päivittää sessioon liittyvien tiedostojen tiedot, kuten lukumäärän ja tiedostokoot.</p> <p>Komennon kutsuminen lähettää session luonnissa annettuun sähköpostiin QR-koodin, jonka avulla loppukäyttäjä voi aloittaa palvelimelle siirrettyjen tiedostojen lataamisen.</p>
addfile.php	<p>Valmistele uuden tiedoston lisäämisen palvelimella olevaan sessioon. Ottaa parametrina tiedostoon liittyvää metadataa ja salausinformaatiota. Palauttaa osoitteen, johon itse tiedosto tulee lähettää.</p>
updatefile.php	<p>Päivittää addfile.php-komennon palauttamien tietojen perusteella palvelimelle lähetetyn tiedoston tilan valmiiksi, jolloin se on käyttäjän ladattavissa.</p> <p>Komennon kutsumisen jälkeen tiedostoa voidaan pitää onnistuneesti palveluun tallennettuna.</p>
<b>Pilvipalvelusta lataaminen</b>	
registerapp.php	<p>Rekisteröi tiedostot lataavan laitteen palvelimella olevalle sessiolle.</p>
status.php	<p>Palauttaa session tiedot, kuten ladattavissa olevien tiedostojen lukumäärän ja yhteiskoon.</p>
getnext.php	<p>Palauttaa sessiossa seuraavana ladattavana olevan tiedoston tarkemmat tiedot, kuten osoitteen, josta itse tiedosto tulee ladata.</p>
completed.php	<p>Päivittää getnext.php-komennon palauttamien tietojen perusteella palvelimelta ladatun tiedoston tilan valmiiksi. Tulee kut-</p>

sua vasta, kun tiedosto on onnistuneesti ladattu, sen salaus purettu ja eheys varmistettu.

Komennon kutsumisen jälkeen tiedostoa voidaan pitää onnistuneesti palvelusta ladattuna, eikä se ole sen jälkeen enää saatavana.

Pilvipalveluun tallentamisen ja sieltä lataamisen toimiessa varsin eri tavalla kuin suoraan mobiililaitteesta toiseen tietoa siirrettäessä, ei sitä haluttu arkkitehtuurissa pakottaa toimimaan täysin samalla tavalla. Sen sijaan, että siirrosta vastaava TransferOperation-instanssi käyttäisi yhteyden muodostus- ja tiedonsiirtokerroksen tarjoamia rajapintoja, vastaa se itse pilvipalvelun rajapinnan käytöstä Android-alustan tarjoamien HTTP-rajapintojen avulla.

Mobiilisovellus ei sellaisenaan toimi esimerkiksi loppukäyttäjien yleisesti käyttämien pilvitallennuspalveluiden kuten Google Driven, Microsoftin OneDriven ja Dropboxin kanssa, niiden tarjotessa omat käyttörajapintansa ja kunkin toimiessa täysin omalla tavallaan. Näitä palveluja käytetään myös enemmän tiedon pidempiaikaiseen säilyttämiseen niiden omien sovelluksien kautta, kuin vain tilapäiseen tallennukseen siirron yhteydessä. Toteutuksen laajennettavuutta ajatellessa, uusien pilvipalveluiden tukeminen vaatisi siis työn jatkokehittämistä. Tämän toteuttamiseksi sovelluksen arkkitehtuuri ja rajapinnat eivät kuitenkaan aseta erityisiä rajoituksia sille.

## 6. ARVIOINTI

Tässä luvussa arvioidaan toteutettua sovellusta vertaamalla sitä työn alussa sille asetettuihin vaatimuksiin. Vaatimuksien täyttymisen lisäksi toteutusta arvioidaan vertaamalla siinä toteutettujen siirtotapojen suorituskykyä muihin tiedonsiirto-sovelluksiin.

