SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

# TYÖMAAN DOKUMENTIN-HALLINTA IOS-ALUSTALLE

TEKIJÄ: Samuli Sinokki

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

| | |
|---|---|
| Koulutusala<br>Tekniikan ja liikenteen ala | |
| Koulutusohjelma/Tutkinto-ohjelma<br>Tietotekniikan koulutusohjelma | |
| Työn tekijä(t)<br>Samuli Sinokki | |
| Työn nimi<br>Työmaan dokumentinhallinta iOS-alustalle | |
| Päiväys 6.2.2018 | Sivumäärä/Liitteet 64/0 |
| Ohjaaja(t)<br>Jussi Koistinen ja Keijo Kuosmanen | |
| Toimeksiantaja/Yhteistyökumppani(t)<br>East Dataconst Oy | |


Tiivistelmä

Opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa dokumentinhallintajärjestelmä iOS-alustalle. Dokumenteilla tarkoitetaan tässä tapauksessa erilaisia sähköisiä tiedostoja kuten Microsoft Word- ja Excel-ohjelmien käyttämiä tiedostoja. Dokumentinhallinta käsitti dokumenttien ja kansioiden lisäämisen, selaamisen, poiston, avaamisen sekä muokkaamisen. Työn tilaajana toimi East Dataconst Oy ja opinnäytteen tavoitteena oli tuoda selaimella ja Android-järjestelmässä saatavilla oleva toiminnallisuus myös iOS-alustalle. Sovellusta käyttävät työmailla työskentelevät henkilöt.

Työssä käydään läpi millaisia haasteita iOS-alusta aiheuttaa dokumentinhallintajärjestelmän toteutukselle ja miten ne voidaan ratkaista. Työssä käsitellään myös teoriapohjaa, johon dokumentinhallintajärjestelmä pohjautuu ja tutkitaan, millaisia muita sovelluksia on saatavilla samaan tarpeeseen. Järjestelmän kehittämisessä käytettiin kehitysalustana Xcode-ohjelmistoa ja ohjelmointikielenä Swift-ohjelmointikieltä. Sovelluksen toimintaa testattiin iPhone- ja iPad-laitteilla.

Dokumentinhallinta lisättiin osaksi olemassa olevaa natiivia iOS-sovellusta. Työn aikana tutkittiin mahdollisia tapoja toteuttaa halutut ominaisuudet ja päädyttiin toteuttamaan järjestelmä kahdessa osassa. Ensimmäinen osa järjestelmää on osa muuta sovellusta ja tarjoaa kaiken muun toiminnallisuuden paitsi tiedostojen muokkaamisen. Toinen osa järjestelmää on File Provider -sovelluslaajennus, jonka avulla mahdollistettiin tiedostojen muokkaus ja muokkausten lähetys palvelimelle. Laajennusta käytetään toisten sovellusten kautta. Kaikki sisältö haettiin ja muutokset päivitettiin olemassa olevalle palvelimelle. Työn lopputuloksena dokumentinhallintajärjestelmään saatiin toteutettua kaikki vaaditut ominaisuudet ja sovellus julkaistiin App Storeen, kun se saatiin valmiiksi ja testattua.



Avainsanat
iOS, Swift, Xcode, iPhone, iPad, dokumentinhallinta, File Provider

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

| | | | |
|---|---|---|---|
| Field of Study<br>Technology, Communication and Transport | | | |
| Degree Programme<br>Degree Programme in Information Technology | | | |
| Author(s)<br>Samuli Sinokki | | | |
| Title of Thesis<br>Worksite Document Management for the iOS Operating System | | | |
| Date | 6 February 2018 | Pages/Appendices | 64/0 |
| Supervisor(s)<br>Mr Jussi Koistinen, Senior Lecturer and Mr Keijo Kuosmanen, Senior Lecturer | | | |
| Client Organization /Partners<br>East Dataconst Oy | | | |


Abstract

The purpose of this thesis was to design and implement a document management system for the iOS operating system. Document in this case means different electronic files such as files used by the Microsoft Word and Excel programs. Document management included creating, browsing, deleting, opening and modifying files and directories. The thesis was commissioned by East Dataconst Oy and the object of the thesis was to provide the same functionality to the iOS operating system that was already available on web browsers and the Android operating system. The application is used by people working on different worksites.

The thesis covered the challenges that implementing a document management system faces on the iOS operating system and how those challenges can be dealt with. The thesis also covered the theory behind document management systems and examined what kind of applications were already available for the same need. The development was done with the Xcode integrated development environment and the programming language was Swift. The application was tested on an iPhone and an iPad.

The document management system was added as a part of an existing native iOS application. During the thesis, possible ways to implement the required features were examined and it was chosen that the system must be implemented in two parts. The first part of the system is a part of the existing application and it includes every feature except the modification of files. The second part of the system is a File Provider extension and it manages the modification of files and the uploading of modifications to the server. It is used through other applications. All the content was read from and all the modifications were written to a server that was already operational. As a result, all the required features were implemented to the document management system and the application was published to the App Store after it was completed and tested.



Keywords
iOS, Swift, Xcode, iPhone, iPad, document management, File Provider

SISÄLTÖ

# 1 JOHDANTO

Rakennus- ja muut vastaavat työmaat ovat monimutkaisia kokonaisuuksia ja niitä koskevat monet eri määräykset ja lait. Työmailla on usein tarvetta, ja lakien kautta pakkokin, laaja-alaiselle dokumentoinnille liittyen niin työvaiheiden toteutukseen kuin varastonhallintaan ja budjettiin. Dokumentointia suoritetaan niin valokuvaamalla kuin kirjoittamalla.

Koska dokumentteja syntyy paljon, on tärkeää, että ne pysyvät järjestyksessä, helposti saatavilla ja samalla tallessa myöhempää tarvetta varten. Paperisten dokumenttien kanssa tämä voi olla haastavaa ja se vie helposti paljon tilaa. Sähköinen dokumentinhallinta auttaa moniin ongelmiin ja helpottaa sekä nopeuttaa työtä. Sähköinen dokumentinhallinta onkin koko ajan yleistymässä. Lisäksi nykyaikana älypuhelimet ja muut mobiiliratkaisut tietojenkäsittelyssä ovat jatkuvasti kasvattamassa suosiotaan ja ohjelmistojen tulee vastata tähän mobiiliratkaisujen tarpeeseen.

Tämän opinnäytteen aiheena on sähköinen dokumentinhallinta työmaiden tarpeisiin iOS-käyttöjärjestelmälle. iOS on suosittu käyttöjärjestelmä mobiililaitteille ja se näkyy myös työmailla käytettävissä laitteissa. iOS tarjoaa omat haasteensa sujuvaan dokumentinhallintaan suljetun ja rajoitetun rakenteensa takia ja tämä opinnäyte tutkii miten sen voisi parhaiten toteuttaa tällä hetkellä saatavilla olevilla tekniikoilla.

Dokumentit, joita tässä yhteydessä käsitellään, ovat erilaisia tiedostoja, joita laitteeseen asennetuilla sovelluksilla voidaan käsitellä. Vaikka valokuvaamista käytetäänkin paljon työmailla dokumentointiin (Lehto 2016, 11), tässä opinnäytteessä valokuvia tai niiden kanssa toimimista iOS-alustalla ei käsitellä. Käsiteltävät tiedostot ovat esimerkiksi tekstitiedostoja tai taulukkolaskentaohjelmien käyttämiä tiedostoja kuten Excelin .xlsx -tiedostoja.

Dokumentinhallinta käsittää kansioiden luomisen ja dokumenttien lisäämisen, näiden selaamisen, avaamisen sekä poistamisen. Kansioita avatessa näytetään niiden sisältö. Dokumentteja avatessa ne avataan soveliaassa ohjelmassa, jos sellainen käytettävältä laitteelta löytyy. Dokumentteja tulee lisäksi pystyä muokkaamaan ja muokkaukset tallentamaan. Kaikki tapahtumat ja muutokset lähetetään palvelimelle, joka pitää yllä dokumentinhallinnan rakennetta ja sisältöä. Kaikki tieto haetaan aina palvelimelta laitteelle esitystä varten. Vaaditut ominaisuudet on laadittu yhdessä tilaajan kanssa (Collan 2017-11-22). Tämä opinnäyte käsittelee vain käyttäjän puolta ja käytetty taustajärjestelmä on valmiiksi olemassa.

Dokumentinhallinnasta ei tehty omaa sovellustaan vaan se integroitiin osaksi tilaajan olemassa olevaa iOS-sovellusta, joka tarjoaa jo muita ominaisuuksia liittyen työmaiden hallintaan. Dokumentinhallinta hyödyntää osittain sovelluksessa olemassa olevia ratkaisuja kuten kirjautumistietojen hallintaa ja tallennustilaratkaisuja. Dokumentteja, kuten muitakin asioita sovelluksessa, käsitellään työmaakohtaisesti.

## 1.1 Tilaaja

Työn tilaajana ja toimeksiantajana toimi East Dataconst Oy. East Dataconst Oy on kuopiolainen ohjelmistoalan yritys, joka kehittää ja myy moderneja ohjelmistoratkaisuja rakennusteollisuuden käyttöön. Yritys on perustettu vuoden 2013 syksyllä ja työllistää tällä hetkellä alle 10 henkilöä.

Yrityksen päätuote on työmaajärjestelmä, joka tarjoaa helppokäyttöiset ja monipuoliset työkalut työmaiden hallintaan. Se on kehitetty yhdessä alan toimijoiden kanssa ja tuettuina alustoina ovat niin tietokoneet kuin mobiililaitteet. Järjestelmän käytöstä peritään kiinteää kuukausihintaa, jonka suuruuteen ei vaikuta käyttäjien määrä. (East Dataconst Oy 2018.)

## 1.2 Vastaavia sovelluksia

Markkinoilla on jo saatavilla sovelluksia, jotka on tarkoitettu työmaiden dokumentointiin ja suunniteltu kyseisen alan tarpeisiin. Dokumentointia voi suorittaa monin eri tavoin mikä näkyy sovellusten erilaisissa toteutuksissa ja painopisteissä. Työmailla tapahtuvaan dokumentointitarpeeseen ei myöskään välttämättä tarvitse erillistä sitä varten suunniteltua sovellusta vaan yleiset dokumenttien hallinnointiin tarkoitetut pilvipalvelut voivat sopia erinomaisesti myös työmaakäyttöön. Monesti kuitenkin työmaakäyttöön erikseen suunnitellut sovellukset tarjoavat lisäksi muitakin ominaisuuksia, jotka tekevät niistä ylivertaisia, jos ominaisuuksille löytyy käyttöä.

### 1.2.1 Congrid iOS-sovellus

Congrid Oy:n iOS-sovellus tarjoaa laajan valikoiman erilaisia toimintoja rakennustyömaiden valvontaan ja hallintaan. Ominaisuuksiin kuuluu muun muassa PDF-dokumenttien ja -suunnitelmien tarkastelu ja varastointi. Sovelluksessa voi lisätä työmaille havaintoja valokuvien avulla ja lisäämällä PDF-pohjaisiin pohjapiirustuksiin merkintöjä. Havaintoja voi seurata, eli onko ne tarkastettu tai korjattu. Lisäksi sovellus tarjoaa paljon muitakin ominaisuuksia ja sen lisäksi se toimii myös offline-tilassa. (Congrid Oy 2018a.)

Congridin sovelluksessa dokumentteja on helppo selata kansioittain, sovelluksesta löytyy myös helppoja hakutoimintoja ja sovellus mahdollistaa useiden eri tyyppisten dokumenttien tallentamisen. Dokumentteja voi ladata offline-käyttöä varten laitteelle. (Congrid Oy 2018b.) Toteutuksessa on paljon samanlaista kuin opinnäytteen suunnitellussa toteutuksessa. Opinnäyte menee kuitenkin vielä pidemmälle dokumenttien kanssa toimimisessa, koska tarkoituksena on avata dokumentteja muissa sovelluksissa ja mahdollistaa niiden muokkaaminen. Congridin sovellus keskittyy muokkausominaisuuksissaan lähinnä PDF-pohjaisiin pohjapiirustuksiin ja suunnitelmiin, joita muokataan sovelluksen sisällä. Dokumenttien muokkaus kolmannen osapuolen sovelluksissa tarjoaa ylimääräisiä haasteita toteutukseen.

### 1.2.2 Fluent Valokuvadokumentointi

Fluent Progress RT Oy tarjoaa Fluent Valokuvadokumentointi -sovellusta, joka nimensä mukaisesti keskittyy työmaiden dokumentointiin pääasiassa valokuvien avulla. Dokumentointi hoidetaan lisäämällä sovellukseen pohjakuvia, joihin lisätään tarkastuspisteitä. Näihin tarkastuspisteisiin voi sitten lisätä valokuvaamalla havaintoja. Tarkastuspisteisiin ja kuviin voi lisätä lisäksi metatietoja, kuten tietoja kuvaajasta ja kuvaussuunnasta.

Sovelluksesta ei mainita onko sitä saatavilla iOS-alustalle. Valokuvaaminen on tärkeä osa työmaiden dokumentaatiota, mutta sovelluksen toiminnot ovat kuitenkin melko suppeat. Otettuja valokuvia voi lisätä osaksi raportteja ja dokumentteja, mutta sovellus ei sisällä erillistä dokumentinhallintaa. (Fluent Progress RT Oy 2018.) Sovellus ei vastaa samaan tarpeeseen kuin opinnäyte, vaikka molemmat keskittyvätkin työmaiden dokumentaatioon ja sen hallintaan.

### 1.2.3 Buildie

Buildie on Pixactor Oy:n ratkaisu työmaiden hallintaan ja dokumentointiin. Se sisältää kuvien, työmaapäiväkirjan ja dokumenttien hallinnan. Tietoja on helppo jakaa ja koostaa esimerkiksi laatukansioksi. Sovelluksen tavoitteena on tarjota kaikki tarpeellinen dokumentaatio helposti yhdestä paikasta ja huolehtia myös tietojen arkistoinnista. (Pixactor Oy 2018.)

Myös Buildien tärkein dokumentointitapa on valokuvaaminen ja monet ominaisuudet liittyvät siihen. Sovelluksessa on kuitenkin myös mahdollisuus muunlaiseen dokumentointiin kuten muistioiden luontiin. Vaikka sovellus mahdollistaa monipuolisen dokumentoinnin ja dokumenttien ylläpidon, dokumentointi tapahtuu kuitenkin sovelluksen sisällä sovelluksen käyttöliittymää käyttäen. Sovelluksen sisäisistä tiedoista voi sitten luoda PDF-tiedostoja. Sovelluksen ei kerrota pystyvän hallinnoimaan ulkoisia tiedostoja ja toimimaan tiedostopankkina niille. Myös tämä sovellus eroaa siinä opinnäytteen tavoitteena olevasta sovelluksesta. Sovelluksesta on saatavilla iOS-versio.

### 1.2.4 Dropbox

Dropbox on tiedostojen säilytykseen ja jakamiseen tarkoitettu pilvipalvelu. Dropboxin avulla voi säilyttää mitä tahansa tiedostoja. Kaikki tiedostot, jotka ladataan Dropboxiin ovat saatavilla sen jälkeen kaikilla tuetuilla laitteilla. Dropboxia voi käyttää selaimella, tietokoneen sovelluksena ja mobiililaitteilla. Tiedostoja on myös helppo jakaa muille käyttäjille ja niitä on tietysti myös helppo muokata käytettävällä laitteella. (Dropbox Inc. 2018.)

Dropboxia ei ole luotu ja tarkoitettu erikseen työmaakäyttöön, mutta se on hyvin tehty pilvipalvelu tiedostojen ylläpitoon, jota on helppo käyttää mobiililaitteilla ja yrityskäytössä. Dropboxin tarjoama toiminnallisuus on hyvin lähellä toiminnallisuutta, mitä myös opinnäytteen on tarkoitus kyetä tekemään. Dropbox olisi hyvä vaihtoehto dokumenttien hallintaan, mutta tekemällä dokumentinhallinnan itse oman sovelluksen sisään tilaaja voi tarjota kaikki työmailla tarvittavat palvelut ja toiminnot itse

ja käyttäjien tarvitsee käyttää vain yhtä sovellusta kaikkiin tarpeisiin. Käyttäjien ja käyttöoikeuksien hallinta on samalla paljon yksinkertaisempaa ja selkeämpää.

## 1.3 Lyhenteet ja määritelmät

Android = Käyttöjärjestelmä mobiililaitteille. Käytetään monien eri valmistajien laitteissa.

iOS = Apple Inc. -yrityksen kehittämä käyttöjärjestelmä mobiililaitteille. Viimeisin versio on iOS 11. Sitä käytetään vain Applen itse kehittämissä älylaitteissa iPod, iPhone ja iPad.

MacOS = Applen kotitietokoneille kehittämä käyttöjärjestelmä, jota käytetään vain Applen itse kehittämissä Mac-tietokoneissa.

Xcode = Pääasiallinen kehitysympäristö iOS-sovelluksille. Saatavilla MacOS-käyttöjärjestelmälle.

Swift = Nykyaikainen ohjelmointikieli, jota käytetään Xcoden kanssa iOS-sovellusten kehittämiseen.

Delegaatti = Tietyn luokan kanssa yhteistyössä tai puolesta toimiva luokka, joka ottaa vastaan tietoja delegoivan luokan tapahtumista ja voi reagoida niihin tai vaikuttaa delegoivan luokan toimintaan. Delegaatti pitää aina asettaa erikseen.

SDK = Software Development Kit. Kokoelma kehitystyökaluja, joiden avulla on mahdollista luoda ohjelmistoja tietylle alustalle.

Kehitysympäristö = Ohjelmisto, joka tarjoaa työkalut sovelluskehitykseen.

URL = Uniform Resource Locator. Viittaus resurssiin ja sen sijaintiin.

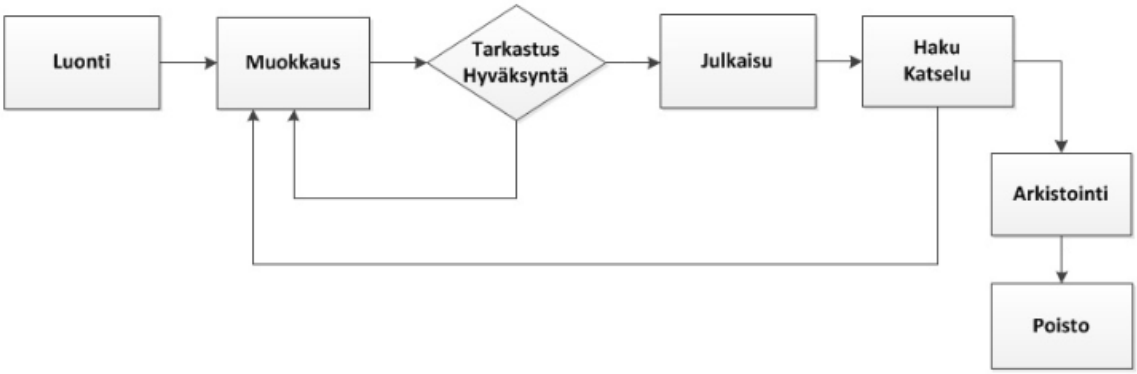
JSON = Javascript Object Notation. Yksinkertainen tiedostomuoto tiedonvälitykseen.

# 2 SÄHKÖINEN DOKUMENTINHALLINTA

Yrityksissä käsitellään paljon erilaista tietoa. Tämän tiedon käsittely tapahtuu useimmiten erilaisten dokumenttien välityksellä. On tärkeää, että kaikkia näitä syntyviä dokumentteja ja niiden sisältämää tietoa pystytään käsittelemään helposti. Lisäksi yritykset siirtyvät koko ajan suuremmissa määrin paperisista dokumenteista sähköisiin dokumentteihin. Tämä suuri määrä erilaisia sähköisiä dokumentteja tallennettuna eri paikkoihin ja niiden hyödyntäminen aiheuttavat suuria haasteita yrityksille.

Jo pelkkä tarvittavien dokumenttien löytäminen on monesti haaste ja on esitetty, että aika jota työpaikoilla käytetään vain oikeiden dokumenttien löytämiseen, on 5-50 prosenttia työajasta. Dokumenteista luodaan myös helposti uusia versioita varsinkin tilanteessa, jossa alkuperäistä on vaikea löytää. (Viitala 2010, 9.) Tämä tekee versionhallinnasta hyvin haasteellista. Dokumenttien määrän kasvaessa myös niiden varastointiin käytettävät kansiorakenteet menevät monesti monimutkaisiksi ja vaikeiksi. Näitä ja muita haasteita vastaamaan on kehitetty erilaisia sähköisiä dokumentinhallintajärjestelmiä.

Dokumentinhallinnalla tarkoitetaan dokumentin koko elinkaaren hallintaa. Se alkaa dokumentin luomisesta ja päättyy kun dokumentti arkistoidaan tai jopa tuhotaan. (Viitala 2010, 7.) Dokumentinhallintajärjestelmä on keskitetty sähköinen arkisto erilaisille dokumenteille, joka valvoo ja hallitsee dokumenttien elinkaarta. Kuviossa 1 on esitetty dokumentin elinkaari.




KUVIO 1. Dokumentin elinkaari (Viitala 2010, 7.)

Naukkarinen ja Anttonen (2010, 12) listaavat dokumentinhallintajärjestelmän tärkeimmiksi ominaisuuksiksi:

- Dokumentin eri versioiden hallinta.
- Dokumentin lukittuminen, jolloin vain yksi henkilö kerrallaan voi muokata tiettyä tiedostoa ja voidaan varmistaa, että ei tule ristiriitoja.
- Dokumenttien helppo löydettävyys.
- Oikeuksien hallinta.
- Muutosten hallinta ja dokumentointi.
- Helppokäyttöisyys, jotta käyttäjät haluavat oikeasti käyttää järjestelmää.