### 6.1 Vaatimuksien täytyminen

Työssä toteutettu sovellus saatiin täyttämään työn alussa sille asetetut toiminnalliset ja ei-toiminnalliset vaatimukset. Sovelluksen käyttöliittymä mahdollistaa luvussa 3.1 määritellyissä toiminnallisissa vaatimuksissa kuvattujen toimintojen tekemisen, sisältäen samalla valmiuden tiedonsiirrosta raportointiin Piceasoftin raportointipalvelimelle.

Toteutetun sovelluksen arkkitehtuurissa ja rajapinnoissa pyrittiin helppoon siirrettävyyteen ja laajennettavuuteen. Siirrettävyyttä edesautettiin toteuttamalla keskeinen toiminnallisuus Android-kirjastomodulina, jolloin sen siirtäminen sovelluksesta toiseen on helppoa. Tämä myös mahdollistaa toiminnallisuuden jakamisen SDK:na, jos sille on koskaan tarvetta. Kirjaston ohjelmointirajapinnasta tehtiin myös mahdollisimman käyttöjärjestelmäriippumaton ja samankaltainen muiden Piceasoftin mobiilikirjastojen kanssa, jolloin vastaavan toteutuksen tekeminen iOS-käyttöjärjestelmälle on helppoa. Arkkitehtuurin laajennettavuutta tuli käytännössä mitattua toteuttaessa tiedonsiirto eri teknologioilla. Arkkitehtuuria ja sen laajennettavuutta hiottiin työn edetessä ja lopputulosta voidaan pitää kohtuullisena, sillä uuden teknologian käyttöönotto oli lopulta melko vaivatonta ja suurin osa ajasta menikin itse teknologian käyttöön perehtyessä.

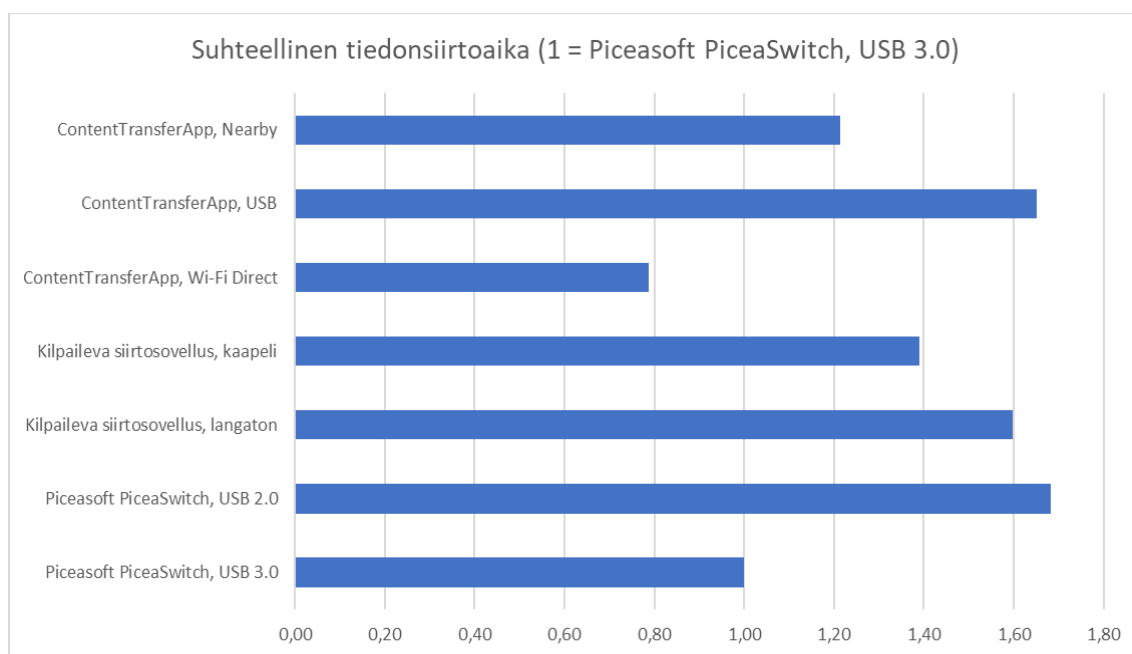
Tietoturvallisuuden ollessa keskeisessä roolilla käyttäjän tietoja käsitellessä, toteutettiin kaikki tiedonsiirto salattuja yhteyksiä käyttäen. Tämän lisäksi rajapinnan mahdollistaessa tiedonsiirto-operaation konfiguroinnin, käyttäjä voi halutessaan määrittää lisäksi käytettävän erillistä tietojen salausta. Tämän lisäksi käyttäjä voi määrittellä mitä tiivistealgoritmia käyttämällä tiedon eheys varmistetaan.