Edellä mainittujen ominaisuuksien lisäksi tärkeitä ominaisuuksia ovat metatietojen hallinta, dokumenttityyppien määrittely ja hallinta, hakuominaisuudet ja varmuuskopiointi (Industrial ITC Oy 2018 ja Viitala 2010, 12-15). Metatietojen hallinta tarkoittaa dokumenttiin liitettyjen tietojen, kuten luojan, muokkaajan sekä luomis- ja muokkauspäivämäärän, hallintaa.

Eri dokumentinhallintajärjestelmät laittavat erilaisia painotuksia eri osa-alueiden tärkeyteen. Metatietojen hyödyntäminen on kuitenkin yksi tärkeimmistä ominaisuuksista, joista myös muut ominaisuudet, kuten hyvin toimiva haku, ovat riippuvaisia. Myöskään hyvän käytettävyyden tärkeyttä ei tule väheksyä. Järjestelmät tarjoavat tietysti myös paljon muita ominaisuuksia, joiden tarpeellisuus riippuu järjestelmää käyttävästä organisaatiosta.

# 3 TYÖMAAN DOKUMENTAATIO

Työmailla syntyy monia erilaisia dokumentteja ja myös siellä siirrytään koko ajan enemmän sähköiseen dokumentointiin. Lehto (2016, 11) mainitsee, että työmailla dokumentoidaan muun muassa erilaisia tarkastuksia ja niiden tuloksia, TR-mittauksia, perehdytyksiä sekä sopimuksia. Myös kaikki lain vaatimat selvitykset ja tarkastukset tulee ehdottomasti dokumentoida.

Rakennusteollisuus (2018) esittelee sivuillaan malliasiakirjoja, joita on tarkoitus käyttää työturvallisuuskansion tekemiseen. Malliasiakirjat ovat pääasiassa Microsoft Word -pohjaisia lomakkeita, joihin on tarkoitus täyttää vaaditut tiedot ja tallentaa tiedosto. Malliasiakirjat ja -lomakkeet sisältävät muun muassa seuraavia tiedostoja:

- Nostotyösuunnitelma
- Työmaan aloittamisen muistilista
- Työmaan turvallisuussuunnitelma
- Työvälineen vastaanottotarkastus
- Työmaahan perehdyttäminen
- Henkilönostimen tarkastus
- Turvallisuusasiat työmaan liikennejärjestelyjen suunnittelussa.

Vaikka, kuten Lehto (2016, 11) mainitseekin, dokumentointi onkin siirtynyt koko ajan enemmän valokuvaamiseen perustuvaan dokumentointiin, voidaan jo yllä olevasta listasta huomata, että työmailla on edelleen runsaasti käyttöä erilaisille lomakkeille ja muille tekstipohjaisille dokumenteille ja tiedostoille.

## 3.1 Työmaan dokumentointi valokuvaamalla

Valokuvaaminen on siis nykyaikana tärkeä keino dokumentointiin työmailla. Valokuvia käytetään esimerkiksi todistamaan tehtyjä töitä ja käytettyjä materiaaleja. Valokuvia syntyy paljon ja niidenkin järkevä hallinta on tärkeää. Tämä opinnäyte on kuitenkin rajattu käsittelemään muita dokumenttityyppejä ja valokuvia tai valokuvausta ei käsitellä enempää työn aikana.

## 3.2 Merkitys

Kuten Salminen (2016, 12) kertoo, työmailla syntyvä dokumentaatio on hyvin tärkeää mahdollisten ristiriitatilanteiden välttämisessä ja ratkaisussa. Monesti dokumentaatiossa ilmenee puutteita ja tähän syynä voi olla esimerkiksi kiire. Myös kaikenlaiset suunnitelmien muutokset olisi hyvä dokumentoida.

Dokumenttien luonnista ei ole kuitenkaan mitään hyötyä, jos niitä tarpeen vaatiessa ei löydä. Siksi dokumentit tuleekin arkistoida huolellisesti. Salminen kertoo, että työmailla on niiden päättymisen

jälkeinen takuu- ja vastuuaika. Takuuaika on yleensä kaksi vuotta ja sinä aikana urakoitsija on vastuussa työssä ilmenneiden virheiden korjauksesta. Se lasketaan alkavaksi, kun kohde luovutetaan tai tilaaja ottaa sen käyttöön.

Vastuuaika jatkuu takuuajan jälkeen ja sen aikana urakoitsija on vastuussa virheistä, jotka ovat johtuneet esimerkiksi törkeistä laiminlyönneistä. Virheiden tulee olla sellaisia, joita tilaajan ei ole voinut kohtuudella olettaa havaitsevan vastaanottotarkastuksessa. Vastuuaika on kymmenen vuotta kohteen vastaanottamisesta. (Salminen 2016, 13.)

Tällaisten pitkien vastuuaikojen takia kunnollinen arkistointi kaikelle dokumentaatiolle on ehdottoman tärkeää, jotta voidaan myöhemmin helposti todistaa, miten työt on todellisuudessa tehty. Kun työmaillakin dokumentointi on koko ajan enemmän sähköistä, kuten aiemmin on mainittu, voi olla hyvin järkevä ratkaisu ottaa käyttöön jokin sähköinen dokumentinhallintajärjestelmä joiden ominaisuuksia esiteltiin aiemmin. Tällainen järjestelmä helpottaa dokumentoinnin ylläpitoa ja arkistointia merkittävästi, jos sitä käytetään asianmukaisesti ja henkilöstö koulutetaan käyttämään sitä.

Tälle pohjalle perustuu myös tämän opinnäytetyön aihe, jonka tarkoituksena on helpottaa dokumentointityötä iOS-alustalla ja yhdistää iOS-alustalla tapahtuva dokumentointityö kaikkien muiden alustojen työhön yhteisellä taustajärjestelmällä. Tämä taustajärjestelmä varmistaa, että samat dokumentit ovat kaikkialla saatavilla ja kaikilla alustoilla luodut dokumentit arkistoidaan samaan paikkaan.

# 4 TEKNIIKKA

Sovelluskehitys iOS-alustalle tapahtuu pääasiassa Mac-tietokoneilla ja käyttäen Xcode-ohjelmistoa. iOS-kehitykseen Windows-pohjaisella tietokoneella löytyy keinoja, mutta ne eivät ole virallisesti tuettuja. Lisäksi iOS-alustan SDK on saatavilla vain Mac-tietokoneille. Xcode ei ole ainut ohjelmisto, jolla voi kehittää toimivia iOS-sovelluksia, mutta se on Applen virallinen kehitysympäristö ja se on hyvin tuettu sekä dokumentoitu.

Kehittämisen iOS-alustalle voi aloittaa ilmaiseksi, mutta julkaistakseen sovelluksen App Storeen ja päästäkseen käsiksi edistyneempiin ominaisuuksiin, kehittäjän tulee liittyä Applen kehittäjäohjelmaan, joka maksaa 99 Yhdysvaltain dollaria vuodessa (Apple Inc. 2018a). Apple tarjoaa lisäksi erillistä kehittäjäohjelmaa yritysten sisäiseen käyttöön, joka maksaa 299 Yhdysvaltain dollaria vuodessa. Yrityksen sisäinen kehittäjälisenssi tarjoaa työkalut sovellusten helppoon jakoon ja ylläpitoon yrityksen sisällä, ilman, että sovelluksia tarvitsee julkaista julkiseen App Storeen. (Apple Inc. 2018b.)

Tässä opinnäytteessä työskentelyyn käytetään Mac mini -tietokonetta, johon asennettua Xcode-ohjelmistoa käytetään itse kehitystyöhön. Ohjelmointikielenä toimii Swift ja sovellusta testataan oikeissa fyysisissä laitteissa. Laitteina käytetään iPhone SE -puhelinta, johon on asennettu iOS 11 -käyttöjärjestelmän viimeisin versio ja iPad Mini 2 -laitetta, johon on asennettu iOS 10 -käyttöjärjestelmän viimeisin versio.

Laitteissa on eri versio iOS-käyttöjärjestelmästä, jotta voidaan varmistaa sovelluksen yhteensopivuus molempien versioiden kanssa. iOS 11 on vielä opinnäytteen tekohetkellä melko uusi versio iOS-käyttöjärjestelmästä ja tilaajan asiakkaiden keskuudessa on vielä runsaasti henkilöitä, jotka käyttävät iOS 10 -versiota. Tuki sille lopetetaan myöhemmin päätettävänä ajankohtana.

## 4.1 Xcode

Xcode on Applen kehittämä ja ylläpitämä kehitysympäristö. Sen uusin versio kirjoitushetkellä on Xcode 9. Xcode on tärkein työkalu iOS-kehitykseen. Se sisältää kaiken tarvittavan sovelluskehitykseen OSX-, iOS-, watchOS- ja tvOS-alustoille.

Tärkeimpiä Xcoden ominaisuuksia ovat

- Tehokas editori lähdekoodin muokkaamiseen.
- Tuki Swift- ja Objective C -ohjelmointikielille.
- Integrointi versionhallintaohjelmistojen ja Git Hubin kanssa.
- Sisäänrakennettu simulaattori mobiililaitteille.
- Kattavat sisäänrakennetut testausominaisuudet.
- Visuaalinen käyttöliittymän suunnittelutyökalu.
- Sisäänrakennettu dokumentaatio. (Apple Inc. 2018c.)

Xcode sisältää myös kaikki tarvittavat työkalut sovelluksien julkaisuun App Storeen ja pystyy hoitamaan automaattisesti kaikkien tarvittavien sähköisten allekirjoitusten tekemisen, kunhan kehittäjällä on voimassa oleva tilaus kehittäjäohjelmaan. Tämän lisäksi kehittäjän tarvitsee vain täyttää tietoja Applen luomalla verkkosivustolla (iTunes Connect), jonka kautta hoidetaan App Storeen julkaistavien sovellusten hallinta.

## 4.2 Swift

Swift on moderni yleiskäyttöinen ohjelmointikieli, jonka tavoitteena on olla helppokäyttöinen ja helposti ylläpidettävä kieli. Swiftillä kirjoitetun koodin tavoitteena on olla turvallista, nopeaa ja ilmaisuvoimaista. (Apple Inc. 2018d.) Se julkistettiin vuonna 2014 ja nykyinen versio kirjoitushetkellä on Swift 4. Swift tulee Xcoden mukana ja on nykyään tärkein kieli iOS-kehitykseen, mutta Objective C -kieltä tuetaan myös edelleen Xcodessa.

Swift-kielen tärkeimpiä ominaisuuksia ovat

- Automaattinen muistinhallinta.
- Ei tarvetta puolipisteille.
- Moduulit, jotka tarjoavat nimiavaruudet ja poistavat tarpeen ylätunnisteille.
- Puhdas syntaksi, jota on helppo ylläpitää.
- Edistynyt vuonhallinta.
- Tehokas sisäänrakennettu virheenkäsittely.
- Suunniteltu turvalliseksi. (Apple Inc. 2018c.)

## 4.3 iOS

iOS on Applen kehittämä käyttöjärjestelmä mobiililaitteille. Sitä käytetään vain Applen kehittämissä laitteissa. Tuettuja laitteita ovat iPhone, iPad ja iPod. iOS on toiseksi suosituin käyttöjärjestelmä mobiililaitteille Androidin jälkeen (Statista 2018). iOS on suunniteltu käytettäväksi kosketusnäytöltä ja se tukee useita erilaisia kosketuseleitä, joilla on erilaisia merkityksiä riippuen kontekstista. Käyttöjärjestelmässä on myös laajat helppokäyttöisyystoiminnot auttamaan ihmisiä, joilla on käyttöä vaikeuttavia vammoja tai rajoitteita. Käyttöjärjestelmä tukee moniajoa ja siihen liittyvät ominaisuudet ovat parantuneet uusien versioiden myötä.

Uusin versio iOS-käyttöjärjestelmästä kirjoitushetkellä on iOS 11. Se on tarjolla iPhone 5S ja uudempiin iPhone-malleihin, kuudennen sukupolven iPod-laitteisiin sekä useisiin eri iPad-malleihin. iOS 11 toi runsaasti uusia ominaisuuksia iOS-käyttöjärjestelmään.

Tärkeimpiä uusia ominaisuuksia iOS 11-käyttöjärjestelmässä ovat

- Uudistettu App Store -kauppapaikka.
- Muokattava ohjauskeskus.
- Uusi Tiedostot-sovellus kaikkien käyttäjän tiedostojen hallintaan.
- Parannuksia iPadin moniajo-ominaisuuksiin.
- Laajempi Apple Pencil -tuki iPad Pro -laitteissa.
- Paranneltu Siri-avustaja. (Apple Inc. 2018e.)

Opinnäytetyön aiheen kannalta merkittävin uudistus on uusi Tiedostot-sovellus, joka muuttaa merkittävästi tapaa, jolla iOS-käyttöjärjestelmässä käsitellään tiedostoja. Päivitys antoi uusia ja uudisti vanhoja työkaluja, joilla sovellukset voivat työskennellä käyttäjän tiedostojen kanssa tai jakaa tiedostoja muiden sovellusten käyttöön. Tarve iOS 10 -tuelle kuitenkin rajoittaa mahdollisuutta hyödyntää kaikkia uusia työkaluja ja tekniikoita.

## 4.4 iPhone SE

iPhone SE on neljätuumaisella näytöllä varustettu Applen valmistama älypuhelin, joka julkistettiin 21. maaliskuuta 2016. Se on ulkonäöltään lähes identtinen iPhone 5s -puhelimeen verrattuna. Suorituskyvyltään iPhone SE vastaa iPhone 6s -puhelinta. Neljätuumaisen näytön resoluutio on 640x1136. Puhelimesta löytyy 3,5 mm kuulokeliitäntä ja Lightning-liitäntä latausta ja tietokoneyhteyttä varten. SIM-korttina käytetään nano-SIM-korttia. Laitteen etukamera on 1,2 megapikseliä ja takakamera on 12 megapikseliä. Laitteen kotipainikkeeseen on integroitu Touch ID -sormenjälkitunnistin. (Lainiala 2016.)

Puhelimesta oli julkaisun aikaan saatavilla 16 Gt ja 64 Gt tallennustilalla varustetut mallit, mutta nykyään saatavilla on vain 32 Gt ja 128 Gt mallit. Opinnäytetyön aikana käytettävä malli on 32 Gt versio.

iOS-sovelluksen testauksen kannalta iPhone SE on erinomainen valinta, koska se on pienimmällä näytöllä varustettu nykyään myynnissä oleva iPhone. Tämän ansiosta on tätä puhelinta käyttäen helppo varmistaa, että tuotettavan sovelluksen käyttöliittymä skaalautuu oikein pienelle näytölle ja käyttö on sujuvaa. Tärkeä asia pienellä näytöllä sovellusta testatessa on varmistaa, että näytöllä näkyvät tekstit eivät katkea ja mene näkymättömiin ruudun ulkopuolelle.

## 4.5 iPad mini 2

iPad mini 2 on Applen 12 marraskuuta 2013 julkaisema taulutietokone. Se on saatavilla Wifi-yhteydellä tai Wifi- ja mobiilidatayhteydellä varustettuna versiona. Tallennustilaa laitteeseen saa 16, 32, 64 tai 128 Gt. iPad mini 2 sisältää 7,9 tuuman Retina-näytön, jonka resoluutio on 2048x1536. Laitteen etukamera on 1,2 megapikseliä ja takakamera on 5 megapikseliä. iPad mini 2:sta löytyy myös 3,5 mm kuulokeliitäntä ja yksi Lightning-liitäntä kuten iPhone SE -puhelimesta. iPadissa ei ole sormenjälkitunnistinta.

iPadin avulla saadaan testauksen aikana varmistettua sovelluksen toimivuus isommalla näytöllä ja varmistettua kaikkien näyttöelementtien oikeanlainen skaalautuminen. Jos sovellus on tarkoitus tehdä iPad-yhteensopivaksi, on tärkeää varmistaa, että varsinkin kaikki modaalisesti aukeavat ikkunat toimivat oikein, koska niiden toiminnassa on eroja iPhone- ja iPad-laitteiden välillä ja nämä erot johtuvat juuri näytön koosta.

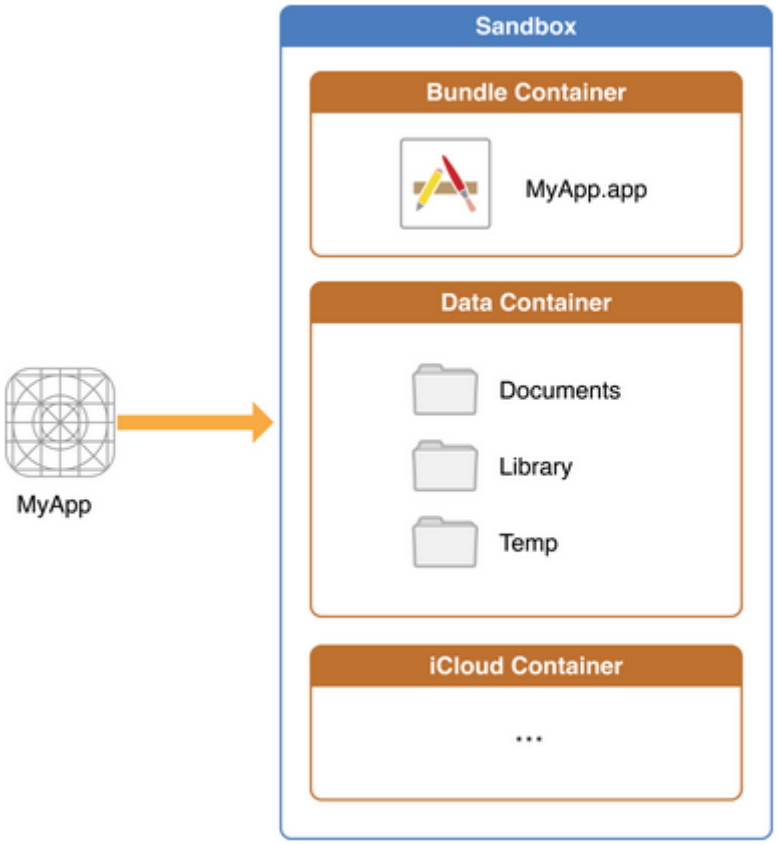
Monet ikkunat, jotka aukeavat iPhonella koko ruudulle, voivatkin aueta iPadilla pienempään ruutuun muun käyttöliittymän päälle. Näissä tapauksissa on ohjelmoijan tehtävä varmistaa, että sovellus tietää missä kohtaa ruutua pienempi ikkuna on tarkoitus esittää. Jos tätä ei ole varmistettu, sovellus voi kaatua.

# 5 IOS-ARKKITEHTUURIN HAASTEET

iOS-käyttöjärjestelmä on alusta alkaen suunniteltu tietoturvalliseksi ja käyttäjän yksityisyyttä suojelevaksi. Kyseinen filosofia näkyy monissa suunnitteluratkaisuissa ja ominaisuuksissa. Vaikka nämä ratkaisut ovat perusteltuja, ne rajoittavat kehittäjien mahdollisuuksia toteuttaa erilaisia ominaisuuksia ja vähentävät sovelluksien mahdollisuuksia. Vaikka tämä on usein hyvä asia epärehellisten kehittäjien sovelluksia vastaan, myös rehelliset kehittäjät joutuvat usein ongelmiin rajoitusten kanssa, joita voi olla vaikea kiertää.

## 5.1 Tiedostojärjestelmä

iOS-sovelluksilla ei ole suoraa pääsyä käyttöjärjestelmän tiedostojärjestelmään. Jokaiselle sovellukselle luodaan asennusvaiheessa sovelluskohtainen hiekkalaatikko, jonka sisällä kaikki sovelluksen tiedot ja tiedostot sijaitsevat.




KUVA 1. Sovelluksen hiekkalaatikko iOS-käyttöjärjestelmässä. (Apple Inc. 2018g.)

Kuva 1 esittää tarkemmin sovelluksen hiekkalaatikon rakennetta. Sovelluksella on pääsy vain näihin tiedostoihin ja sillä ei ole oikeutta käsitellä tiedostoja, jotka sijaitsevat tämän hiekkalaatikon ulkopuolella tai muiden sovelluksien hiekkalaatikoissa. Sovellus voi vapaasti tallentaa omia tiedostojaan oman hiekkalaatikkonsa sisällä, mutta sen tulisi noudattaa Applen laatimia ohjeita, mitkä tiedostot kuuluvat mihinkin kansioon. Kuvassa esitetyt kansiot kuuluvat jokaisen sovelluksen hiekkalaatikkoon. (Apple Inc. 2018g.)

Tämä rakenne vaikeuttaa tiedostojen jakamista sovelluksien välillä ja sovelluksien välistä kommunikaatiota. Tämä on hyvin kriittinen haaste sovelluksessa, jonka on tarkoitus käsitellä tiedostoja ja mahdollistaa niiden avaaminen ja muokkaaminen muissa sovelluksissa ja sitten käsitellä niitä muokkaamisen jälkeen taas alkuperäisessä sovelluksessa.

Näiden rajoitustenkin kanssa sovelluksien on kuitenkin mahdollista pyytää joitain tietoja, kuten käyttäjän kuvia, hiekkalaatikon ulkopuolelta julkisten iOS-käyttöjärjestelmän käyttöliittymien kautta. Sellaisissa tapauksissa käyttöjärjestelmä käyttää apusovelluksia, jotka hoitavat interaktion kohdetietojen kanssa ja jakavat valitut tiedot sovelluksen käyttöön. (Apple Inc. 2018g.) Nämä käyttöliittymätkin ovat kuitenkin toiminnallisuudeltaan rajallisia ja eivät tarjoa keinoja kahden kolmannen osapuolen sovelluksen väliseen interaktioon.

## 5.2 Tiedostojen avaaminen muissa sovelluksissa

iOS-käyttöjärjestelmästä löytyy kuitenkin valmiiksi keinoja, joilla sovelluksen hiekkalaatikossa sijaitsevia tiedostoja voi avata muilla sovelluksilla ja muiden sovelluksien tiedostoja voi avata omalla sovelluksella. Näillä keinoillakin on kuitenkin rajoituksensa, jotka asettavat edelleen suuria haasteita kehittäjälle, jonka tarkoituksena on käsitellä tiedostoja sovelluksellaan.

### 5.2.1 UIActivityViewController

UIActivityViewController on luokka iOS SDK:ssa, jonka kautta voi tarjota standardeja palveluja sovelluksen sisällä käyttäjälle. Se on ollut osa SDK:ta iOS 6.0 -versiosta asti. Palveluiden kohteena on aina jokin yksi tai useampi sovelluksen sisäinen asia, esimerkiksi teksti tai kuva.

Käyttöjärjestelmän tarjoamat palvelut ovat esimerkiksi kohteiden kopiointi leikepöydälle ja kohteiden jako sosiaaliseen mediaan, tekstiviestin kautta tai sähköpostilla. Käyttäjän asentamat sovellukset voivat määritellä lisää palveluja, jotka voivat näkyä valittavana riippuen kohteesta, jota UIActivityViewController on jakamassa. (Apple Inc. 2018h.)

Kuva 2 esittää mille UIActivityViewController näyttää, kun se luodaan ja avataan näytölle alustettuna jollakin jaettavalla asialla, tässä tapauksessa kuvalla.


KUVA 2. UIActivityViewController (Jayesh Kawli 2016-05-28.)

UIActivityViewControllerilla on mahdollista jakaa sovelluksen hiekkalaatikon sisällä sijaitsevia tiedostoja muille sovelluksille eli hiekkalaatikon ulkopuolelle. Luokka alustetaan jaettavan tiedoston paikallisella hiekkalaatikon sisäisellä URL-osoitteella ja avautuvassa ikkunassa esitetään kaikkien käyttöjärjestelmän tarjoamien palvelujen lisäksi kaikki sovellukset, jotka ovat rekisteröineet kyvyn avata jaettavan tiedoston tyyppisiä tiedostoja. Jos käyttäjä valitsee jonkin näistä sovelluksista, tiedosto avataan kyseiseen sovellukseen katseltavaksi.

### 5.2.2 UIDocumentInteractionController

UIDocumentInteractionController on samankaltainen luokka iOS SDK:n sisällä kuin UIActivityViewController, kun käsitellään tiedostoja. Se on ollut saatavilla jo iOS 3.2 -versiosta alkaen. Se mahdollistaa tiedostojen, joita oma sovellus ei kykene avaamaan, esikatselun, kopioinnin ja avaamisen. Se on tarkoitettu vain tiedostojen kanssa toimimiseen toisin kuin UIActivityViewController, joka kykenee toimimaan, vaikka pelkän merkkijonon kanssa. (Apple Inc. 2018i.)

UIDocumentInteractionController sisältää hyvin samankaltaisen valikon (kuva 2) tiedoston avaamiseen toisessa sovelluksessa, kuin UIActivityViewController, mutta tarjoaa myös mahdollisuuden esikatsella tiedostoa suoraan siirtymättä toiseen sovellukseen. Sovelluksen kehittäjällä on mahdollisuus valita tiedostoa avatessa, näytetäänkö tiedostosta suoraan esikatselunäkymä, valikko, josta voi valita

esikatselun tai toiseen sovellukseen avaamisen, vai valikko, josta tiedoston voi ainoastaan avata toiseen sovellukseen. Tiedostosta riippuen kaikki vaihtoehdot eivät ole välttämättä saatavilla ja tämän päättelee UIDocumentInteractionController, kun sitä alustetaan käyttöä varten. (Apple Inc. 2018j.)

UIDocumentInteractionController ja UIActivityViewController ovat melko yksinkertaisia käyttää ja vaativat vähän alustustyötä, jos tarkoitus on vain avata tiedostoja muissa sovelluksissa. Poikkeuksena on tilanne, jossa halutaan näyttää esikatselunäkymä UIDocumentInteractionControllerin kautta. Se vaatii delegaatin asettamista kyseiselle luokalle ja tämän delegaatin täytyy toimittaa näkymä, johon esikatselu avataan. UIDocumentInteractionController voi vaatia delegaattia myös muiden toimintojen kanssa. (Apple Inc. 2018j.)

### 5.2.3 Ongelmat

Edellä esitellyt luokat ovat käteviä ja helppokäyttöisiä työkaluja tiedostojen tarkasteluun, mutta niin UIActivityViewControllerissa kuin UIDocumentInteractionControllerissa on kuitenkin yksi suuri heikkous, kun tarkoituksena on avata tietyn sovelluksen hallinnoimia tiedostoja ja mahdollistaa niiden muokkaaminen ja tallentaminen.

Molemmat luokat siirtävät tiedostoa toiseen sovellukseen avattaessa toisen sovelluksen luettavaksi vain kopion alkuperäisestä tiedostosta, joka sijaitsee alkuperäisen sovelluksen hiekkalaatikossa, ja tämä kopio on myös lukittu niin, että vain lukeminen onnistuu. Muokkauksia tehdäkseen käyttäjän tulee tallentaa avoinna olevasta tiedostosta uusi kopio saatavilla olevaan tallennustilaan, joka usein tarkoittaa avatun sovelluksen omaa hiekkalaatikkoa. Alkuperäisen sovelluksen hiekkalaatikkoon, jossa alkuperäinen tiedosto sijaitsee, tallentaminen ei tietenkään onnistu, koska se rikkoisi koko hiekkalaatikkoihin jaetun järjestelmän ideaa.

Tiedostoja ei voi siis avata paikaltaan suoraan sovelluksen hiekkalaatikosta toiseen sovellukseen, niin että uusi sovellus lukisi tiedoston suoraan alkuperäisen sovelluksen hiekkalaatikosta, jolloin muokkaukset suoraan alkuperäiseen tiedostoon olisivat mahdollisia. Toisella sovelluksella ei ole yksinkertaisesti mitään oikeutta käsitellä alkuperäisen sovelluksen hiekkalaatikon sisällä olevia tiedostoja, joten tiedostoista on pakko tehdä kopioita muiden sovellusten käyttöön. Tämä taas vie kaikki mahdollisuuden muokata tiedostoja helposti ja järkevästi.

iOS 11 toi muiden tiedostonhallintaan tuotujen muutosten myötä myös mahdollisuuden avata sovelluksen hiekkalaatikosta Documents-kansio muiden sovellusten käyttöön. Muilla sovelluksilla on siis muutoksen myötä mahdollisuus lukea ja kirjoittaa tietyn sovelluksen hiekkalaatikon sisälle. Tämä onnistuu lisäämällä muutama asetusarvo kehitettävän sovelluksen plist-asetustiedostoon. (Apple Inc. 2018k.)

Tämä muutos ei kuitenkaan vaikuta UIActivityViewController- tai UIDocumentInteractionController-luokan toimintaan ja vaikka Documents-kansio olisikin avattu muiden sovellusten käyttöön, kyseiset

luokat luovat edelleen lukittuja kopioita tiedostoista, jotka on tarkoitus avata toiseen sovellukseen, vaikka ne sijaitsisivat avatussa Documents-kansiossa.

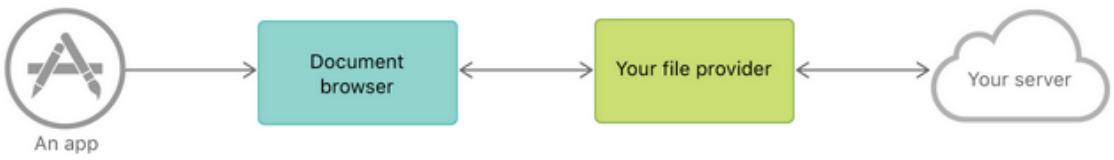
Avatussa kansiossa sijaitsevat tiedostot on kyllä mahdollista avata toisessa sovelluksessa niin, että niitä voi muokata, mutta se vaatii niiden avaamista sen sovelluksen käyttöliittymästä, johon ne halutaan avata. Vaikka sovelluksen hiekkalaatikko olisikin avattu muiden sovellusten käyttöön, ei iOS SDK tarjoa keinoja avata siellä sijaitsevia tiedostoja muihin sovelluksiin suoraan kyseisestä sovelluksesta niin, että muokkaus olisi mahdollista. Tiedostojen muutoksien ylläpito palvelimen suuntaan on myös hyvin vaikeaa, koska mikään prosessi ei vahdi tiedostoja ja niiden muutoksia.

### 5.2.4 File Provider

iOS SDK:ssa on kuitenkin yksi keino, joka mahdollistaa sovelluksen hallinnoimien tiedostojen helpon avaamisen ja muokkaamisen muissa sovelluksissa niin, että myös palvelin on helppo pitää mukana muutoksissa ja ajan tasalla. Tämä keino on FileProvider-ohjelmakehikko. Se mahdollistaa sovelluksen hallinnoiman kansio- ja tiedostorakenteen avaamisen muiden sovellusten käyttöön. Tämä tapahtuu lisäämällä pääsovellukseen File Provider -sovelluslaajennus. (Apple Inc. 2018k.) File Provider -laajennuksen tarjoama kansiorakenne sijaitsee eri paikassa kuin pääsovelluksen hiekkalaatikon kansiorakenne ja laajennuksen toiminta ei vastaa Documents-kansion avaamista, josta kerrottiin edellisessä luvussa.

Sovelluslaajennukset mahdollistavat sovelluksen toiminnallisuuden laajentamisen sovelluksen ulkopuolelle ja käyttäjät voivat käyttää laajennuksia muita sovelluksia käyttäessään. Sovelluslaajennukset on suunniteltu erilaisia tiettyjä tehtäviä varten. File Provider -laajennuksen lisäksi tarjolla olevia laajennuksia ovat muun muassa kustomoidut näppäimistöt ja kuvanmuokkaus-laajennukset Kuvat-sovellukseen. On tärkeää valita oikean tyyppinen laajennus tiettyä tarvetta varten.

Sovelluslaajennus on erillinen kokonaisuus sovelluksesta, mutta sovelluslaajennuksia ei voi luoda itsenäisinä, vaan ne vaativat aina pääsovelluksen, jonka toimintaa ne laajentavat ja ne jaellaan sen mukana App Storessa. Niitä kuitenkin ajetaan käyttöjärjestelmässä itsenäisesti riippumatta pääsovelluksen tilasta. (Apple Inc. 2018l.)




KUVA 3. File Provider -sovelluslaajennuksen toimintaperiaate (Apple Inc. 2018k.)

Kuva 3 selventää File Provider -sovelluslaajennuksen toimintaperiaatetta. Käyttäjä käyttää jotain kolmannen osapuolen sovellusta, joka avaa tiedostojen valintaa varten iOS-käyttöjärjestelmän tarjoa-

man käyttöliittymän (Document browser). Tämän käyttöliittymän kautta on mahdollista avata sovellukseen lisätty File Provider -sovelluslaajennus, joka pitää yhteyttä sovelluksen ja sen laajennuksen taustalla olevaan verkkopalveluun, joka tarjoaa laajennuksen varsinaisen sisällön.

File Provider -sovelluslaajennuksen taustalla toimii siis FileProvider-ohjelmakehikko. Tämä ohjelmakehikko tarjoaa seuraavanlaista toiminnallisuutta:

- Kyky luoda paikanpitäjiä verkossa oleville tiedostoille, jotka ladataan vain tarvittaessa.
- Ottaa vastaan lukuyritykset kolmannen osapuolen sovelluksesta, jotta tiedostot voidaan ladata ennen kuin luku oikeasti tapahtuu.
- Aktivoi ilmoituksen kolmannen osapuolen sovelluksen kirjoitustapahtumien jälkeen, jotta muutokset voidaan ladata takaisin palvelimelle.
- Mahdollisuus selata kansioita ja tiedostoja.
- Mahdollisuus erilaisiin toimintoihin kuten tiedostojen tuontiin, poistoon ja uudelleennimeämiseen. (Apple Inc. 2018k.)

Tämä laajennus on kuitenkin tarkoitettu varsinaisesti vain tilanteisiin, joissa tiedostot säilytetään etäällä sijaitsevalla palvelimella. Paikallisten tiedostojen jakoon kannattaa harkita Documents-kansion jakamista tai muita keinoja.

### 5.2.5 File Providerin ongelmat

Kuten viime kappaleessa luetelluista ominaisuuksista voi huomata, File Provider -laajennus tarjoaa käytännössä kaikki tarvittavat ominaisuudet, joita järkevään tiedostojen muokkaamiseen voi tarvita tämän opinnäytteen aiheena olevassa sovelluksessa. Siinäkin on kuitenkin omat ongelmansa ja haasteensa.

Yksi ongelma on File Providerin ja yleensä sovelluslaajennusten toimintaperiaate. Niitä käytetään avaamalla ne muista sovelluksista ja ne toimivat itsenäisesti niiden isäntäsovelluksesta riippumatta. Tämä estää sujuvan käyttökokemuksen rakentamisen isäntäsovellukseen. Isäntäsovellus voi selata palvelimella sijaitsevia tiedostoja samalla lailla kuin laajennus ja avatakin niitä muihin sovellukseen aiemmin esitellyillä tavoilla eli käyttäen avuksi UIActivityViewControlleria tai UIDocumentInteractionControlleria. Tiedostot aukeavat kuitenkin lukittuina, kuten aiemmin on mainittu.

Jos tiedostoja haluaisikin muokata, täytyy avata jokin toinen sovellus, jolla tiedostoja on mahdollista muokata, kuten Microsoft Word ja avata File Provider -laajennus ja valita sama tiedosto sieltä, jolloin se aukeaa Wordiin siten, että sitä voi muokata. Tämä pakottaa vaihtelemaan sovellusten välillä ja vähentää käyttömukavuutta ja sujuvuutta. Paras ratkaisu olisi, kun tiedostot saisi avattua suoraan isäntäsovelluksesta toiseen sovellukseen muokkaustilassa. Sovelluslaajennusta käyttämällä tätä toiminnallisuutta ei ole kuitenkaan mahdollista toteuttaa ja vaikuttaa siltä, että tällä hetkellä se ei ole ollenkaan mahdollista iOS-käyttöjärjestelmässä lukuun ottamatta muutamaa erikoistunutta poikkeusta, jotka kiertävät käyttöjärjestelmän rajoitteet verkkopalvelujen avulla.

Toinen haaste liittyy File Provider -laajennuksen toteutukseen. Tiedostojenhallinta koki suuria muutoksia, kun iOS 11 julkaistiin ja samalla File Provider -laajennuksen toteutustapa muuttui dramaattisesti. Tämä aiheuttaa haasteita, jos tarkoituksena on luoda File Provider -laajennus, joka tukee niin iOS 10 kuin 11 -versiota.

iOS 10 -versiossa FileProvider-ohjelmakehikko oli vain puolet File Provider laajennuksesta ja ajoi taustajärjestelmän virkaa, joka huolehti, että tarvittavat tiedostot on ladattu levylle silloin, kun niitä tarvitaan ja muusta tiedostoihin liittyvistä operaatioista. Tämän lisäksi piti luoda toinen laajennus, Document Picker View Controller Extension, joka huolehti itse käyttöliittymästä. Näiden yhdistelmä loi koko käyttökokemuksen.

iOS 11 -versiossa tämä muuttui niin, että FileProvider-ohjelmakehikko hoitaa kaikki toimet kansioiden selaamisesta tiedostojen lataamiseen levylle ja käyttöjärjestelmä toimittaa vakioidun käyttöliittymän, jota ei voi itse muokata. Tämän lisäksi iOS 11 -version mukana tuli uusi keino selata dokumentteja ja File Provider -laajennuksia eli UIDocumentBrowserViewController. Vanha keino eli UIDocumentPickerViewController on myös edelleen käytettävissä. Merkittävää tässä on, että edellä mainittu uusi keino näyttää vain File Provider laajennukset, jotka on tehty uudella iOS 11 -version vaatimalla tavalla ja myöhemmin mainittu vanhempi tapa osaa näyttää niin uudella kuin vanhalla tavalla tehdyt laajennukset. (Apple Inc. 2018m.)

Molempia iOS-versioita tukevan File Provider -laajennuksen luominen vaatii paljon ylimääräistä työtä ja ylimääräistä haastetta tuo uusimman Xcoden toiminta. Xcode ei tarjoa enää valmista pohjaa vanhemmalle tavalle luoda File Provider -laajennus ja ainoa pohja on uudelle tavalle. FileProvider-ohjelmakehikko on molemmissa versioissa sama, mutta Xcoden kautta ei ole enää mahdollista luoda pohjaa laajennukselle, joka toimittaa iOS 10 -version itse luotavan käyttöliittymän. Vaadittavan laajennuksen saa luotua käsin, mutta se vaatii hieman säätämistä ja vaivannäköä.

Tämän lisäksi pitää olla huolellinen eri metodeja käytettäessä FileProviderin sisällä, koska osa ei ole saatavilla iOS 10 -versiossa ja osa on vanhentunut iOS 11 -versiossa. Eri iOS-versioille tarkoitetut laajennukset on mahdollista saada toimimaan käyttämällä vain yhtä ja samaa FileProvider-ohjelmakehikkoa molempien versioiden pohjalla, mutta se vaatii tarkkuutta ja ylimääräistä työtä.

# 6 TOTEUTUS

Tilaajan vaatimat ominaisuudet dokumentinhallintaan olivat siis

- Kansioiden luonti
- Tiedostojen lisäys
- Kansioiden ja tiedostojen selaaminen
- Kansioiden ja tiedostojen poisto
- Tiedostojen avaaminen toiseen ohjelmaan
- Tiedostojen muokkaus
- Muokkausten päivitys palvelimelle. (Collan 2017-11-22.)

Nämä vaatimukset mielessä ja mahdollisten toteutustapojen tutkimisen jälkeen sovelluksesta päätettiin tehdä kaksiosainen. Ensimmäinen osa dokumentinhallinnasta käsittää suurimman osan ominaisuuksista ja se integroidaan saumattomasti osaksi tilaajan olemassa olevaa iOS-sovellusta, jonka kirjoittaja on tehnyt aiemman harjoittelun aikana tilaajalle ja tuntee sen toiminnan kauttaaltaan. Sovelluksessa on olemassa valikko, josta valitaan mitä sovelluksessa halutaan tehdä ja dokumentinhallinta avataan luontevasti tämän valikon kautta.

Tämä osio sovelluksesta sisältää kaiken muun toiminnallisuuden paitsi tiedostojen muokkaamisen ja muokkausten päivityksen palvelimelle. Tiedostojen muokkaamisen mahdollistaa sovelluksen toinen osa, joka on aiemmassa luvussa mainittu File Provider -sovelluslaajennus. Tätä laajennusta on pakko käyttää avaamalla se toisesta sovelluksesta, joten se muodostaa selkeästi erillisen kokonaisuuden muista ominaisuuksista, joita käytetään tilaajan sovelluksen kautta. Sovelluslaajennuksen kanssa päätettiin luopua iOS 10 -yhteensopivuudesta, mutta muu sovellus toimii iOS 10 ja 11 -versioilla.

Kaikki toiminnot kuten kansioiden lisäys ja kansioiden sisällön lataus keskustelevat palvelimen kanssa, jossa säilytetään kaikki tieto ja jonne kaikki muutokset tietoihin tulee aina lähettää. Laitteella säilytetään vain minimimäärä tietoa, joka vaaditaan käyttöliittymän tarpeisiin.

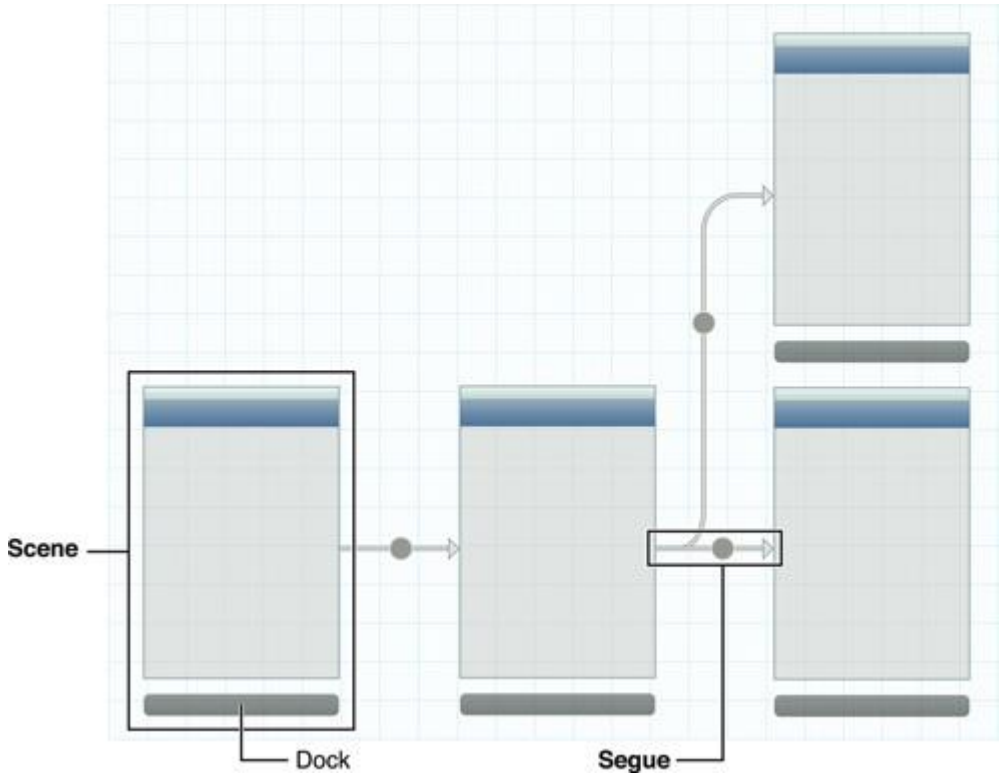
Sovellukseen luotavasta dokumentinhallintajärjestelmästä ei tule aivan täysimittaista sähköistä dokumentinhallintajärjestelmää, kuten kappaleessa 2 esiteltiin. Sovellus kyllä tarjoaa keskitetyn tallennuspaikan dokumenteille ja hallitsee niiden elinkaarta ja ylläpitää metatietoja jokaisesta tiedostosta, mutta käyttäjillä on hyvin vapaat kädet tiedostojen nimeämisen ja kansiorakenteen suhteen. Käyttäjät voivat myös melko vapaasti lisätä ja poistaa tiedostoja. Myöskään hakutoiminnot eivät ole yhtä monipuoliset kuin täysimittaisessa dokumentinhallintajärjestelmässä.