Sovelluksen käyttöliittymä jätettiin hyvin yksinkertaiseksi, sillä sen haluttiin vain mahdollistavan toiminnallisissa vaatimuksissa mainittujen toimintojen tekemisen. Väreinä käytettiin paikoin Piceasoftin brändiin kuuluvia värejä, mutta sovellusta ei muuten brändätty Piceasoftille. Toteutetun sovelluksen ulkoasu on lopulta kuitenkin helposti muokattavissa ja sovelluksen toimiessa pääasiassa toiminnallisuuden esittelyyn, ei lopullista ulkoasua ei kannattanut alkaa kehittämään, sen mahdollisesti muuttuessa täysin, jos toiminnallisuus siirretään Piceasoftin omaan PiceaOne-mobiiliapplikaatioon.

## 6.2 Suorituskyky

Toteutetulle sovellukselle suoritettiin myös yksinkertaiset suorituskykymittaukset. Mittauksissa verrattiin työssä toteutettujen eri tiedonsiirtotekniikoiden nopeuksia Piceasoftin omaan PiceaSwitch-sovellukseen sekä erääseen suoraan mobiililaitteesta toiseen tiedot siirtävään kilpailevaan mobiilisovellukseen. Suorituskykymittaukset suoritettiin siirtämällä multimediatiedostoja *Google Pixel 2 XL* -laitteesta *Samsung Galaxy S9 Plus* -laitteeseen. Molemmat laitteet tukevat USB 3.0 -standardia sekä 802.11ac Wi-Fi-standardia.

Työssä toteutetulla, ContentTransferAppiksi nimetyllä sovelluksella siirrot suoritettiin USB 3.0 -kaapelilla, Wi-Fi Directillä ja Googlen Nearby Connections API:lla, joka käyttää sisäisesti Bluetooth- sekä Wi-Fi -teknologioita. Kilpailevalla mobiilisovelluksella siirto suoritettiin USB 3.0 -kaapelilla sekä langattomasti. PiceaSwitchillä siirto suoritettiin USB 2.0 ja 3.0 -kaapeleita käyttäen. Kuvassa 14 on esitetty eri tiedonsiirtotavoilla tehtyjen siirtojen siirtoajat suhteessa PiceaSwitchillä ja USB 3.0 -kaapeleilla tehtyyn siirtoon.



**Kuva 15.** Suhteellinen tiedonsiirtoaika eri siirtotapojen välillä

Työssä toteutettu ContentTransferApp suoriutui mittauksissa hyvin. Verrattaessa mobiilisovelluksien USB 3.0 -kaapelilla tehtyjä siirtoja PiceaSwitchin USB 2.0 -kaapeleilla tehtyyn siirtoon, erot jäivät yllättävän pieniksi. PiceaSwitchin USB 3.0 -kaapeleilla tehdyn siirron ollessa paljon muita USB-tiedonsiirtoja nopeampi, voidaan olettaa suoraan mobiililaitteiden välisen USB-tiedonsiirron tapahtuneen mahdollisesti vain USB 2.0 nopeudella, kaapelin kuitenkin tukiessa USB 3.0 -standardia.

Mobiililaitteiden välillä toimivien langattomien teknologioiden erottuessa edukseen PiceaSwitchillä ja USB 2.0 -kaapeleilla tehtyyn siirtoon verrattuna, kuitenkin vain työssä

Wi-Fi Direct -teknologialla ja Androidin TCP-protokollarajapintoja käyttäen toteutettu siirto oli PiceaSwitchillä ja USB 3.0 -kaapeleilla tehtyä siirtoa nopeampi. Tietämättä sen tarkemmin Nearby Connections API:n sisäisestä toteutuksesta ja kilpailevalla mobiilisovelluksella tehdyn langattoman siirron toteutuksesta, voi erojen olettaa muodostuvan pääasiassa ohjelmallisesta toteutuksesta.

Vaikka suorituskykymittaukset antoivat varsin positiivisia tuloksia, tulee ottaa huomioon, että ne ovat lopulta vain suuntaa antavat. Tiedonsiirtoaikoihin vaikuttavat kuitenkin siirrettävän sisällön koko, käytetyt tietoturvamenetelmät ja muu ohjelmallinen toteutus, sekä erityisesti käytettyjen laitteiden laitteisto.

## 7. YHTEENVETO

Diplomityön tavoitteena oli suunnitella ja toteuttaa Android-mobiilisovellus, joka mahdollistaa mobiililaitteiden välisen tiedonsiirron. Toteutetun sovelluksen tarkoituksena oli toimia soveltuvuus selvityksenä sille, kuinka mobiililaitteiden välinen tiedonsiirto voidaan toteuttaa eri teknologioita käyttäen ilman tietokonetta. Tavoitteena oli myös hyödyntää QR-koodeja, joiden avulla mobiililaitteiden välinen tiedonsiirto voidaan aloittaa.

Työssä toteutettu sovellus saatiin täyttämään sille asetetut toiminalliset vaatimukset. Tiedonsiirto laitteesta toiseen eri tiedonsiirtotekniikoita käyttäen ja QR-koodeja hyödyntäen onnistui. Ei-toiminnallisten vaatimuksien toteuttamisessa onnistuttiin myös melko hyvin, sillä toteutuksesta saatiin siirrettävä ja SDK-käyttötapaukseenkin sopiva toteuttamalla sovelluksen keskeinen toiminnallisuus kirjastona, jonka ohjelmointirajapinnasta tuli myös täysin käyttäjärjestelmäriippumaton.

Toteutetun sovelluksen ottaminen kaupalliseen käyttöön on kuitenkin vielä kaukana, sillä sen kaupallinen käyttötapaus on vielä epäselvä ja markkinoilta löytyy myös ennestään vastaavan toiminnallisuuden tarjoavia ei-kaupallisia sovelluksia. Toteutetun sovelluksen avulla onnistuu kuitenkin toiminallisuuden esittely ja sillä tehdyt suorituskykymittaukset antoivat lupaavia tuloksia verrattuna tietokoneella tehtyyn tiedonsiirtoon. Jos toiminallisuudelle siis löytyisi kaupallinen käyttötapaus, toteutuksen toimintavarmuuden kehittämisen jälkeen voitaisiin se ottaa kirjastona helposti mukaan Piceasoftin omaan PiceaOne-mobiilisovellukseen.