Nämä seikat antavat käyttäjille enemmän vapautta, mutta myös vastuuta ja väärinkäytökset ovat mahdollisia. Toteutettavaa järjestelmää voisi kuvailla käyttäjän kannalta kevennetyksi dokumentinhallintajärjestelmäksi. Järjestelmään olisi voinut myös helposti lisätä ominaisuuksia, jotka tekisivät siitä järeämmän dokumentinhallintajärjestelmän, mutta tässä toteutettava järjestelmä oli tilaajan ja sitä myöten selvästi myös tilaajan asiakkaiden toive.

## 6.1 Käyttöliittymä

Käyttöliittymä toteutettiin Xcoden tarjoamalla Interface Builder -työkalulla, joka tarjoaa graafisen käyttöliittymän sovelluksen käyttöliittymän suunnitteluun. Graafisesti suunniteltuun käyttöliittymään lisättiin lisäksi jonkin verran elementtejä ohjelmallisesti graafisen toteutuksen jälkeen.

Käyttöliittymän suunnittelun pohjalla Interface Builder -työkalussa on Storyboard. Storyboard on visuaalinen esitys koko sovelluksen käyttöliittymästä eli eri näkymistä ja niitä yhdistävistä siirtymistä, jotka siirtävät käyttäjän uuteen näkymään. (Apple Inc. 2018n.)




KUVA 4. Storyboardin toimintaperiaate (Apple Inc. 2018n.)

Kuva 4. selventää Storyboardin toimintaa. Siihen lisätään näkymiä, kuvassa nimellä Scene, joihin voi sitten lisätä graafisen käyttöliittymän avulla erilaisia elementtejä kuten tekstikenttiä, nappeja, kuvia ja taulukoita. Näkymät eli Scenet ovat aina jotain tyyppiä, jolle löytyy vastaava luokka iOS SDK:sta. Yleensä luokka on UIViewController tai jokin tämän luokan aliluokka. Näistä luokista voi luoda omia aliluokkia, jotka sitten liitetään yhteen jonkin näkymän kanssa. Kun näkymä ja jokin konkreettinen luokka on yhdistetty toisiinsa, näkymän elementit voi yhdistää kyseiseen luokkaan käyttämällä liitoksia, joista käytetään nimitystä outlet.

Outletit luovan käyttöliittymän elementeistä muuttujia luokan sisälle ja mahdollistavat näkymissä olevien elementtien käsittelyn lähdekoodissa. Näin saadaan graafinen käyttöliittymä yhdistettyä lähdekoodiin helposti käytettäväksi ja muokattavaksi. Esimerkiksi käyttöliittymässä oleva nappi liitetään lähdekoodiin muuttujaksi, jonka tyyppi on UIButton.

Yhden näkymän sisällä olevien elementtien välille lisätään erilaisia rajoitteita ja suhteita, jotta ne pysyvät erilaisilla näytöillä ja eri tilanteissa halutussa järjestyksessä ja muodossa. Näistä rajoitteista käytetään Interface Builder -työkalun sisällä nimeä Constraint.

Kun näkymään lisätään esimerkiksi tekstikenttä, sille voidaan kertoa rajoitteiden avulla, että sen tulee joka tilanteessa sijaita 10 pikseliä yläreunasta ja sen leveyden tulee vastata koko näkymän leveyttä. Nämä rajoitteet laatimalla voidaan varmistaa, että käytetään sovellusta sitten iPhone- tai iPad-laitteella, pysyy tekstikenttä aina samassa paikassa ja koko ruudun levyisenä. Elementtejä ja rajoitteita yhdistelemällä saadaan luotua helposti hyvinkin monimutkaisia käyttöliittymiä, jotka skaalautuvat ja toimivat täydellisesti kaikenkokoisilla näytöillä ja kaikilla rotaatioilla. Myös rajoitteita on mahdollista liittää outlet-liitosten avulla näkymän taustalla toimivaan luokkaan, jotta niitä voi lukea ja muokata lähdekoodissa.

Eri näkymät yhdistetään toisiinsa siirtymillä, joista käytetään nimeä segue. Näitä siirtymiä voidaan aktivoida esimerkiksi klikkaamalla käyttöliittymään lisättyä nappia, joka on Interface Builder -työkalussa tai lähdekoodissa määritelty klikattaessa aktivoimaan tietty siirtymä. Lisäämällä näkymiä ja siirtymiä Storyboardiin, saadaan lopulta luotua koko sovelluksen käyttöliittymä ja sen virtaus näkymästä toiseen.

Koska dokumentinhallinta on tarkoitus lisätä olemassa olevaan sovellukseen, sen käyttöliittymän toteutus aloitettiin siis lisäämällä olemassa olevaan Storyboardiin uusi näkymä, josta tulee dokumentinhallinnan käyttöliittymän pohja. Tämä uusi näkymä liitettiin segue-siirtymällä olemassa olevaan näkymään, jossa on toteutettuna valikko, josta pääsee käyttämään sovelluksen eri ominaisuuksia. Tähän pohjana toimivaan uuteen näkymään alettiin sitten toteuttamaan itse dokumentinhallinnan käyttöliittymää.

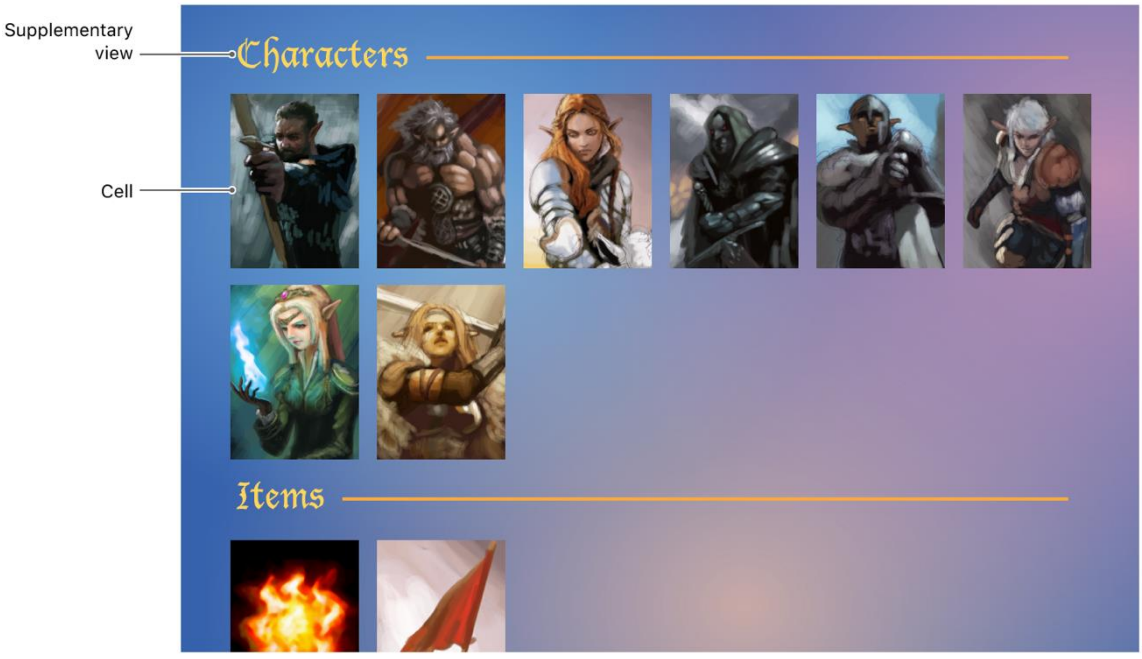
### 6.1.1 Dokumentinhallinnan käyttöliittymä

Dokumentinhallinnan keskiössä ovat tietysti dokumentit ja niitä sisältävät kansiot. Käyttöliittymään vaaditaan siis jokin rakenne, jolla niitä voidaan mahdollisimman tehokkaasti selata. iOS SDK tarjoaa kaksi erilaista käyttöliittymäelementtiä useiden kohteiden tehokkaaseen esittämiseen ja selaamiseen: lista ja kokoelma.

Lista on yksinkertainen elementti, joka sisältää kohteita, joita voi selata vertikaalisesti ja jokainen kohde on yhtä leveä kuin itse lista. Kohteiden korkeus on itse määriteltävissä. Kuva 5 näyttää mille lista näyttää yksinkertaisimmillaan. Listan kohteisiin voi kuitenkin lisätä hyvin vapaasti muita elementtejä kuten kuvia ja nappeja. Listaan voi myös lisätä ylä- ja alatunnisteita sekä eri osioita. Listan toimintaa ja sisältöä voi muokata ja säätää hyvin kattavasti delegaattien avulla, jotka saavat tiedon kaikista listan tapahtumista ja voivat reagoida niihin ja muokata listaa. iOS SDK:ssa listaelementti on nimeltään UITableView.


KUVA 5. Yksinkertainen lista iOS-käyttöjärjestelmässä (Apple Inc. 2018o.)

Kokoelma on hyvin samankaltainen toiminnaltaan listan kanssa ja siitä voi luoda ulkonäöltään lähes samanlaisen, jos niin haluaa. Se on kuitenkin paljon muokattavampi ja mahdollistaa erilaiset asettelut joiden välillä voi vaihtaa, vaikka käytön aikana. Tavanomainen käyttötarkoitus kokoelman käyttöön listan sijasta on tarve taulukkomaiselle rakenteelle, jossa näytöllä on useita kohteita vierekkäin ja allekkain. Kuva 6 antaa esimerkin mille kokoelma voi näyttää. Kuten lista, myös kokoelma on hyvin muokattava niin ulkonäöltään kuin toiminnaltaan ja hyödyntää myös delegaatteja. Myös kokoelmaan saa lisättyä ylä- ja alatunnisteita. Niistä käytetään nimitystä Supplementary View ja ne ovat nähtävissä kuvassa 6. iOS SDK:ssa kokoelma on nimeltään UICollectionView.




KUVA 6. Kokoelma iOS-käyttöjärjestelmässä (Apple Inc. 2018p.)

Mahdollisuus käyttää erilaisia asetteluja ja kyky vaihtaa niiden välillä milloin vain on todella hyödyllinen ja tarjoaa monipuolisia mahdollisuuksia kehittäjille kokoelman hyödyntämiseen. Usein käytetty tapa erilaisten asettelujen käytössä on nipistämällä ja venyttämällä toimiva lähentäminen ja loitontaminen, jotka suurentavat ja pienentävät kokoelmassa olevia kohteita.

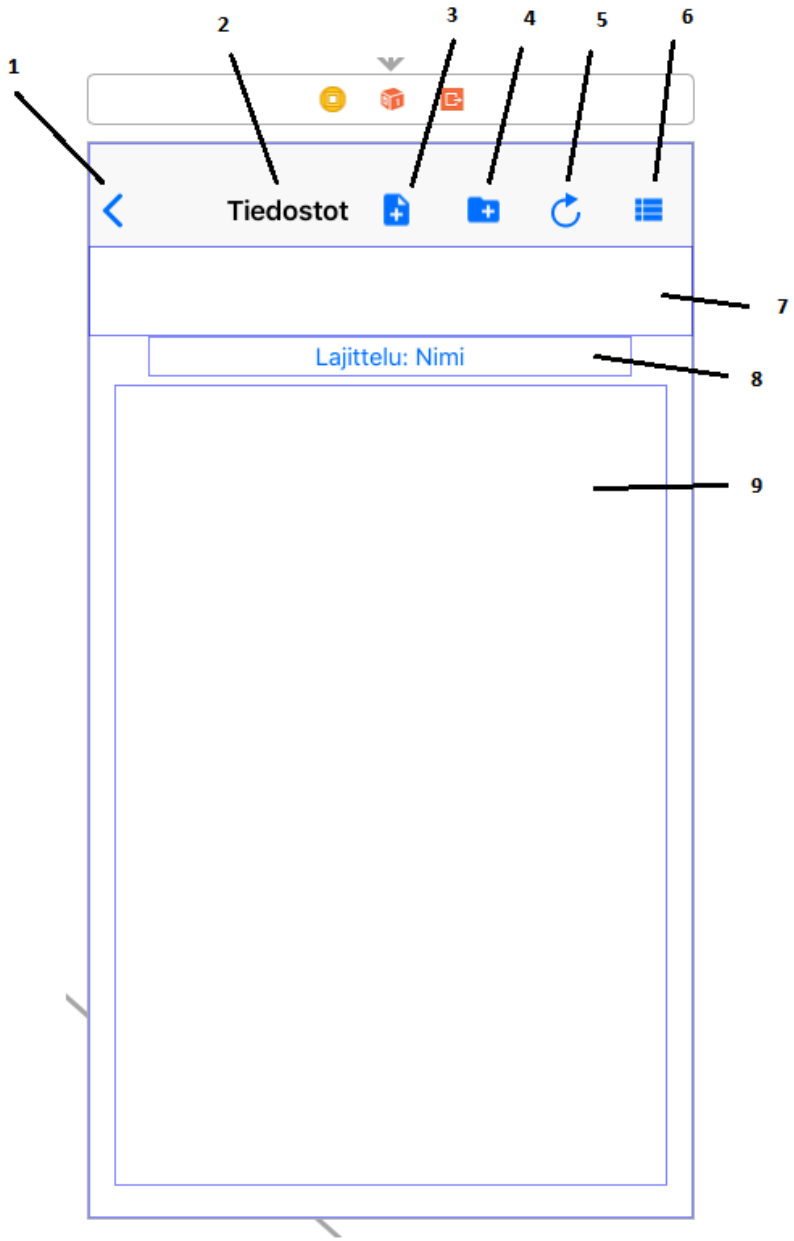
Dokumentinhallintaan valitsin käytettäväksi elementiksi kokoelman, koska se tarjoaa laajemmat muokkausmahdollisuudet ja sen avulla on mahdollista luoda kansioiden selailuun ja tarkasteluun käyttöliittymä, joka muistuttaa hyvin paljon muiden käyttöjärjestelmien resurssienhallinnan ulkonäköä taulukkomaisine rakenteineen. Toinen ja vielä suurempi visuaalisen ilmeen innoittaja ja kokoelman valintaan vaikuttanut lähde oli iOS-käyttöjärjestelmästä löytyvä iCloud Drive -verkkotallennustila ja iOS 11 -versiossa lisätty Tiedostot-sovellus.

Sovellus pyrkii käyttöliittymässään jäljittelemään Tiedostot-sovelluksen ulkonäköä, jolloin käyttäjät tuntevat dokumentinhallinnan käyttöliittymän tutuksi ja osaavat käyttää sitä nopeasti. Sovellukseen lisättiin mahdollisuus vaihtaa kokoelman ulkonäköä taulukon ja listan välillä. Tämä ominaisuus löytyy myös Tiedostot-sovelluksesta. Ominaisuus tarjoaa monia hyötyjä.

Taulukko-asettelussa kansioita ja tiedostoja on nopea selata ja niitä voi esittää enemmän kerrallaan ruudulla varsinkin, jos käyttäjä käyttää sovellusta iPadilla. Kohteista ei kuitenkaan mahdu näyttämään suurta määrää tietoja ja tässä mahdollisuus vaihtaa lista-asetteluun auttaa huomattavasti. Lista-asettelussa kohteista voidaan näyttää enemmän eri tietoja ja pitkätkin tekstit voidaan näyttää koko pituudessaan. Lista-asettelussa kohteisiin on myös helppo lisätä muita elementtejä kuten painikkeen, josta voi avata kohdekohtaisen valikon. Haittapuolena kohteita on tietysti vähemmän ruudulla, koska ne vievät enemmän tilaa. Antamalla mahdollisuuden vaihtaa eri asettelujen välillä saadaan kuitenkin käytännössä parhaat asiat molemmista toteutuksista samaan näkymään.

Kokoelman lisäksi käyttöliittymästä löytyy hakupalkki kokoelman kohteille sekä mahdollisuus muuttaa tapaa, jolla kokoelman kohteet järjestellään. Myös nämä elementit ovat saaneet innoituksensa Tiedostot-sovelluksen toteutuksesta ja niiden olemassaololle on vankat perusteet, kun tavoitteena on luoda helppokäyttöinen dokumentinhallintajärjestelmä. Varsinkin kohteiden haku on tärkeä ominaisuus.

Dokumentinhallinnan näkymästä löytyy myös iOS-sovelluksissa usein käytettävä navigointipalkki, joka sisältää tärkeitä toimintoja. Navigointipalkin kautta päästään myös palaamaan edelliseen näkymään ja suljettua dokumentinhallinta. Navigointipalkissa on painikkeet uuden tiedoston lisäämiselle nykyiseen kansioon, uuden kansion lisäämiselle nykyiseen kansioon, näkymän päivittämiselle ja asettelun vaihdolle taulukon ja listan välillä.




KUVA 7. Valmis käyttöliittymä Storyboardissa.

Kuva 7 esittää mille lopullinen käyttöliittymä näyttää Storyboardissa ja sisältää seuraavat elementit:

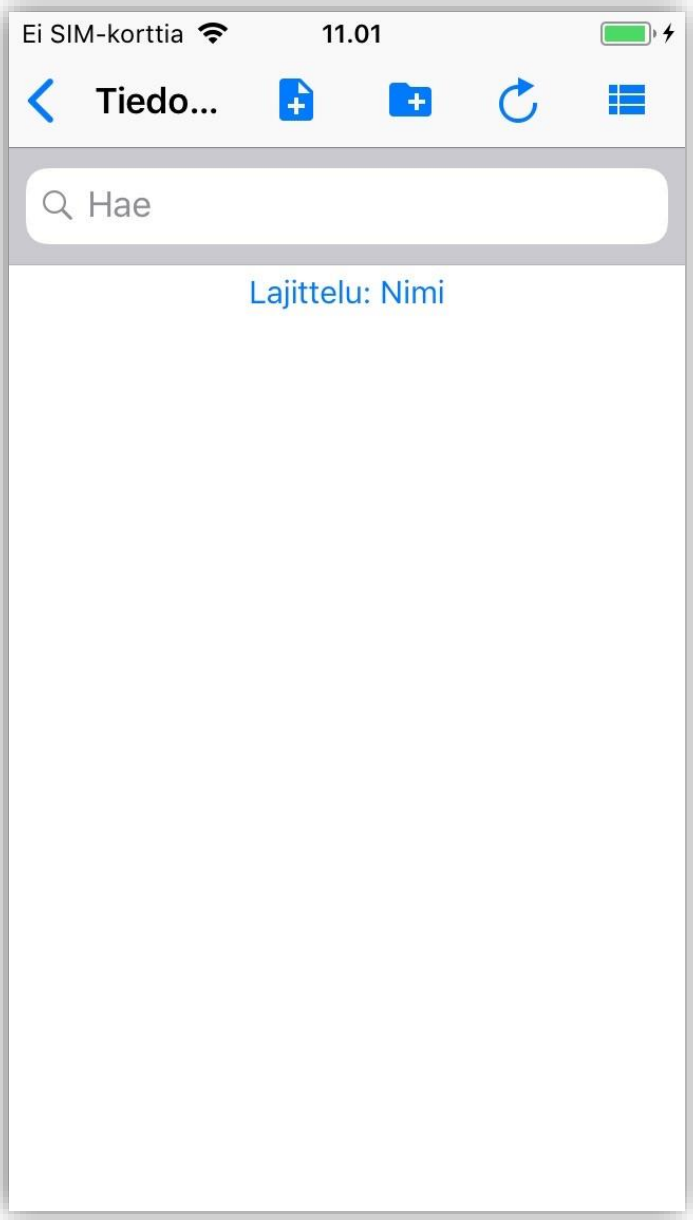
1. Navigointipalkkiin automaattisesti luotu edelliseen näkymään palauttava painike.
2. Navigointipalkin otsikko.
3. Painike uuden tiedoston lisäämiselle nykyiseen näkymään.
4. Painike uuden kansion lisäämiselle nykyiseen näkymään.
5. Painike näkymän päivittämiselle.
6. Painike näkymän asettelun vaihdolle taulukon ja listan välillä.
7. Tyhjä tila, johon lisätään ohjelmallisesti hakupalkki.
8. Painike, jolla voi vaihtaa kohteiden lajittelutapaa.
9. Kokoelma, joka näyttää kaikki kansiot ja tiedostot. Kokoelma voi sisältää Storyboardissa solujen prototyyppejä, mutta tässä tapauksessa solut suunnitellaan muualla ja kokoelma on tyhjä.

```
@IBOutlet weak var TiedostotCollectionView: UICollectionView!
@IBOutlet weak var AddFileButton: UIBarButtonItem!
@IBOutlet weak var AddFolderButton: UIBarButtonItem!
@IBOutlet weak var RefreshButton: UIBarButtonItem!
@IBOutlet weak var ChangeLayoutButton: UIBarButtonItem!
@IBOutlet weak var SearchBarParentView: UIView!
@IBOutlet weak var SearchBarParentViewHeight: NSLayoutConstraint!
@IBOutlet weak var SortingButton: UIButton!
```

KUVA 8. Käyttöliittymän elementit yhdistettynä lähdekoodiin outlet-liitoksilla.

Kuvassa 8 on esitetty muuttujat, jotka syntyivät, kun käyttöliittymän elementit yhdistettiin lähdekoodiin. Muuttujat sijaitsevat UIViewController-luokan, joka on näkymän isäntä, jäseninä. Vain näkymän isäntänä toimivaan luokkaan on mahdollista yhdistää elementtejä outlet-liitoksilla.

Yhdistäminen tehdään yleensä Interface Builder -työkalun kautta vetämällä haluttu elementti lähdekoodiin ctrl-näppäin pohjassa. Tätä helpottaa Xcoden tarjoama mahdollisuus käyttää apulaiseditoria, jonka avulla Xcoden ikkunassa voi pitää kahta eri editori-ikkunaa avoinna vierekkäin. Toisessa voi pitää auki Storyboardia Interface Builderissa ja toisessa itse lähdekoodia, jolloin yhdistäminen on todella vaivatonta.

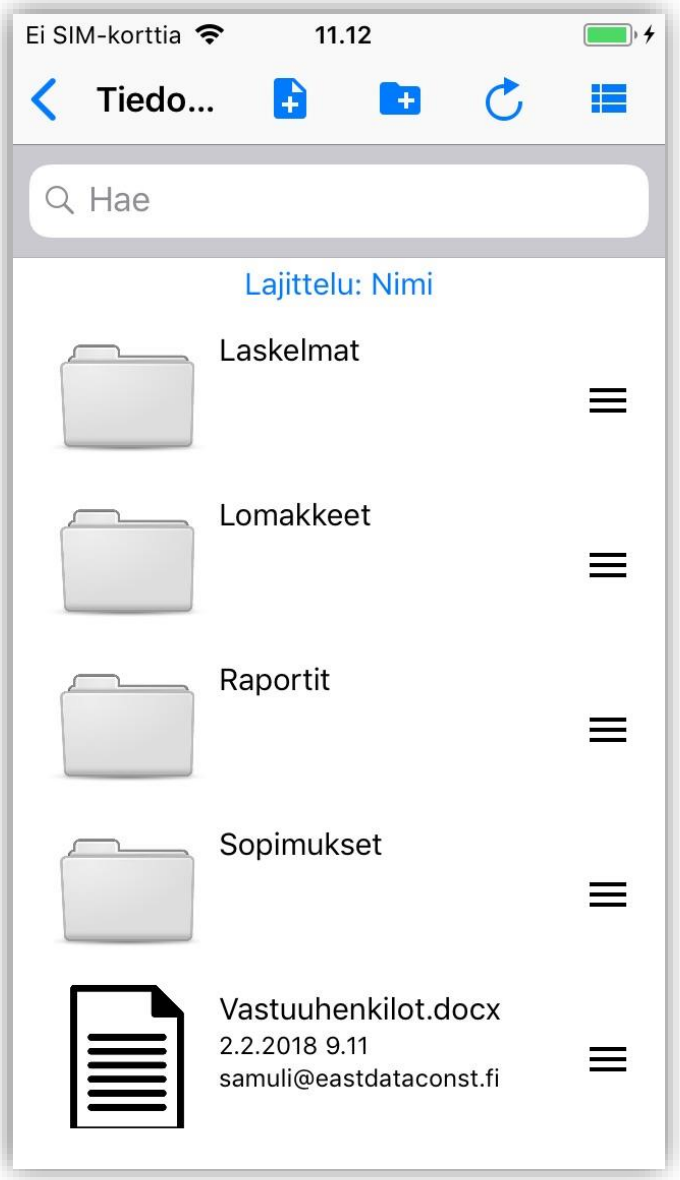



KUVA 9. Valmis käyttöliittymä tyhjällä kokoelmalla iPhone SE -laitteessa (iOS 11)

Kuva 9 näyttää mille valmis käyttöliittymä, johon on ohjelmallisesti lisätty hakupalkki, näyttää itse laitteessa eli tässä tapauksessa iPhone SE -puhelimessa, jossa on iOS 11 -käyttöjärjestelmä. Ruudun pienen koon takia navigaatiopalkin otsikko ei mahdu kokonaisuudessaan ruudulle neljän painikkeen takia. Kuvassa kokoelmaan ei ole vielä lisätty ainuttakaan kohdetta.

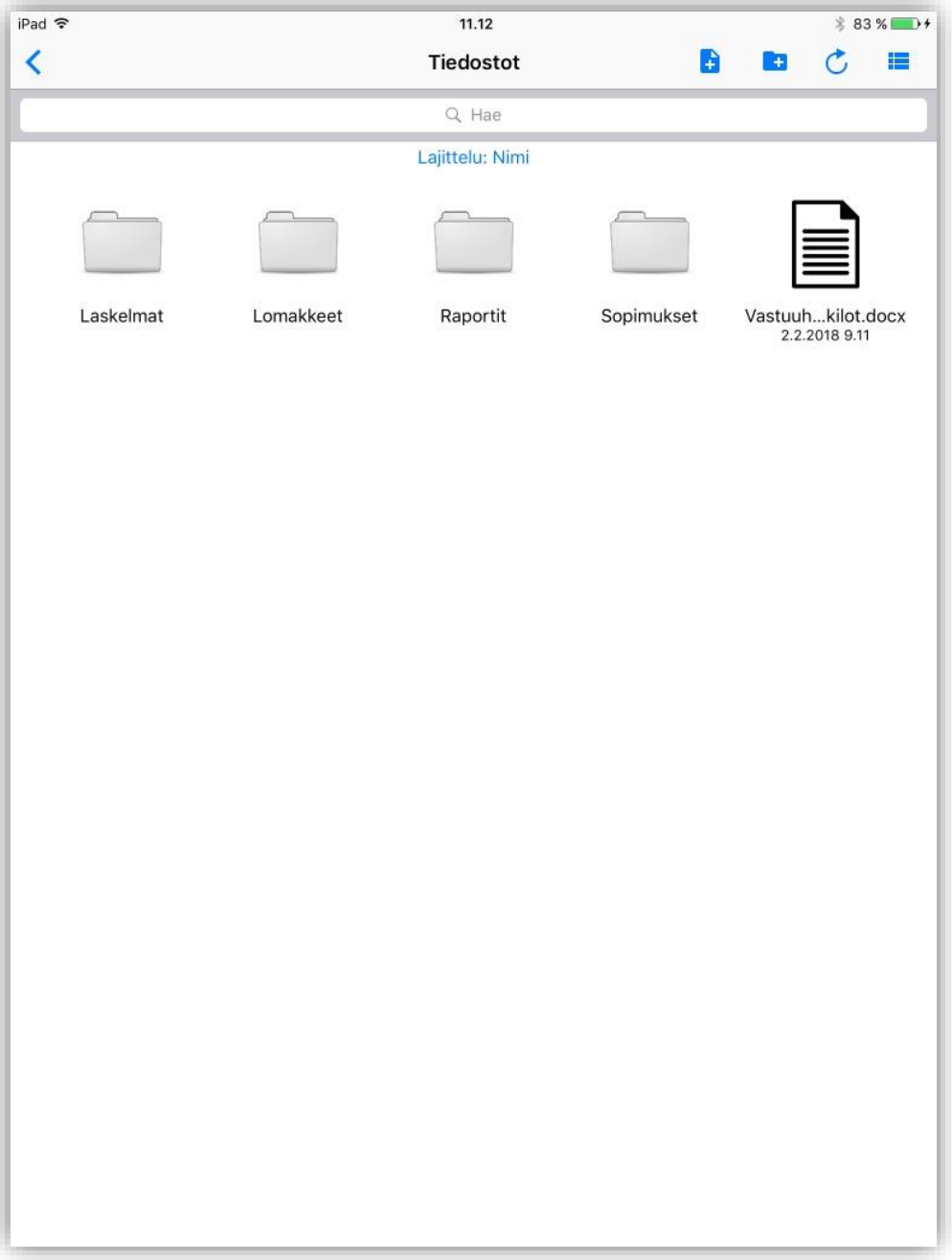

KUVA 10. Valmis käyttöliittymä taulukkoasettelulla iPhone SE -laitteessa (iOS 11)

Kuvassa 10 kokoelmaan on lisätty kohteita ja käytetään vakiona käytettävää taulukko-asettelua. Kohteista näytetään vain nimi. iPhone SE -laitteessa kohteita mahtuu kaksi vierekkäin. Kokoelmaa voi vierittää pystysuunnassa, jotta kaikki kohteet ovat saatavilla.




KUVA 11. Valmis käyttöliittymä lista-asettelulla iPhone SE -laitteessa (iOS 11)

Kuvassa 11 kokoelman asettelu on vaihdettu lista-asetteluun ja kohteista on saatavilla enemmän tietoa. Listassa on myös jokaiselle kohteelle oma painike, josta avautuu kohdekohtainen valikko.




KUVA 12. Valmis käyttöliittymä taulukkoasettelulla iPad mini 2 -laitteessa (iOS 10)

Kuvassa 12 on esitettynä käyttöliittymän ulkonäkö iPad mini 2 -laitteessa. Käytössä on taulukko-asettelu ja iPad-laitteen suurempi näyttö mahdollistaa useamman kohteen näyttämisen vierekkäin ja näytölle mahtuu kokonaisuudessaan kerralla huomattavasti enemmän kohteita kuin iPhone SE -laitteessa. Myös lista-asettelua käytettäessä kohteita mahtuu enemmän suuremman pystyresoluution ansiosta. Molemmissa iOS-versioissa käyttöliittymän ulkonäkö pysyy samanlaisena.

### 6.1.2 Kokoelma, asettelut ja solut

Kokoelman käyttö sekä sen asettelun ja ulkonäön luominen vaatii kaksi eri asiaa: tiedon kokoelman solujen asettelusta sekä käytettävistä soluista. Asettelu siis määrittelee solujen muodon eli miten korkeita ja leveitä ne ovat sekä miten niitä selataan. Käytettävä solu taas määrittelee mitä elementtejä kuten tekstikenttiä ja kuvia solu pitää sisällään. Yhdessä nämä luovat kokoelman halutun muotoisia soluja, jotka sisältävät halutut tiedot.

Kokoelman, jonka pohjalla oli iOS SDK:ssa UICollectionView-luokka, asettelut hoidetaan UICollectionViewFlowLayout-luokan avulla, joka sisältää tarvittavat määrittelyt asettelun aikaansaamiseksi. Tärkeimmät muuttujat, jotka luokka tarjoaa ja joita kokoelma hyödyntää, ovat itemSize-muuttuja, joka kertoo solujen vakiokoon, ja scrollDirection, joka määrittelee mihin suuntaan kokoelmaa voi vierittää. ItemSize sisältää tiedon solun korkeudesta ja leveydestä.

Luomalla uuden luokan, joka periytyy UICollectionViewFlowLayout-luokasta, voi luoda omia muokattuja asetteluluokkia, joilla kokoelman asettelu on helppo hoitaa ja ylläpitää. Kokoelmalle kerrotaan mitä asetteluluokkaa sen tulee käyttää ja päivittämällä näkymä asettelu otetaan käyttöön. Dokumentinhallinnan kokoelman oli tarkoitus tarjota kaksi erilaista asettelua eli taulukko ja lista, joten kokoelmaa varten piti luoda kaksi uutta UICollectionViewFlowLayout-luokasta periytyvää luokkaa, joilla tarvittavat asettelut saadaan hoidettua. Molemmissa asetteluissa vierityssuunta on vertikaalinen.

Taulukkoasettelussa ja sen UICollectionViewFlowLayout-luokassa solujen korkeus on vakioitu ja leveys vaihtelee. Leveydellä on minimiarvo ja aina kun kokoelman asettelu päivitetään, solujen leveys lasketaan uudelleen. Asettelu laskee, kuinka monta kokonaista minimileveyden solua kokoelmaan mahtuu vierekkäin. Kokoelmaan tulee niin monta solua vierekkäin, mutta ylijäänyt tila jaetaan vielä solujen kesken, jolloin niistä tulee hieman leveämpiä. Kuvassa 13 esitellään asetteluluokan sisäinen ohjelmakoodi, jolla solujen leveys lasketaan.

```
///Return the width of the element that is atleast the minimum width
func itemWidth() -> CGFloat {
    let numberOfColumns : Int = Int(collectionView!.frame.width / minimumWidth)
    return ((collectionView!.frame.width) / CGFloat(integerLiteral: numberOfColumns)) - 1
}

override var itemSize: CGSize {
    set {
        self.itemSize = CGSize(width: itemWidth(), height: itemHeight)
    }
    get {
        return CGSize(width: itemWidth(), height: itemHeight)
    }
}
```

KUVA 13. Solujen koon laskeminen taulukkoasettelua varten luodussa luokassa.

Lista-asettelussa ja sen UICollectionViewFlowLayout-luokassa solut ovat aina koko kokoelman levyisiä, jolloin yhdelle riville mahtuu vain yksi solu. Korkeudelle on määritelty minimiarvo, mutta jotta lista-asettelussa voidaan näyttää kaikista kohteista kaikki tekstit täydessä pituudessaan, kuten tarkoitus oli, korkeus lasketaan aina dynaamisesti uudelleen, kun näkymä päivitetään. Jotta listassa ei olisi tyhjää tilaa, korkeus täytyy laskea jokaiselle solulle erikseen. Tätä laskentaa ei voida suorittaa asetteluluokan sisällä, koska se vaatii tietoa, jota asetteluluokalla ei ole, soluissa olevista teksteistä ja niiden pituuksista, eikä asetteluluokka salli muutenkaan koon laskemista jokaiselle solulle erikseen.

Ratkaisu tähän ongelmaan on käyttää apuna delegaattia, joka toimii yhteistyössä asetteluluokan kanssa asettelun luomiseksi ja saa ilmoituksia asettelussa tapahtuvista muutoksista. UICollectionViewDelegateFlowLayout on iOS SDK:ssa oleva protokolla, joka sisältää tarvittavat metodit ja jonka voi toteuttaa jollain luokalla. Sen jälkeen kokoelmalle määritellään, mikä instanssi jostain protokollan toteuttavasta luokasta toimii delegaattina. Protokollasta löytyy metodi, jossa voidaan määritellä koko jokaiselle solulle erikseen (collectionView(_:layout:sizeForItemAt:)).

Tässä tapauksessa delegaatiksi asetettiin koko näkymän taustalla toimiva UIViewController-luokka, joka hallinnoi niin kokoelmaa kuin sen tietosisältöä. Tällä luokalla on siis pääsy solujen sisältöön, joten sen avulla on mahdollista laskea tarvittava korkeus jokaiselle solulle erikseen ja ilmoittaa se asetteluluokalle.

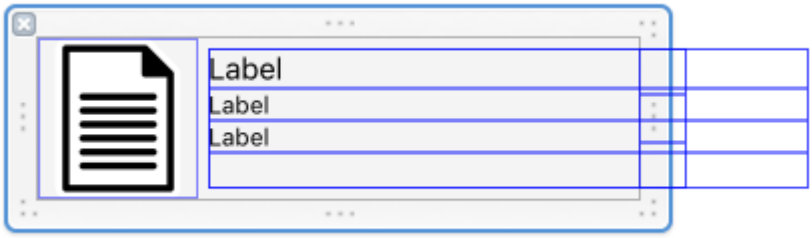
Kokoelma on mahdollista luoda käyttäen useita erilaisia asetteluja ja vain yhdentyyppistä solua kaikkien eri asettelujen kanssa. Tämän sovelluksen tapauksessa, koska tarkoituksena on näyttää erilaisia tietoja kohteista riippuen onko käytettävä asettelu taulukko vai lista, täytyy eri asetteluja varten luoda myös erilaiset solut omilla sisäisillä asetteluillaan.

Solujen pohjana toimii UICollectionViewCell-luokka, josta luodaan yleensä aliluokka periyttämällä. Tässä tapauksessa luotiin kaksi erilaista uutta aliluokkaa, yksi kummallekin eri asettelulle. Solujen ulkonäkö luotiin Nib-tiedostojen avulla, joita luotiin yksi kummallekin solulle.

Nib-tiedostot ovat tiedostoja, jotka sisältävät yhden tietyn näkymän ja sen asettelun. Niitä voi muokata Xcodessa samalla tavalla Interface Builderilla kuin Storyboardissa muokataan useita näkymiä kerralla. Nib-tiedostoihin tallennettuja käyttöliittymän palasia voi lisätä Storyboardissa tai muualla määriteltyihin näkymiin vapaasti, mutta se pitää suorittaa ohjelmallisesti.

Nib-tiedostossa suunniteltu näkymä yhdistetään kokoelman soluun valitsemalla näkymälle taustalla toimiva luokka, josta tulee sen isäntä. Tämä luokka oli tässä tapauksessa samalla tyyppiä UICollectionViewCell. Nib-tiedostoissa suunnitellun solun elementit yhdistettiin solua esittävään luokkaan outlet-liitosten avulla, jotta elementteihin päästään ohjelmallisesti käsiksi. Tietty Nib-tiedosto pitää vielä rekisteröidä tietyn solun kanssa käytettäväksi UIViewController-luokassa, joka hallinnoi kokoelmaa, jotta niiden käyttäminen kokoelman kanssa onnistuu.


KUVA 14. Taulukon solu Nib-tiedostossa Interface Builder -työkalulla katseltuna.




KUVA 15. Listan solu Nib-tiedostossa Interface Builder -työkalulla katseltuna.

Kuva 14 esittää millainen taulukossa käytettävä solu on. Se on tarkoitus pitää mahdollisimman lähellä neliön muotoa ja se sisältää vain kohteen nimen ja mahdollisen lyhyen lisätiedon. Tekstit voidaan tarvittaessa katkaista, jos ne ovat liian pitkiä. Kuva esittää kohteen tyyppiä ja se muutetaan aina kohteen mukaan.

Kuvassa 15 esitellään listassa käytettävän solun ulkomuoto. Kohteesta näytetään useampia tietoja ja kaikki tekstit näytetään myös kokonaisuudessaan. Tässä tapauksessa Interface Builder renderöi solun ulkonäön väärin ja tekstikentät menevät solun ulkopuolelle ja peittävät oikeassa reunassa olevat elementit. Soluun asetetut rajoitteet ovat kuitenkin oikein ja solu toimii laitteessa kuten pitääkin.

Oikeassa reunassa on lisäksi painike, josta aukeaa kohdekohtainen valikko. Kuvassa se on keskimmäinen pieni neliö solun reunalla. Solujen koolla Interface Builderissa ei ole väliä, koska aiemmin esitellyt asetteluluokat ja mahdollinen delegaatti ylikirjoittavat näkymän koon. Suunnilleen oikea koko auttaa kuitenkin visualisoimaan solun ulkonäön suunnitteluvaiheessa.

### 6.1.3 File Providerin käyttöliittymä

File Providerin käyttöliittymän toimittaa järjestelmä ja siihen ei voi itse vaikuttaa. Käyttöliittymä on samanlainen kuin iCloud Drive -sovelluksessa oli iOS 10 -versiossa. iOS 11 -versiossa, jossa esiteltiin uusi Tiedostot-sovellus, johon iCloud Drive yhdistettiin, tämä käyttöliittymä ja sen ulkonäkö siirrettiin kaikkien File Provider -laajennuksien käyttöön ja samalla luovuttiin kehittäjien itse luomista käyttöliittymistä.

Jos kehittää edelleen File Provider -laajennusta, joka on tarkoitettu vain iOS 10 -versioon tai sitä vanhempiin, joutuu itse luomaan käyttöliittymän, mutta tässä tapauksessa sille ei ollut tarvetta, koska laajennus on tarkoitettu vain iOS 11 -version tai sitä uudempien versioiden käyttöön. Valmis käyttöliittymä vähentää kehittäjän työtä, mutta vie samalla myös mahdollisuudet muokkauksiin.

## 6.2 Dokumentinhallinnan toiminnot

Dokumentinhallintajärjestelmä pääsovelluksen sisällä sisältää kaikki muut toiminnot paitsi tiedostojen muokkaamisen muissa sovelluksissa ja muokkauksien päivityksen verkkopalveluun. File Provider -sovelluslaajennus huolehtii kyseisten ominaisuuksien toteutuksesta.

### 6.2.1 Kokoelman selaaminen

Kun dokumentinhallintanäkymään saavutaan ensimmäistä kertaa, kokoelma on tyhjä. Kaikki kokoelmassa näytettävät kohteet sijaitsevat tilaajan verkkopalvelussa, josta ne pitää ladata laitteeseen, jotta ne voidaan näyttää. Dokumentinhallinnassa käsitellään kansioita ja tiedostoja. Ne ladataan erikseen. Kansioita ladattaessa ladataan koko kansiorakenne haettavasta kansiosta alaspäin. Tiedostoista ladataan vain kyseisen kansion tiedostot. Kaikki ladattu tieto tallennetaan ilman verkkoyhteyttä tapahtuvaa käyttöä varten sovelluksen hiekkalaatikkoon talteen. Seuraavilla kerroilla, kun näkymään saavutaan, tiedot latautuvat huomattavasti nopeammin, koska ne ladataan ensin suoraan levyltä ja vasta sitten ne päivitetään palvelimelta.

Kokoelma sijaitsee näkymässä, josta vastaa UIViewController-luokasta periytyvä luokka, joka hallitsee kaikkea näkymään ja sen käyttöliittymään liittyvää. Kokoelman sisällöstä vastaa erillinen luokka, joka hoitaa kaikki sen tietosisältöön liittyvät toimet. Tästä luokasta käytetään tässä nimitystä Data provider. Näin UIViewController-luokka voi keskittyä hoitamaan vain käyttöliittymään liittyvät tehtävät ja kertoa tarvittaessa Data providerille, miten kokoelman tulee reagoida käyttöliittymän tapahtumiin. Kun kokoelman tietosisältö on päivitetty, UIViewController-luokan tarvitsee vain päivittää näkymä, jolloin päivitetyt tiedot luetaan ja käyttöliittymä päivittyy. Tavoitteena on pitää ohjelmakoodi selkeänä ja rajata luokkien vastuita.