## LÄHTEET

- [1] Andress, J., *The Basics of Information Security – Understanding the Fundamentals of InfoSec in Theory and Practice* 2nd Edition, Syngress, 2014
- [2] Android Source, Licenses, Saatavissa: <https://source.android.com/setup/start/licenses>
- [3] Camps-Mur, D., Garcia-Saavedra, A., & Serrano, P., *Device to device communications with WiFi Direct: overview and experimentation*, Saatavissa: [http://www.it.uc3m.es/pablo/papers/pdf/2012\\_camps\\_commag\\_wifidirect.pdf](http://www.it.uc3m.es/pablo/papers/pdf/2012_camps_commag_wifidirect.pdf)
- [4] Cisco, *Technical White Paper, 802.11ac: The Fifth Generation of Wi-Fi*, Saatavissa: <https://www.cisco.com/c/dam/en/us/products/collateral/wireless/aironet-3600-series/white-paper-c11-713103.pdf>
- [5] Committee on National Security Systems (CNSS), *CNSS Policy No. 15, Fact Sheet No. 1 National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information*, 2003, Saatavissa: <https://www.hsdl.org/?view&did=453540>
- [6] Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips, *Universal Serial Bus Specification Revision 2.0*, 2000, Saatavissa: <https://www.usb.org/document-library/usb-20-specification>
- [7] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T., *Hypertext Transfer Protocol -- HTTP/1.1*, IETF, 1999, Saatavissa: <https://tools.ietf.org/html/rfc2616>
- [8] Garlan, D. & Shaw, M., *An Introduction to Software Architecture*, School of Computer Science Carnegie Mellon University, 1994
- [9] Google Developers, *Android Debug Bridge*, Saatavissa: <https://developer.android.com/studio/command-line/adb>
- [10] Google Developers, *Android Platform Architecture*, Saatavissa: <https://developer.android.com/guide/platform/>
- [11] Google Developers, *Content providers*, Saatavissa: <https://developer.android.com/guide/topics/providers/content-providers>
- [12] Google Developers, *Create an Android library*, Saatavissa: <https://developer.android.com/studio/projects/android-library>

- [13] Google Developers, Data and file storage overview, Saatavissa: <https://developer.android.com/guide/topics/data/data-storage>
- [14] Google Developers, Develop Android apps with Kotlin, Saatavissa: <https://developer.android.com/kotlin/>
- [15] Google Developers, Fragments Overview, Saatavissa: <https://developer.android.com/guide/components/fragments>
- [16] Google Developers, Introduction to Activities, Saatavissa: <https://developer.android.com/guide/components/activities/intro-activities>
- [17] Google Developers, Save files on device storage, Saatavissa: <https://developer.android.com/training/data-storage/files.html>
- [18] Google Developers, USB host and accessory overview, Saatavissa: <https://developer.android.com/guide/topics/connectivity/usb/>
- [19] Google Developers, Wi-Fi peer-to-peer overview, Saatavissa: <https://developer.android.com/guide/topics/connectivity/wifip2p>
- [20] Hewlett-Packard Company, Intel Corporation, LSI Corporation, Microsoft Corporation, Renesas Electronics Corporation & ST-Ericsson, On-The-Go and Embedded Host Supplement to the USB Revision 2.0 Specification Revision 2.0 version 1.1a, 2012, Saatavissa: <https://www.usb.org/document-library/usb-20-specification>
- [21] Hewlett-Packard Company, Intel Corporation, Microsoft Corporation, NEC Corporation, ST-NXP Wireless, Texas Instruments, Universal Serial Bus 3.0 Specification Revision 1.0, 2008
- [22] Hoffman, P., & Schneier, B., Attacks on Cryptographic Hashes in Internet Protocols, IETF, 2005, Saatavissa: <https://tools.ietf.org/html/rfc4270>
- [23] Koskimies, K. & Mikkonen, T., Ohjelmistoarkkitehtuurit. Talentum, 2005
- [24] Lee, P., Used smartphones: the \$17 billion market you may never have heard of, Deloitte, 2016, Saatavissa: <https://www2.deloitte.com/content/dam/Deloitte/global/Documents/Technology-Media-Telecommunications/gx-tmt-prediction-used-smartphones.pdf>
- [25] McKay, K., & Cooper, D., NIST Special Publication 800-52 Revision 2 (2nd Draft) - Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations, National Institute of Standards and Technology (NIST), 2018

- [26] Mooney, J. D., Bringing Portability to the Software Process, West Virginia University Dept. of Statistics and Computer Science, Saatavissa: <https://pdfs.semanticscholar.org/ef08/695a3e437ea58f48e247f72b4bac61140c5c.pdf>
- [27] National Institute of Standards and Technology (NIST), Federal Information Processing Standards Publication 180-4: Announcing the Secure Hash Standard, 2012
- [28] National Institute of Standards and Technology (NIST), Federal Information Processing Standards Publication 197: Specification for the Advanced Encryption Standard (AES), 2001, Saatavissa: <https://csrc.nist.gov/csrc/media/publications/fips/197/final/documents/fips-197.pdf>
- [29] Nielsen, J., Usability Engineering. Morgan Kaufmann, 1994
- [30] Open Handset Alliance (OHA), Open Handset Alliance FAQ, 2007, Saatavissa: [http://www.openhandsetalliance.com/oha\\_faq.html](http://www.openhandsetalliance.com/oha_faq.html)
- [31] Oppliger, R., SSL and TLS: Theory and Practice 2nd Edition, Artech House, 2016
- [32] Padgett, J., Chen, L., Scarfone, K., Bahr, J., Batra, M., Holtmann, M. & Smithbey, R., NIST Special Publication 800-121 Revision 2 - Guide to Bluetooth Security, National Institute of Standards and Technology (NIST), 2017
- [33] Parziale, L., Britt, D. T., Davis, C., Forrester, J., Liu, W., Matthews, C. & Rossetol, N., TCP/IP Tutorial and Technical Overview, IBM, 2006, Saatavissa: <https://www.redbooks.ibm.com/pubs/pdfs/redbooks/gg243376.pdf>
- [34] Piceasoft, OnTheFly-tuotekuvaus, Saatavissa: <https://www.piceasoft.com/products-on-the-fly/>
- [35] Piceasoft, PiceaSwitch-tuotekuvaus, Saatavissa: <http://piceasoft.com/products/switch/>
- [36] Qualys SSL Labs, SSL and TLS Deployment Best Practices, 2017, Saatavissa: <https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices>
- [37] Rajeswari, H., Yegireddi, R. & Govinda, V., Performance Analysis of Hash Algorithms and File Integrity, International Journal of Computer Science and Information Technologies, Vol. 5 (6), 2014, Saatavissa: <https://ijcsit.com/docs/Volume%205/vol5issue06/ijcsit20140506103.pdf>

- [38] Rescorla, E., HTTP Over TLS, IETF, 2000, Saatavissa: <https://tools.ietf.org/html/rfc2818>
- [39] Rivest, R., The MD5 Message-Digest Algorithm, IETF, 1992, Saatavissa: <https://tools.ietf.org/html/rfc1321>
- [40] Sandoval, K., What is the Difference Between an API and an SDK, Nordic APIs, 2016, Saatavissa: <https://nordicapis.com/what-is-the-difference-between-an-api-and-an-sdk/>
- [41] Smart, N.P., Rijmen, V., Gierlichs, B., Paterson, K. G., Stam, M., Warinschi, B. & Watson, G., Algorithms, key size and parameters report – 2014, European Union Agency for Network and Information Security, 2014
- [42] The Open Web Application Security Project (OWASP), Cryptographic Storage Cheat Sheet, 2018, Saatavissa: [https://www.owasp.org/index.php/Cryptographic\\_Storage\\_Cheat\\_Sheet](https://www.owasp.org/index.php/Cryptographic_Storage_Cheat_Sheet)
- [43] Thomas, S., SSL & TLS Essentials – Securing the Web, John Wiley & Son, Inc, 2000
- [44] Thornycroft, P., White Paper: Designed for Speed: Network Infrastructure in an 802.11n World, Aruba Networks, Saatavissa: [https://www.arubanetworks.com/pdf/technology/whitepapers/wp\\_Designed\\_Speed\\_802.11n.pdf](https://www.arubanetworks.com/pdf/technology/whitepapers/wp_Designed_Speed_802.11n.pdf)
- [45] Wi-Fi Alliance, Wi-Fi Direct FAQ, Saatavissa: <https://www.wi-fi.org/discover-wi-fi/wi-fi-direct>
- [46] Wi-Fi Alliance, Wi-Fi Peer-to-Peer (P2P) Technical Specification v1.5, 2014
- [47] Woolley, M., Bluetooth 5 Core Specification, Bluetooth SIG, Saatavissa: <https://www.bluetooth.com/specifications/bluetooth-core-specification>