Kuvassa 16 UIViewController pyytää Data provider -luokkaa päivittämään nykyisen kansion alikansiot, jonka jälkeen näkymä päivitetään. Jos tapahtui virhe, näytetään virheviesti. Päivityspyyntö voi olla seurausta käyttäjän pyynnöstä päivittää näkymä navigaatiopalkista löytyvän napin avulla tai automaattisesti tapahtuva päivitys, kun levyltä on ensin ladattu ilman verkkoyhteyttä tapahtuvaa käyttöä varten tallennettu sisältö. Myös kansiorakenteessa siirtyminen aiheuttaa tietojen päivitystä.

```
///Get all the subdirectories of the current directory
private func GetDirectories() {
    DataProvider.GetDirectories { (error) in
        self.StopRefresh()
        if error == nil {
            self.TiedostotCollectionView.reloadData()
        } else {
            UIApplication.shared.keyWindow?.makeToast(Strings.Localized.DirUpdateFail)
        }
    }
}
```

KUVA 16. UIViewController-luokka kertoo Data provider luokalle, että nykyisen kansion alikansiot tulee päivittää.

Kun näkymään sitten saavutaan, siitä tulee ilmoitus UIViewController-luokalle ja tällöin kyseinen luokka ilmoittaa Data provider -luokalle, että sen tulee hakea kokoelmassa näytettävä tietosisältö. Kun sisältö on haettu, Data provider ilmoittaa UIViewController-luokalle, että haku on valmis, jolloin UIViewController päivittää näkymän ja haetut kohteet näkyvät kokoelmassa. Kokoelmassa näkyy kyseisen kansion alikansiot ja tiedostot. Tämän jälkeen tiedot vielä mahdollisesti päivitetään palvelimelta. Ensimmäistä kertaa näkymään saavuttaessa kansio, jonka tiedot haetaan, on juurikansio. Haku voi tapahtua vaiheittain ja joka vaiheen jälkeen UIViewController-luokka päivittää näkymän näyttämään siinä sillä hetkellä olevat kohteet.

UIViewController-luokka on asetettu kokoelman delegaatiksi ja saa ilmoituksia kokoelman tapahtumista. Yksi näistä tapahtumista on, kun käyttäjä painaa jotain kokoelman kohdetta. Jos kohde on jokin kokoelmassa näkyvä kansio, ilmoitetaan Data provider -luokalle, että tulee suorittaa siirtyminen kyseiseen kansioon. Data provider suorittaa tarvittavat toimet, joka tarkoittaa valitun alikansion valintaa nykyiseksi kansioksi, edellisen kansion tallentamista hierarkiassa palaamista varten sekä edellisen kansion tietojen poistamista kokoelmasta. Tämän jälkeen kansioon ladataan saatavilla oleva tieto siellä olevista kohteista levyltä. Tämän jälkeen kansiossa olevat kohteet päivitetään palvelimelta ja tallennetaan taas levylle, jos haku onnistuu.

Näkymä päivitetään aina, kun sille on jotain tarvetta, eli kansioon siirryttäessä kokoelman sisältö päivitetään käytännössä heti kun nykyinen kansio on vaihdettu uuteen, seuraavan kerran kun levylle tallennettu sisältö on ladattu kansioon ja viimeiseksi kun kansion tiedot on päivitetty palvelimelta. Tavoitteena on mahdollisimman responsiivinen käyttäjäkokemus, jossa kansion vaihto toimii nopeasti.

Kuvassa 17 on kuvattuna osa UIViewController-luokan delegaattimetodista, joka vastaanottaa tiedon, kun jokin kokoelman kohde valitaan. Metodissa kerrotaan ensin Data provider -luokalle, että tapahtui valinta, jolloin kyseinen luokka voi tehdä omat sisäiset toimensa liittyen valittuun kohteeseen. Jos kyseessä oli kansiorakenteessa siirtyminen, tämän jälkeen näkymä voidaan päivittää, jonka jälkeen itse tiedot voidaan päivittää.

```
func collectionView(_ collectionView: UICollectionView, didSelectItemAt indexPath: IndexPath) {
    self.StopRefresh()
    //Valitaan kohde
    let itemType = DataProvider.SelectItemAt(Index: indexPath.row)
    //Tehdään toiminto valitun kohteen tyypin mukaan
    switch itemType {
    case .BackButton, .Directory:
        //Siirryttiin kansiorakenteessa
        //Päivitetään näkymä ja haetaan tiedot uudestaan
        TiedostotCollectionView.reloadData()
        backgroundUtils.startActivityIndicator()
        DataProvider.GetOfflineFilesForCurrentDirectory {
            self.TiedostotCollectionView.reloadData()
            self.GetDirectories()
            self.GetFiles()
        }
```

KUVA 17. Toiminnot, jotka suoritetaan, kun käyttäjä valitsee kansion tai palaa edelliseen kansioon.


KUVA 18. Käyttäjä on siirtynyt alikansioon lista-asettelussa iPhone SE -laitteessa (iOS 11)

Kuva 18 esittää näkymää sen jälkeen, kun käyttäjä on valinnut jonkin alikansion ja näkymä on päivittynyt vastaamaan alikansion sisältöä. Käytössä on lista-asettelu. Kun siirrytään juurikansiosta johonkin alikansioon ja mahdollisesti vielä niidenkin alikansioihin, ensimmäinen kohde listassa tai taulukossa on aina Takaisin-painike. Sen painaminen siirtää käyttäjän takaisin edelliseen kansioon ja sisäiseltä toiminnaltaan se on käytännössä identtinen tapahtuma alikansioon siirtymisen kanssa. Ainut ero on, että nykyiseksi kansioksi valittava kansio valitaan edellisistä kansioista eikä alikansioista.

### 6.2.2 Tiedostojen avaaminen

Kun käyttäjä valitsee kokoelmasta jonkin tiedoston, Data provider-luokalle lähetetään tieto, että kyseinen tiedosto pitäisi ladata laitteelle. Tiedostoistahan ei ladata itse sisältöä kuin tarvittaessa. Kokoelmassa näkyvät kohteet ovat vain paikallisia esityksiä palvelimella sijaitsevista tiedostoista.

Data provider -luokka kerää tarvittavat tiedot, jotta lataus on mahdollinen ja aloittaa latauksen. Kun lataus alkaa, dokumentinhallinnan näkymän päälle lisätään ikkuna, jossa kerrotaan meneillään olevasta latauksesta ja annetaan mahdollisuus peruuttaa se. Ikkunassa näkyy myös latauksen eteneminen latauspalkin muodossa.

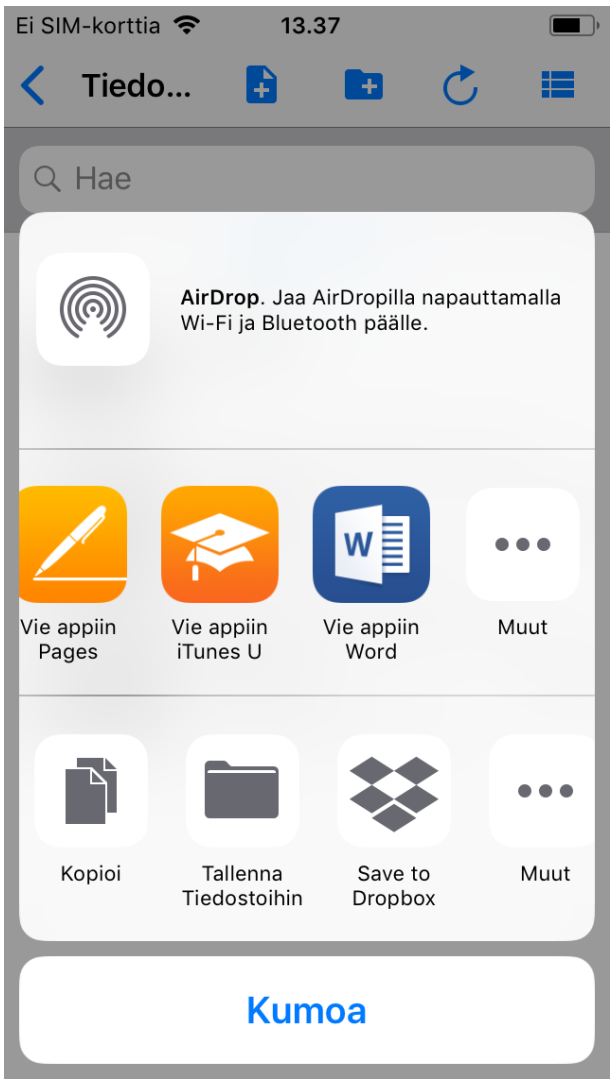
Data provider lataa tiedoston käyttäjäkohtaiseen kansioon sovelluksen hiekkalaatikon sisällä. Kun lataus valmistuu, Data provider jakaa UIViewController-luokalle, joka antoi käskyn ladata kyseinen tiedosto, URL-osoitteen kyseiseen tiedostoon.

Tiedosto on tarkoitus avata toisessa sovelluksessa, jotta sen sisältöä voi tarkastella. Tähän tarkoitukseen käytetään aiemmin esiteltyä UIActivityViewController-luokkaa. kyseinen luokka alustetaan tiedoston URL-osoitteella, jolloin se osaa avata valikon, jossa näytetään kaikki mahdolliset sovellusvaihtoehdot, millä kyseistä tiedostoa voidaan katsella. Kuva 19 näyttää miten kyseinen luokka alustetaan tiedoston URL-osoitteella ja esitetään. Valikon voi myös sulkea suoraan avaamatta tiedostoa.

Kun tiedosto avataan tämän luokan avulla, se on lukittu niin, että vain lukeminen onnistuu. Tiedostoja ei voi muokata pääsovelluksen kautta avaamalla. Sitä tarkoitusta varten tulee käyttää File Provider -laajennusta, joka avataan suoraan toisesta sovelluksesta, jolla tiedostoa halutaan muokata. Kuvassa 20 on esitetty UIActivityViewControllerin ulkonäkö, kun se on avattu dokumentinhallinnan käyttöliittymän päälle alustettuna ladatun tiedoston URL-osoitteella. Tiedosto on Wordin käyttämä .docx-tiedosto, joten, koska laitteeseen on asennettu Word, valikon kautta tiedoston voi avata Wordiin katselua varten.

```
self.FileMenu = UIActivityViewController(activityItems: [fileURL], applicationActivities: nil)
self.FileMenu?.popoverPresentationController?.sourceView = view
self.FileMenu?.popoverPresentationController?.sourceRect = view.bounds
self.present(self.FileMenu!, animated: true, completion: nil)
```

KUVA 19. UIActivityViewController-luokan alustus ladatun tiedoston URL-osoitteella.




KUVA 20. UIActivityViewController avattuna dokumentinhallinnan käyttöliittymän päälle.

Tiedostot ladataan aina uudestaan palvelimelta, kun ne valitaan listalta, eikä niitä tallenneta ilman verkkoyhteyttä tapahtuvaa käyttöä varten. Tiedostot, jotka on ladattu sovelluksen hiekkalaatikkoon avaamista varten, poistetaan heti, kun se on mahdollista. Näin varmistetaan, että sovellus vie mahdollisimman vähän tallennustilaa käyttäjän laitteelta.

### 6.2.3 Haku ja järjestely

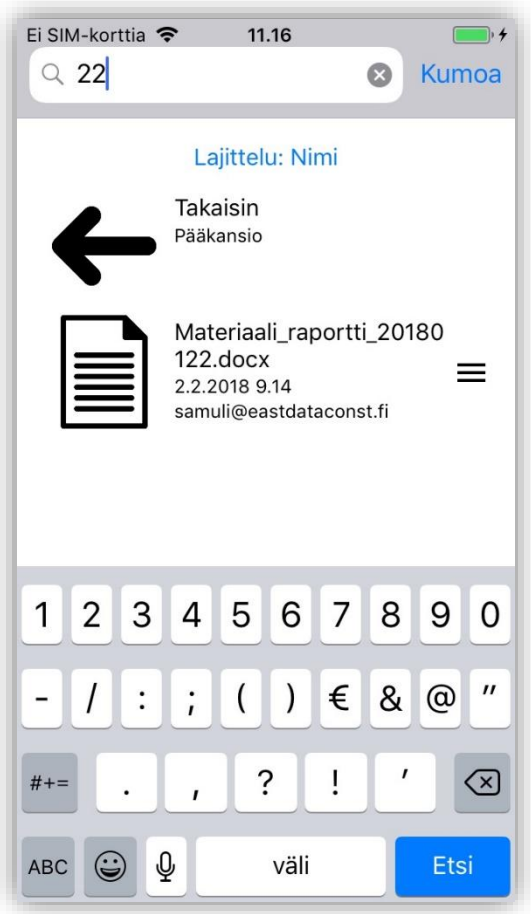
Kansioita ja tiedostoja voi hakea ohjelmallisesti lisätyn hakupalkin avulla. Haku aktivoituu valitsemalla hakupalkki, jolloin se nousee navigointipalkin paikalle ja laitteen näppäimistö nousee esiin. Kuva 21 esittää tätä tilannetta.

Kun haku ja näppäimistö ovat aktiivisena, kokoelman taustalla olevan ScrollView-elementin, joka mahdollistaa kokoelman selaamisen, toimintaa muokataan. Koska näppäimistö vie suuren osan näyttöpinta-alasta, ScrollView-elementtiä muokataan niin, että kyseisen elementin alareuna sijaitseekin näppäimistön yläreunassa näytön alareunan sijasta. Tämän muokkauksen ansiosta kaikki kokoelman kohteet ovat edelleen selattavissa ja mikään kohde ei jää näppäimistön taakse.

Tähän muokkaukseen käytetään apuna ohjelmallisia kuuntelijoita, jotka saavat ilmoituksen, kun näppäimistö tulee esiin tai piilotetaan. Sillä hetkellä ScrollView-elementin rajoituksia muokataan ottamaan huomioon näppäimistön viemä tila tai palautetaan sen rajoitukset ennalleen, jos näppäimistö piilotettiin.

Haku toimii yksinkertaisesti ja se vain suodattaa kokoelman kohteita niiden nimen perusteella verraten niitä hakuun. Hakusana saa esiintyä missä vain kohtaa nimeä. Kuvassa 22 hakuun asetetaan merkkijono ”22” ja kokoelmassa näytetään sen jälkeen vain kohteet, joiden nimestä kyseinen merkkijono löytyy. Takaisin-painike näytetään silti aina. Haun tyhjentäminen tai kumoaminen palauttaa kaikki kohteet näkyviin.


KUVA 21. Aktiivinen hakupalkki iPhone SE -laitteessa (iOS 11)



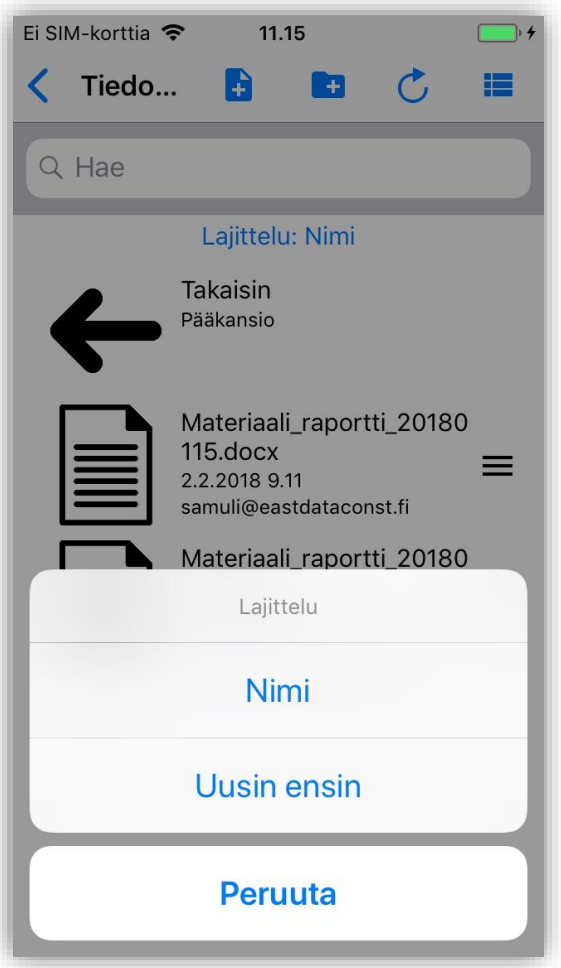

KUVA 22. Haku on suodattanut näytettäviä kohteita iPhone SE -laitteessa (iOS 11)

Hakua varten näkymää hallinnoiva UIViewController-luokka on asetettu hakupalkin delegaatiksi, jolloin se saa ilmoituksia hakupalkin tapahtumista. Kuvassa 23 esitetty metodi ottaa vastaan hakupalkin haun muutokset ja sen sisällä kokoelmaa voidaan muokata vastaamaan hakua. Tässä tapauksessa UIViewController-luokka delegoi työn Data provider -luokalle, koska kyseisen luokan vastuulla on huolehtia kokoelman sisällöstä. Kun kokoelman sisältö on päivitetty, myös näkymä päivitetään.

```
func updateSearchResults(for searchController: UISearchController) {
    let search : String? = searchController.searchBar.text
    DataProvider.Filter(WithSearch: search)
    TiedostotCollectionView.reloadData()
}
```

KUVA 23. Delegaattimetodi, jolla hakupalkin tekstin muutokset saadaan luettua ja kokoelmaa muokattua.

Lajittelu koskee vain tiedostoja. Mahdolliset lajittelutavat ovat nimen mukaan nousevaan järjestykseen tai päivämäärän mukaan laskevasti eli näytetään uusin ensin. Kansiot ovat vakiona nousevassa aakkosjärjestyksessä ja sitä ei voi muuttaa. Nimen mukaan lajittelu on vakioarvo tiedostojen lajittelullekin. Lajittelua voi vaihtaa hakupalkin alla olevasta painikkeesta, josta aukeaa valikko, josta lajittelua voi vaihtaa. Valikko ja sen ulkonäkö on esitelty kuvassa 24.




KUVA 24. Lajitteluvalikko iPhone SE -laitteessa (iOS 11)

Lajittelutavan valinnan jälkeen tiedostot järjestellään valitun tavan mukaisesti ja näkymä päivittyy näyttämään uuden järjestyksen. Päivämäärän mukaan lajittelussa käytetään lista-asettelussa esitettyä päivämäärää.

### 6.2.4 Kansioiden lisääminen

Kansioita lisätään navigointipalkista löytyvällä painikkeella. Kansioita voi lisätä yksi kerrallaan ja ne lisätään aina nykyiseen kansioon. Käyttäjän tarvitsee antaa vain kansiolle nimi. Kun painiketta painaa, näkymään aukeaa kuvassa 25 esitetty uusi ikkuna, jossa käyttäjä antaa nimen uudelle kansiolle. Kun nimi on annettu, uusi kansio lisätään nykyiseen kansioon. Käyttäjä voi myös perua kansion lisäämisen.

Kansio lisätään ennakoivasti välittömästi kokoelmaan. Kansion lisäämiseen sisältyy kuitenkin verkkokutsu, joka lähettää tiedot uudesta kansiosta palvelimelle. Jos tietojen lähetys palvelimelle epäonnistuu tai kansion lisäämistä ei hyväksytä jostain muusta syystä kansio poistetaan kokoelmasta, kun palvelimelta saadaan vastausviesti. Tästä syystä kansion lisääminen ei onnistu ilman verkkoyhteyttä.

Kansio lisätään ennakoivasti kokoelmaan, koska todennäköisimmin käyttäjällä on verkkoyhteys ja lisäys onnistuu. Tämän ansiosta lisäys tuntuu välittömältä ja sovelluksen käyttö sujuvalta. Jos lisäys ei onnistu, käyttäjälle annetaan ilmoitus siitä samalla, kun ennakoivasti lisätty kansio poistetaan.


KUVA 25. Uuden kansion lisääminen nykyiseen kansioon iPhone SE -laitteessa (iOS 11)

### 6.2.5 Tiedostojen lisääminen

Tiedostojen lisääminen aloitetaan samalla lailla kuin kansioiden lisääminen eli painamalla painiketta, joka löytyy sovelluksen navigointipalkista. Tiedostojen lisäämiseen käytetään UIDocumentPicker-ViewController-luokkaa. Näkymää hallinnoiva UIViewController-luokka asetetaan UIDocumentPicker-ViewController-luokan delegaatiksi, jolloin se saa tiedon valitusta dokumentista.

UIDocumentPickerViewController on iOS SDK:sta löytyvä luokka, joka tarjoaa vakioidun käyttöliittymän ja toimintatavan sovelluksen ulkopuolella sijaitsevien dokumenttien ja sijaintien käsittelyyn. Sen käyttäminen vaatii iCloud-tuen lisäämistä sovellukseen. Se tarjoaa neljä erilaista tapaa käsitellä dokumentteja: ulkoisen dokumentin tuonti sovellukseen, paikallisen dokumentin vienti sovelluksesta, ulkoisen dokumentin avaaminen sovelluksessa ja paikallisen dokumentin siirto ulkoiseen sijaintiin. (Apple Inc. 2018q.) Koska tässä tapauksessa tarkoituksena on lisätä uusi dokumentti sovellukseen, käytetään käsittelytapana ulkoisen dokumentin tuomista sovellukseen.

Painikkeesta painamalla avautuu instanssi UIDocumentPickerViewController-luokasta ja sen käyttöliittymästä, joka antaa käyttäjälle mahdollisuuden selata omaa iCloud Drive -tallennustilaa ja muita tarjolla olevia File Provider -tarjoajia. Kun käyttäjä valitsee jonkin tiedoston, UIDocumentPickerViewController suljetaan ja sovelluksella on nyt pääsy uuteen dokumenttiin.

Valittu dokumentti ja sen metatiedot valmistellaan palvelimelle lähetystä varten ja laitetaan lähetysjonoon taustalähetystä varten. Lähetykseen käytetään sovelluksesta löytyvää aiemmin luotua luokkaa, joka hoitaa tiedostojen lähetyksen palvelimelle. Tämä tapahtuu jonossa lisäysjärjestyksessä ja lähetykset toimivat myös taustalla, kun sovellus ei ole aktiivinen. Käyttäjä saa ilmoituksen onnistuiko vai epäonnistuiko tiedoston lisääminen lähetysjonoon, mutta muuten koko prosessi tapahtuu käyttäjältä piilossa.

Käyttäjä ei tiedä milloin lähetys valmistuu, koska jonossa voi olla juuri lisättyä tiedostoa ennen muitakin tietoja odottamassa lähetystä. Tiedoston lähetys kuitenkin valmistuu joskus ja sen jälkeen tiedosto ilmestyy siihen kansioon, jossa käyttäjä oli, kun hän lisäsi tiedoston. Käyttäjän pitää manuaalisesti päivittää näkymä, jos hän jäi odottamaan lähetyksen valmistumista. Jos käyttäjä poistui dokumentinhallinnasta tiedoston lisäämisen jälkeen, näkymä päivittyy automaattisesti, kun käyttäjä navigoi siihen seuraavan kerran verkkoyhteyden kanssa.

### 6.2.6 Kohteiden poistaminen

Kohteiden poistaminen onnistuu vain, kun kokoelma on lista-asettelussa. Silloin kaikille kohteille on näkyvissä painike, josta aukeaa kohdekohtainen valikko. Tästä valikosta voi poistaa kohteen. Poistaminen on tällä hetkellä ainut toiminto valikossa.

Kuvassa 26 kyseinen valikko on näkyvillä. Valikko aukeaa kuvassakin taustalla näkyvistä kohteiden oikeassa laidasta olevista painikkeista, jotka käyttävät kolmea viivaa kuvaamaan, että painikkeesta aukeaa valikko. Kun valikosta valitsee ”Poista”, aukeaa vielä uusi ikkuna, jossa varmistetaan, että käyttäjä haluaa poistaa kohteen. Käyttäjällä on siis kaksi mahdollisuutta perua poistaminen.

Poistaminen tapahtuu samalla tavalla kuin uuden kansion lisääminen, eli kohde poistetaan ennakoivasti välittömästi valinnan jälkeen ja sen jälkeen lähetetään tieto palvelimelle, että kohde halutaan poistaa. Jos palvelin vastaa, että poisto onnistui, sen jälkeen ei tapahdu enää mitään. Jos poisto epäonnistuu, kohde palautetaan takaisin listaan.

Poiston onnistumisen jälkeen sovellus poistaa vielä kohteen tiedot levyltä, jotka tallennettiin ilman verkkoyhteyttä tapahtuvaa käyttöä varten. Jos poistettu kohde oli tiedosto, päivitetään kansion, jossa tiedosto oli, tiedot kansion tiedostoista. Jos poistettu kohde oli kansio, poistetaan kansion ja kaikkien sen alikansioiden tiedot levyltä. Näin levyllä pidetään vain ajantasainen informaatio ja tallennustilan käyttö minimissä.


KUVA 26. Kohdekohtainen valikko kohteen poistamiseen iPhone SE -laitteella (iOS 11)

## 6.3 File Provider -laajennus

Ennen kuin sovelluslaajennusta voi työstää, sellainen täytyy lisätä sovellukseen. Uusi sovelluslaajennus lisätään pääsovellukseen lisäämällä siihen uusi kohde (target). Helpoiten uuden sovelluslaajennuskohteen lisääminen onnistuu Xcoden avulla käyttämällä Xcoden mallipohjaa. Mallipohjia löytyy kaikille tarjolla oleville laajennuksille ja nyt valittiin tietysti File Provider -laajennus. Uusi mallipohja lisätään Xcoden valikoista. Laajennukselle annettiin nimi ja valittiin muita tietoja, joita Xcode vaati.

Xcode luo File Provider -laajennukselle valmiin pohjan, jota voi lähteä sitten muokkaamaan, jotta siihen saadaan luotua varsinainen toiminnallisuus. Xcoden luoma mallipohja luo kuitenkin kaikki tarpeelliset tiedostot ja asetukset, jotta sovelluslaajennus toimii ja sovelluksen saa käännettyä ja ajettua laitteessa. Mitään oikeaa toiminnallisuutta se ei kuitenkaan tarjoa.

Mallipohja sisältää mallit kaikista luokista, joita File Provider vaatii toimiakseen. Tämä sisältää tärkeimmän pääluokan eli NSFileProviderExtension-luokasta periytyvän luokan, joka hoitaa kaiken File Providerin keskeisen toiminnallisuuden kuten tiedostojen lataamisen ja enumeraattorien luonnin. Tämän lisäksi Xcode luo valmiin mallin NSFileProviderEnumerator-luokasta periytyvästä enumeraattorista, jonka vastuulla on tarjota File Provider -laajennuksessa näytettävät kohteet. Viimeinen valmiiksi luotu malli on NSFileProviderItem-luokasta periytyvä luokka, joka kuvaa kohdetta, joka näytetään File Provider -laajennuksessa ja joita enumeraattori tarjoaa.

Näihin valmiiksi luotuihin esimerkkiluokkiin on lisätty runsaasti kommentteja mitä missäkin vaiheessa on tarkoitus tehdä. Nämä malliluokat, kommentit ja muu dokumentaatio tarjoavat hyvän aloituksen File Provider -laajennuksen luontiin, mutta se vaatii silti paljon työtä ja dokumentaatio ei ole täydellistä.

Xcoden luoma pohja File Provider -laajennukselle noudattaa iOS 11 -versiossa toimivaa mallia ja Xcoden kautta ei ole enää mahdollista luoda mallipohjaa iOS 10 -versiossa toimivalle File Provider -laajennukselle. iOS 10 -yhteensopivuuden luominen vaatii runsaasti ylimääräistä työtä ja siitä päätettiin luopua.

File Provider ja sitä kautta myös tiedostojen muokkaaminen toimii siis vain iOS 11 -versiossa tai sitä uudemmissa. iOS 10 -yhteensopivuus olisi ollut hyvä lisä, mutta ylimääräistä työtä ja säätöä ei katsottu vaivan arvoiseksi, koska kaikki muut ominaisuudet, jotka pääsovellus tarjoaa, kuitenkin toimivat iOS 10 -versiossa.

### 6.3.1 Kokoelman selaaminen

File Provider -laajennuksessa, kuten pääsovelluksessakin, kokoelmaa selataan kansioittain. Jokaisella kansiolla on oltava oma tunniste ja jokaiselle kansiolle täytyy luoda oma NSFileProviderEnumerator-luokasta periytyvä enumeraattori, joka hoitaa kansion sisällön lataamisen.

Koska sovelluksessa käsitellään tietoja työmaakohtaisesti ja jokaisella käyttäjällä voi olla useita työmaita, täytyy File Provider -laajennuksessakin valita työmaa, jonka dokumentteja käyttäjä haluaa tarkastella. Pääsovelluksessa käyttäjä valitsee ensin työmaan listasta, jonka jälkeen dokumentinhallinta avataan työmaan valinnan jälkeen avautuvasta valikosta. Laajennuksessa työmaat näytetään laajennuksen juuressa olevina kansioina, joita avaamalla avautuu työmaan tiedostorakenteen juurikansio, josta voi navigoida kansiorakennetta syvemmälle.

Kun File Provider -laajennus avataan ensimmäistä kertaa, järjestelmä pyytää laajennukseen luotua NSFileProviderExtension-luokasta periytyvää luokkaa, joka toimii laajennuksen runkona, toimittamaan järjestelmälle enumeraattorin, jonka tunnisteena toimii RootContainer. Tämä RootContainer on järjestelmän toimittama vakiotunniste, josta selviää, että nyt ollaan aivan laajennuksen juuressa. Koska nyt ollaan aivan laajennuksen juuressa, järjestelmälle toimitetaan enumeraattori, joka lataa näkymään työmaita esittävät kansiot, jotta käyttäjä voi valita oikean työmaan.

Työmaiden tiedot ladataan palvelimelta enumeraattorin sisällä, mutta enumeraattori ei voi palauttaa kaikkien työmaiden tietoja kerralla esitystä varten. Kohteiden lataamisessa ja palauttamisessa täytyy käyttää sivutusta, jossa palautetaan vain pieni osa kohteista kerrallaan esitystä varten. Syy tähän on muistinkäytössä.

Sovelluslaajennukset saavat iOS-järjestelmässä huomattavasti vähemmän muistia käyttöönsä kuin oikeat sovellukset, jotta oikeilla ruudulla näkyvillä sovelluksilla olisi mahdollisimman paljon muistia

omaan käyttöönsä jolloin niiden käyttö on mahdollisimman sulavaa. Tämä pakottaa käyttämään kohteiden lataamisessa sivutusta, koska jos sovelluslaajennus palauttaisi kerralla vaikka 100 työmaita kuvastaa kohdetta, sovelluslaajennus kaatuisi muistin loppumisen takia ja sen käyttö olisi mahdotonta.

Kun järjestelmä pyytää ensimmäistä kertaa kohteita esitettäväksi enumeraattorilta, se lisää pyyntöön mukaan, että pyyntö on ensimmäinen sivu. Tämän jälkeen enumeraattori pitää itse yllä omaa sivutusdataa, jonka se lähettää aina järjestelmälle samalla, kun se palauttaa seuraavan sivun kohteita. Jos enumeraattori ei palauta kohteiden mukana sivutusdataa, se on merkki järjestelmälle, että kaikki sivut ja kohteet on nyt ladattu. Sivutusdata on vapaamuotoinen palanen binääridataa, jota enumeraattori voi hyödyntää miten haluaa.

Työmaita ladattaessa enumeraattori tekee verkkokutsun vain kerran silloin kun järjestelmä kertoo, että kyseessä on ensimmäinen sivu. Kun tiedot työmaista on ladattu, enumeraattori tallentaa tiedot paikallisesti, jotta verkkokutsua ei tarvitse toistaa. Sitten se ottaa sopivan kokoisen määrän työmaita, luo niistä NSFileProviderItem-protokollan toteuttavia kohteita ja palauttaa ne järjestelmälle sivutusdatan kera.

Kun järjestelmä palaa hakemaan loppuja työmaita sivutusdatan perusteella, työmaiden tiedot luetaan suoraan paikallisesti tallennetusta tiedosta ja NSFileProviderItem-protokollan toteuttavia kohteita palautetaan sivutusdatan kanssa, kunnes kaikki työmaat on palautettu. Sivutusdatana käytetään yksinkertaista JSON-dataa, joka sisältää tiedot jo palautetuista työmaista.

Kuva 27 antaa esimerkin ohjelmakoodista, jolla enumeraattori palauttaa esitettävät kohteet ja sivutusdatan järjestelmälle haun jälkeen. Jos sivutusdatan arvo on nil, haku on kokonaisuudessaan valmis. Muuten järjestelmä palaa hakemaan loput kohteet sivutusdatan kanssa kutsumalla samaa hakumetodia uudestaan.

```
observer.didEnumerate(items)
observer.finishEnumerating(upTo: (pageData == nil ? nil : NSFileProviderPage.init(pageData!)))
```

KUVA 27. Lähetetään seuraava sivu kohteita järjestelmälle enumeraattorin sisällä, kunnes sivutusdata ei enää ole.

Nyt File Provider -laajennus esittää kokoelman työmaita kansioina. Kun käyttäjä valitsee jonkin kansion, järjestelmä pyytää NSFileProviderExtension-luokalta uutta enumeraattoria, jonka on tarkoitus ladata ja palauttaa tämän kansion sisältö. Tämän kansion ja enumeraattorin isännän tunnisteena toimii valitun työmaan tunniste.

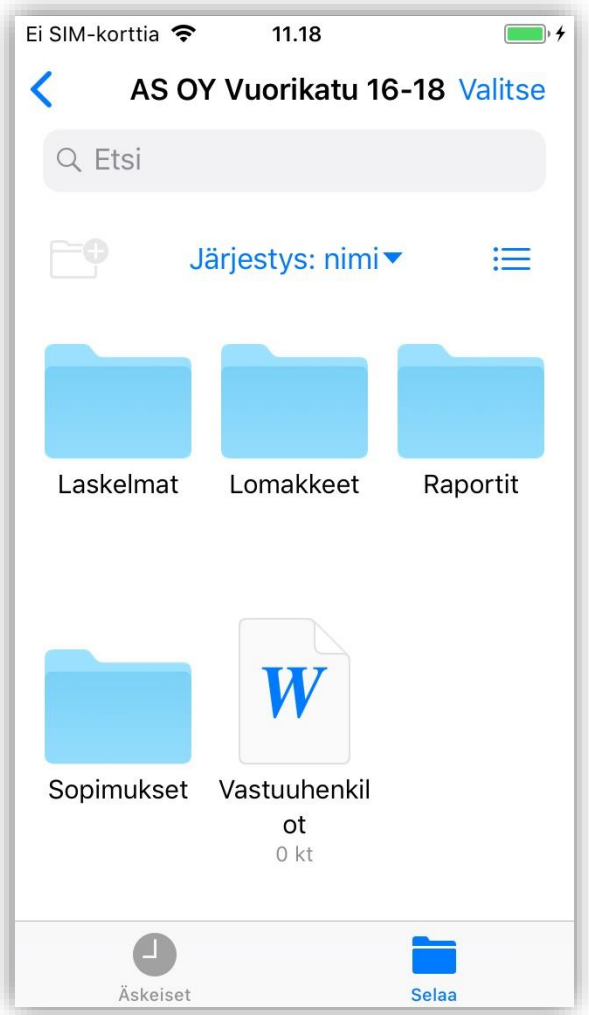
Tämä uusi enumeraattori toimii samalla lailla kuin työmaita ladannut enumeraattori, mutta tällä kertaa ladataan tietyn työmaan kansioita ja tiedostoja esitystä varten. Logiikka on kuitenkin sama ja taas täytyy käyttää sivutusta.

Kansioiden selaaminen siis toimii aina uusia enumeraattoreja pyytämällä ja luomalla kansioiden tunnisteilla ja nämä enumeraattorit huolehtivat sisällön lataamisesta ja palauttamista järjestelmälle. Järjestelmä voi pitää useita enumeraattoreja aktiivisena yhtä aikaa ja varsinkin kun palataan edelliseen kansioon navigaatiopalkin avulla, järjestelmä käyttää usein vanhaa enumeraattoria, jolloin kansion sisältöä ei tarvitse ladata palvelimelta uudestaan.

Kun enumeraattorit lataavat ja palauttavat kohteita, järjestelmä pyytää usein NSFileProviderExtension-luokalta lisätoimia liittyen enumeraattorien palauttamiin kohteisiin. Näihin kuuluu NSFileProviderItem-protokollaa toteuttavien kohteiden palauttamista annetun tunnisteen perusteella ja kansiorakenteen, URL-osoitteiden ja kohteiden paikanpitäjien luomista ja palauttamista. Kaikki File Provider -laajennuksen kohteet sijaitsevat pääsovelluksen kanssa jaetussa tallennustilassa. Pääsovelluksen pitää kuitenkin pyytää tähän tallennustilaan erikseen oikeudet Xcoden kautta, jolloin pääsovellus ja sovelluslaajennus voivat lukea ja kirjoittaa samaan tallennustilaan.

Koska enumeraattorien palauttamia kohteita ja niiden tietoja pitää käsitellä myös NSFileProviderExtension-luokan sisällä, sovellukseen täytyi luoda luokka, jonne tietoja voi tallentaa enumeraattorista ja josta niitä voi lukea NSFileProviderExtension-luokasta. Muuten ei olisi mahdollista luoda NSFileProviderItem-protokollaa toteuttavia kohteita NSFileProviderExtension-luokan sisällä pelkän enumeraattorissa asetetun tunnisteen perusteella.

NSFileProviderItem-protokollan avulla esitetään siis kohteita, joita käytetään File Provider -laajennuksen sisällä. Protokollan tärkeimpiä ja pakollisia tietoja ovat kohteen tunniste, kohteen vanhemman tunniste, kohteen nimi ja kohteen tyyppi. Kohteen vanhempi tarkoittaa kansiota, jossa kohde sijaitsee. Molemmat tunnisteet ovat tärkeitä kohteiden selaamisen takia. Protokollassa on myös monia muita tarpeellisia ja hyödyllisiä tietoja, joita voi ja kannattaa hyödyntää File Provider -laajennuksen toteutuksessa.




KUVA 28. Työmaan tiedostot File Provider -sovelluslaajennuksessa iPhone SE -laitteella (iOS 11)

Kuva 28 esittää näkymää työmaan valinnan jälkeen, kun työmaan tunnisteella luotu enumeraattori on ladannut työmaan tiedostojen ylimmän kansion sisällön. Navigaatiopalkin vasemmassa reunassa sijaitsevasta painikkeesta pääsee palaamaan työmaan valintaan eli laajennuksen juurikansioon. Kansion valinta luo uuden enumeraattorin kyseisen kansion tunnisteella. Koska tässä on hyödynnetty NSFileProviderItem-protokollan tarjoamia mahdollisuuksia, näkyy kansiossa olevassa tiedostosta jo sen kuvakkeesta, että kyseessä on Wordin käyttämä .docx-tiedosto. protokollan kautta voi määritellä kohteen tyypin, jolloin käyttöliittymä osaa näyttää oikeanlaisen kuvakkeen. Tiedoston valitsemalla se avataan sovellukseen, jonka kautta File Provider -laajennus on avattu. Kuvassa 28 laajennusta käytetään Tiedostot-sovelluksen kautta, joka osaa tiedostoja valittaessa näyttää niistä esikatselun uudessa ikkunassa.

### 6.3.2 Tiedostojen avaaminen ja muokkaaminen

Tiedostojen avaaminen tapahtuu NSFileProviderExtension-luokan koordinoimana ja pyyntö avata tietty tiedosto tulee aina kyseiselle luokalle. Usein osana prosessia, jossa järjestelmä pyytää laajennusta avaamaan jonkin tiedoston, se pyytää laajennusta toimittamaan avattavalle tiedostolle paikanpitäjän.

Paikanpitäjä on nimensä mukaisesti korvaava tiedosto varsinaiselle tiedostolle, jota ei ole tallennettu paikallisesti. Järjestelmä pyytää laajennusta luomaan paikanpitäjän tiedostolle sen URL-osoitteen perusteella, jonka laajennus on toimittanut aiemmin järjestelmälle. Paikanpitäjän luomisessa käytetään apuna tiedostoon liittyvää NSFileProviderItem-protokollan toteuttavaa luokkaa. (Apple Inc. 2018r.) Paikanpitäjien luominen on tärkeä osa tiedostojen tarjoamista muille ohjelmille ja ilman sitä tiedostojen avaaminen laajennuksen avulla ei toimi.

Varsinainen tiedoston lataaminen tapahtuu, kun järjestelmä pyytää File Provider -laajennusta aloittamaan tietyssä URL-osoitteessa sijaitsevan tiedoston tarjoamisen. Järjestelmä kutsuu NSFileProviderExtension-luokan startProvidingItem-metodia ja antaa parametrinä tiedoston URL-osoitteen. Tämän metodin sisällä laajennuksen on tarkoitus varmistaa, että kyseisessä URL-osoitteessa on varsinainen tiedosto, joka on mahdollista avata sovellukseen, joka kutsun teki. Tavallisesti tämä tapahtuu lataamalla tiedosto palvelimelta annettuun URL-osoitteeseen. Tiedosto voi olla myös jo valmiiksi saatavilla annetussa URL-osoitteessa, jolloin metodin ei tarvitse tehdä mitään. Tiedoston kanssa voi olla myös ristiriitoja paikallisen ja palvelimella sijaitsevan version välillä, jotka pitää ratkoa tämän metodin sisällä.

Kun laajennus on tehnyt tarvittavat toimet tiedoston olemassaolon varmistamiseksi, se kutsuu tämän metodin toisena parametrina tulevaa completionHandler-metodia. Tämä metodi ottaa vastaan parametrina mahdollisen virheen, joka syntyi tiedoston lataamisen aikana. Jos virhettä ei ole, se merkitsee, että kaikki sujui hyvin ja tiedoston voi avata. Jos tapahtui virhe, tiedostoa ei voi avata ja taustalla oleva sovellus, joka pyysi tiedoston avaamista, voi näyttää virheviestin.

Tässä opinnäytetyössä luotu toteutus lataa tiedoston joka kerta uudestaan palvelimelta ja ei huomioi mahdollisia paikallisia versioita. Niitä ei pitäisi ollakaan, koska ne on tarkoitus poistaa aina, kun tiedoston käyttäminen loppuu. Tämän toimintatavan pitäisi myös estää ristiriitojen syntyminen.

Kun tiedosto sitten avataan johonkin sovellukseen File Provider -laajennuksen avulla, sitä on mahdollista muokata. Nämä muokkaukset pitäisi saada päivitettyä myös palvelimelle. NSFileProviderExtension-luokasta löytyy metodi, jota järjestelmä osaa kutsua automaattisesti aina välillä, kun laajennuksen kautta avattua tiedostoa muokataan. Kun kyseinen metodi aktivoituu, voidaan lukea tiedoston nykyinen sisältö ja lähettää se palvelimelle taustalla. Näin tiedoston sisältö saadaan päivitettyä palvelimelle. Järjestelmä osaa kutsua samaa metodia myös silloin, kun tiedostoa on muokattu ja se suljetaan ennen kuin muutokset on ehditty huomioida aiemmalla metodikutsulla. Näin ollen kaikki File Provider -laajennuksen kautta tehtävät muokkaukset tiedostoihin saadaan helposti kiinni ja lähetettyä palvelimelle. Sovelluslaajennus on toteutettu juuri näin, eli aina kun metodia kutsutaan, luetaan ja lähetetään tiedoston tiedot palvelimelle. Jos palvelimelle ollaan vielä lähettämässä aiempia muutoksia, kyseiset lähetykset keskeytetään, ennen kuin uudemmat aloitetaan. Näin minimoidaan verkossa lähetettävän datan määrä.

Kun viimeinenkin ohjelma tai prosessi lopettaa tietyn laajennuksen tarjoaman tiedoston käytön, NSFileProviderExtension-luokka saa toisen metodin kautta tiedon, että kyseinen tiedosto ei ole enää käytössä. Tämän metodin aikana tiedoston paikallinen kopio voidaan poistaa, koska sitä ei enää tarvita. Tämä säästää tallennustilaa. Metodin aikana voi suorittaa myös muuta ylläpitoa ja siivousta liittyen tiedostoon, joka vapautui.

### 6.3.3 Kirjautumisen tarkistus

Pääsovelluksen käyttäminen on kirjautumisen takana, joten myös laajennuksen tulee vaatia kirjautumista. Pääsovellus ja laajennus voivat hyödyntää samoja kirjautumistietoja. Pääsovelluksen ja laajennuksen ominaisuuksista täytyy vain laittaa päälle yhteisen tallennustilan jakaminen sekä salattujen tietojen jakaminen ja tallentaminen yhteiseen säilöön. Xcode osaa hoitaa automaattisesti oikeiden asetusten asettamisen, jotta ominaisuudet toimivat. Kehittäjän täytyy vain napsauttaa ominaisuudet päälle ja mahdollisesti muokata hieman luokkia, jotka käyttävät jaettavaa dataa.

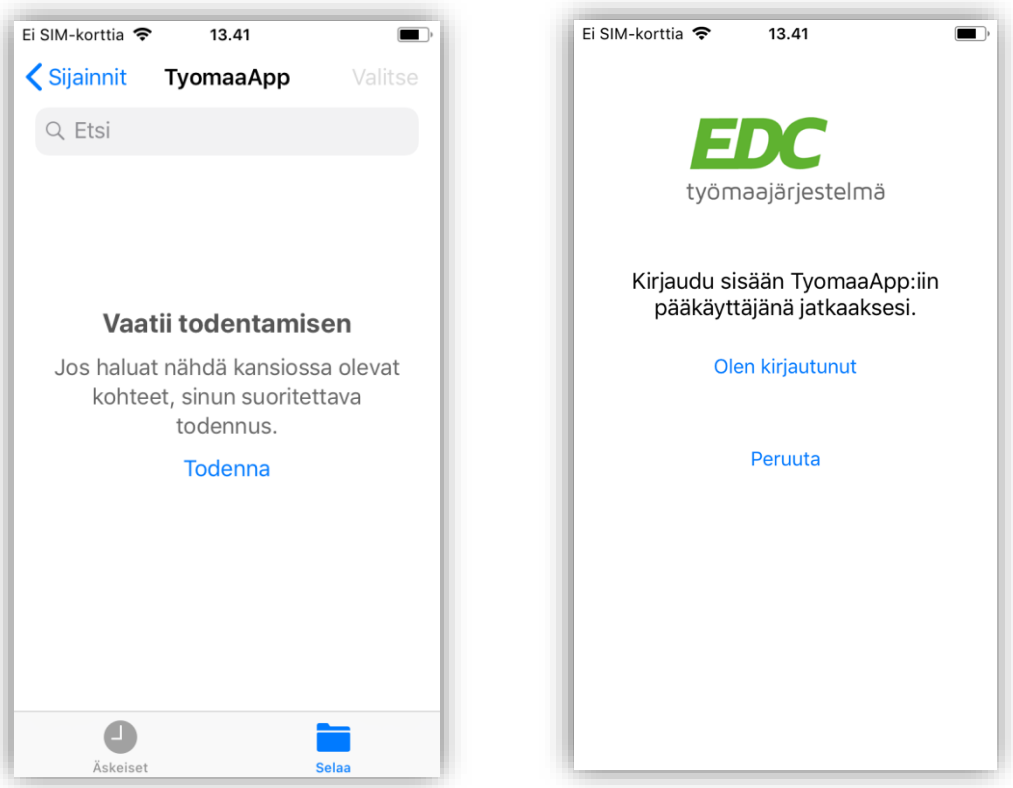
Kirjautumistietojen tarkistus onnistuu niiden jakamisen jälkeen täysin samalla lailla laajennuksessa kuin pääsovelluksessa. Ongelmana onkin miten laajennuksen tulisi toimia, jos käyttäjä ei ole kirjautunut, koska käyttöliittymä ei ole kehittäjän hallinnassa. File Provider -laajennus sisältää kuitenkin valmiin toimintatavan kirjautumisen vaatimiseen.

Sovellukseen pitää ensin lisätä toinen laajennus samalla lailla kuin lisättiin File Provider -laajennus. Lisäämiseen kannattaa taas käyttää Xcoden valmista mallipohjaa, joka tekee paljon työtä valmiiksi. Tällä kertaa sovellukseen lisätään File Provider UI -laajennus. Tämä laajennus on tarkoitettu pääasiassa tarjoamaan kehittäjän luomia lisätoimintoja, kun käyttäjä painaa File Provider -laajennuksessa jotain kohdetta pitkään. Pitkään painaminen avaa kohdekohtaisen valikon lisätoiminnoille kuten uudelleennimeämiselle ja kohteen tietojen tarkastelulle. Jokaiselle kohteelle on aina saatavilla ainakin muutama perustoiminto. File Provider UI -laajennuksella tähän valikkoon voi lisätä omia toimintoja, joita valitsemalla voi tarjota itse luodun käyttöliittymän toiminnon toteuttamiseen.

File Provider UI -laajennus tarjoaa myös mahdollisuuden ottaa vastaan virhe käyttäjän kirjautumistietojen puutteesta. Kun laajennus vastaanottaa kyseisen virheen, se voi avata kehittäjän luoman käyttöliittymän, jossa virhe voidaan käsitellä. Tämä käyttöliittymä voisi esimerkiksi sisältää lomakkeen kirjautumista varten.

Virhe kirjautumisesta tulee heittää File Provider -laajennuksessa NSFileProviderExtension-luokan metodissa, jossa järjestelmä pyytää luokalta uutta enumeraattoria. Virheen vastaanotettuaan järjestelmä näyttää enumeraattorin sisällön sijasta ikkunan, jossa kerrotaan laajennuksen vaativan kirjautumista. Ikkunassa on linkki, joka avaa Provider UI -laajennuksen, jos sellainen on luotu, joka vastaanottaa virheen omaan metodiinsa, jonka sisältä laajennus voi avata kirjautumisvirhettä varten suunnitellun käyttöliittymän.

Tässä tapauksessa File Provider UI -laajennukseen luotu käyttöliittymä kehottaa vain käyttäjää kirjautumaan sisään itse pääsovellukseen, koska kirjautumistiedot on jaettu pääsovelluksen ja laajennuksen kesken. Ikkunassa on linkit peruuttamiselle ja hyväksymiselle. Kun käyttäjä kirjautuu pääsovellukseen ja sen jälkeen hyväksyy laajennuksen kehotuksen kirjautua, järjestelmä pyytää enumeraattoria uudestaan ja, koska käyttäjä on nyt kirjautunut, se onnistuu ja File Provider -laajennuksen sisältö ladataan normaalisti. Kuva 29 ja 30 kuvaavat prosessia. Ensin käyttäjä saa ilmoituksen todennuksen tarpeesta, jonka jälkeen linkistä avautuu kehittäjän suunnittelema käyttöliittymä.




KUVA 29 ja 30. Kirjautumisen tarkistus File Provider -laajennuksessa iPhone SE -laitteella (iOS 11)

### 6.3.4 Muut toiminnot

File Provider -laajennus sisältää tuen monille toiminnoille kuten kansioiden luomiselle ja kohteiden poistamiselle. Lisäksi File Provider UI -laajennuksella voi luoda kohdekohtaisia lisätoimintoja omalla käyttöliittymällään. Jotta File Provider -laajennus voi tukea toimintoja, NSFileProviderExtension-luokan sisälle luodaan jokaista toimintoa varten oma metodi valmiin mallin mukaisesti, jonka toiminnon käyttö aktivoi. Tässä metodissa voidaan sitten tehdä tarvittavat toimet, jotta toiminto onnistuu ja sen muutokset heijastuvat myös palvelimelle. Tämän lisäksi toimintojen kohteiden tulee tukea kyseisiä toimintoja. Kohteiden, jotka kaikki toteuttavat NSFileProviderItem-protokollan, tukemat toiminnot määritellään protokollan capabilities-muuttujalla joka kohteelle erikseen.

Tässä tapauksessa kaikki toiminnot kuten kansioiden lisääminen on jo toteutettu pääsovelluksessa ja käyttäjien halutaan muutenkin käyttävän enemmän pääsovellusta. File Provider -laajennus toteutettiin, koska parempaakaan mahdollisuutta tiedostojen muokkaamiseen ei ollut tarjolla. Näistä syistä

File Provider -laajennus ei tue mitään ylimääräisiä toimintoja. Kansioita on sallittu vain selata ja tiedostoja voi lukemisen lisäksi kirjoittaa. Kohteilla ei ole oikeuksia muihin toimintoihin ja niitä ei ole toteutettu. Laajennus on tarkoitettu vain tiedostojen muokkaamiseen.

Myöskään File Provider UI -laajennuksella ei ole toteutettu mitään ylimääräisiä toimintoja. Jokaisella kohteella on kuitenkin muutenkin muutama perustoiminto, jos niitä painaa pitkään käyttöliittymässä, joiden olemassaoloon ei voi vaikuttaa. Niiden avulla voi lähinnä tarkastella kohteen perustietoja uudessa ikkunassa.

# 7 JATKOKEHITYS

Vaikka sovellukseen saatiinkin toteutettua kaikki vaaditut ominaisuudet eheäksi kokonaisuudeksi, sovelluksessa on kuitenkin paljon mahdollisia kehittämis- ja parannuskohteita. Jotkin eivät ole vielä tämänhetkisellä tekniikalla mahdollisia, mutta jo iOS 11 oli suuri askel eteenpäin tiedostonhallinnan kannalta iOS-järjestelmässä ja voidaan olettaa, että samalla kun ihmiset siirtyvät käyttämään yhä suuremmissa määrin mobiililaitteita, kehityskulku jatkuu ja tulevaisuudessa iOS-käyttöjärjestelmä ja sen SDK tarjoavat entistä enemmän uusia mahdollisuuksia hyödynnettäväksi liittyen dokumenttien ja tiedostojen hallintaan.

Nykyisellä tekniikalla ja saatavilla olevilla työkaluilla sovellusta voisi eniten kehittää File Provider -laajennuksen osalta. Laajennuksen kautta olisi mahdollista tarjota kaikki samat ominaisuudet kuin tällä hetkellä pääsovellus tarjoaa. Tähän kuuluu muun muassa tiedostojen ja kansioiden selaaminen sekä tiedostojen haku. Sovelluksen osalta voisikin harkita, olisiko aiheellista ja mahdollista siirtää koko dokumentinhallinta kokonaan File Provider -laajennuksen kautta toimivaksi. Näin toimiessa dokumentinhallinnan voisi periaatteessa poistaa pääsovelluksesta. Se voisi yksinkertaistaa sovellusta ja mahdollisesti vähentää ylläpitoa, koska File Provider -laajennuksessa järjestelmä huolehtii käyttöliittymästä ja kehittäjän tarvitsee huolehtia vain oikean datan tarjoamisesta.

File Provider -laajennuksen suurempi hyödyntäminen vaatii dokumentinhallinnan osalta kuitenkin iOS 10 -yhteensopivuudesta luopumista, koska laajennus ei tue iOS 11 -versiota aiempia järjestelmäversioita. Tämän takia laajennuksen laajempi käyttö tai jopa pääsovelluksen sisäisestä dokumentinhallinnasta luopuminen ei vielä tällä hetkellä mahdollista. iOS 10 -yhteensopivuutta ei tosin tarkoitus ylläpitää enää kovin pitkään, joten suunnittelutyötä tulevaisuuden kannalta voisi olla jo aiheellista tehdä.

Laajennuksen toiminnallisuutta voisi tietysti laajentaa siitä huolimatta iOS 11 -versiota käyttävien iloksi ja hyödyksi. Tässä tapauksessa samat toimet voisi hoitaa joko pääsovelluksen kautta tai laajennuksen avulla. Tämä lisäisi joustavuutta sovelluksen käyttöön, mutta toisi lisää työtä ylläpitoon ja tietysti itse toteutukseen. File Provider -laajennuksessa on joka tapauksessa osioita ja ominaisuuksia joita voisi vielä parannella ja kehittää. Esimerkiksi hakuominaisuudet eivät tällä hetkellä toimi oikein. Muun kehittämisen osalta tulisi käydä keskustelua sovelluksen suunnasta.

Tiedostojen avaaminen pääsovelluksesta toisiin sovelluksiin ei onnistu tällä hetkellä niin, että tiedostoja voisi myös muokata. Jos tulevaisuudessa iOS-järjestelmään tulisi keinoja toteuttaa tiedostojen muokkaaminen niin, että se toimisi suoraan pääsovelluksen kautta, se olisi erinomainen kehityskohde.

Kyseinen tapa toimia oli alkuperäinen tavoite sovelluksen toteutukseen, mutta iOS-järjestelmästä ei tällä hetkellä löydy keinoja sen toteutukseen. Jos sen saisi toteutettua, File Provider -laajennuksesta voisi luopua ja käyttäjien ei tarvitsisi poistua pääsovelluksesta tiedostojen muokkausta varten ja sovelluksen käyttö olisi hieman yksinkertaisempaa ja selkeämpää. Tämä riippuu kuitenkin täysin iOS-

järjestelmän kehityssuunnasta ja tietysti, jos dokumentinhallinta siirretään kokonaan File Provider -laajennuksen varaan ja poistetaan pääsovelluksesta, tämä ominaisuus on turha.

Pääsovelluksen käyttöliittymää voisi aina hioa ja parantaa. Pääsovelluksen kokoelmaan olisi hyvä lisätä omat kuvakkeet tärkeimmille tiedostomuodoille kuten Word- ja Excel-tiedostoille. Tällä hetkellä kaikilla tiedostoilla on sama kuvake. Tiedostotyypin mukaiset kuvakkeet antaisivat enemmän informaatiota ja auttaisivat löytämään oikean tiedoston nopeammin. Myös haku- ja lajitteluominaisuuksia tulisi parantaa ja antaa lisää vaihtoehtoja lajitteluun sekä mahdollisuus hakea muunkin tiedon kuin vain nimen perusteella.

# 8 YHTEENVETO

Tämän opinnäytetyön tuotoksena toteutettiin dokumentinhallintajärjestelmä iOS-alustalle. Työn tilaajana toimi East Dataconst Oy ja dokumentinhallintajärjestelmä lisättiin osaksi olemassa olevaa iOS-sovellusta. East Dataconst Oy tarjoaa dokumentinhallinnan jo selainpohjaisesti ja Android-laitteille ja yrityksellä oli tarve saada vastaava toiminnallisuus myös iOS-laitteille.

Dokumentinhallinta sisälsi tiedostojen ja kansioiden lisäämisen, selaamisen ja poiston. Lisäksi tiedostoja tuli kyetä avaamaan toisiin sovelluksiin ja muokkaamaan ja muokkaukset lähettämään palvelimelle, jossa kaikki tiedot säilytetään. Sovellukseen lisättiin lisäksi haku- ja lajitteluominaisuuksia.

Dokumentinhallinta jouduttiin toteuttamaan kaksiosaisena, koska iOS-järjestelmä ei tarjonnut keinoa avata tiedostoja suoraan pääsovelluksesta toisiin sovelluksiin niin, että niitä kykenee muokkaamaan. Olemassa olevan pääsovellukseen sisään toteutettiin dokumentinhallintajärjestelmä, joka kykeni kaikkeen muuhun paitsi tiedostojen muokkaamiseen. Muokkaaminen mahdollistettiin käyttämällä File Provider -laajennusta, jota käytetään toisten sovellusten kautta. Se tarjoaa saman sisällön kuin pääsovellus, mutta tiedostoja voi muokata muissa sovelluksissa. Muutokset saadaan myös lähetettyä palvelimelle.

Sovelluksen toteuttaminen onnistui hyvin ja kaikki vaaditut ominaisuudet saatiin toteutettua. Suurimmat haasteet ja ongelmat tarjosi File Provider -laajennus. File Provider -laajennuksen toteutus muuttui iOS 11 -version julkaisussa merkittävästi erilaiseksi kuin aiemmin. Tavoitteena oli tukea niin iOS 10 kuin 11 -versiota. Tästä kuitenkin luovuttiin, koska File Provider -laajennus olisi vaatinut käytännössä kaksi erilaista päällekkäistä toteutusta. Laajennus toteutettiin siis vain iOS 11 -versiolla ja näin ollen tiedostojen muokkaus onnistuu vain iOS 11 -versiolla.

Vaikka iOS 10 -yhteensopivuudesta luovuttiin, File Provider -laajennus oli silti haastava toteuttaa. Koska iOS 11 -version julkaisusta ei ollut kulunut vielä työn aloitushetkellä montaa kuukautta, dokumentaatio uudesta File Provider -laajennuksesta ja sen toteutuksesta oli heikohkoa. Osa virallisesta dokumentaatiosta koski vielä vanhempaa iOS 10 -yhteensopivaa versiota File Provider -laajennuksesta ja monet asiat joutui tekemään yrityksen ja erehdyksen kautta kokeilemalla ja testaamalla. Myös muu dokumentaatio koskien tiedostojen kanssa toimimista ei ollut aivan tyydyttävää, mutta tarpeeksi laajan etsinnän jälkeen sai riittävän hyvän kuvan, miten sovellus kannattaa toteuttaa.

Kokonaisuutena tutkimus- ja kehitystyö meni kuitenkin erittäin sujuvasti ja työn aikana ei syntynyt kehitystyön pysäyttäviä ongelmia. Työn teko sujuikin melko ripeästi. Työn sujuvuuteen vaikutti myös, että koko muu iOS-sovellus on myös minun toteuttamani, jolloin kaikki sen osa-alueet olivat hyvin tuttuja ja dokumentinhallinta oli helppo lisätä sovelluksen sisään. iOS-ohjelmointi Swift-kielellä on muutenkin jo tuttua ja sujuvaa. Vaatimukset olivat selkeät ja tilaaja tuki opinnäytteen suoritusta erinomaisesti. Olen todella tyytyväinen työn lopputulokseen.

Jos tekisin jotain toisin, sopisin iOS 10 -yhteensopivuuden poistamisesta tiedostojen muokkauksesta aiemmin ja ennen kehitystyön aloittamista, koska sen yrittäminen aiheutti turhaa työtä ja ongelmia. Kun yhteensopivuudesta sitten päätettiin luopua, jouduin kesken kehitystyön tarkistamaan suunnitelmia ja tekemään lisätutkimuksia, koska iOS 10 -yhteensopivuudesta luopuminen avasi muutaman uuden mahdollisuuden File Provider -laajennuksen lisäksi tiedostojen muokkaukseen, joita täytyi kokeilla. Tähän sisältyi sovelluksen hiekkalaatikon sisällä sijaitsevan Documents-kansion avaaminen muiden sovellusten käyttöön. Nämä mahdollisuudet paljastuivat lopulta mahdottomiksi hyödyntää tällä hetkellä, mutta olisi ollut parempi, jos kaikki tutkimus- ja suunnittelutyö olisi tapahtunut ennen kehitystyötä. Muilta osin suunnitelmat olivat hyvin tehtyjä ja helpottivat sovelluksen toteutusta.

LÄHTEET JA TUOTETUT AINEISTOT

APPLE INC. 2018a. How the program works [verkkoaineisto]. [Viitattu 2018-01-17]. Saatavissa: https://developer.apple.com/programs/how-it-works/

APPLE INC. 2018b. Deploy the next generation of apps to your enterprise. [Verkkoaineisto]. [Viitattu 2018-01-18]. Saatavissa: https://developer.apple.com/programs/enterprise/

APPLE INC. 2018c. Xcode IDE [Verkkoaineisto]. [Viitattu 2018-01-17]. Saatavissa: https://developer.apple.com/xcode/features/

APPLE INC. 2018d. About Swift [Verkkoaineisto]. [Viitattu 2018-01-17]. Saatavissa: https://swift.org/about/

APPLE INC. 2018e. iOS 11 [Verkkoaineisto]. [Viitattu 2018-01-21]. Saatavissa: https://www.apple.com/fi/ios/ios-11/

APPLE INC. 2018f. iPad mini 2 Retina-näytöllä - Tekniset tiedot [verkkoaineisto]. [Viitattu 2018-01-21]. Saatavissa: https://support.apple.com/kb/SP693?locale=fi_FI&viewlocale=fi_FI

APPLE INC. 2018g. File System Basics [verkkoaineisto]. [Viitattu 2018-01-25]. Saatavissa: https://developer.apple.com/library/content/documentation/FileManagement/Conceptual/FileSystemProgrammingGuide/FileSystemOverview/FileSystemOverview.html

APPLE INC. 2018h. UIActivityViewController [verkkoaineisto]. [Viitattu 2018-01-27]. Saatavissa: https://developer.apple.com/documentation/uikit/uiactivityviewcontroller

APPLE INC. 2018i. UIDocumentInteractionController [verkkoaineisto]. [Viitattu 2018-01-27]. Saatavissa: https://developer.apple.com/documentation/uikit/uidocumentinteractioncontroller

APPLE INC. 2018j. Previewing and Opening Files [verkkoaineisto]. [Viitattu 2018-01-27]. Saatavissa: https://developer.apple.com/library/content/documentation/FileManagement/Conceptual/DocumentInteraction_TopicsForIOS/Articles/PreviewingandOpeningItems.html

APPLE INC. 2018k. FileProvider [verkkoaineisto]. [Viitattu 2018-01-28]. Saatavissa: https://developer.apple.com/documentation/fileprovider

APPLE INC. 2018l. App Extensions Increase Your Impact [verkkoaineisto]. [Viitattu 2018-01-28]. Saatavissa: https://developer.apple.com/library/content/documentation/General/Conceptual/ExtensibilityPG/index.html#//apple_ref/doc/uid/TP40014214-CH20-SW1

APPLE INC. 2018m. Creating File Providers for Multiple Versions of iOS. [verkkoaineisto]. [Viitattu 2018-01-28]. Saatavissa: https://developer.apple.com/documentation/fileprovider/creating_file_providers_for_multiple_versions_of_ios

APPLE INC. 2018n. Storyboard [verkkoaineisto]. [Viitattu 2018-01-30]. Saatavissa: https://developer.apple.com/library/content/documentation/General/Conceptual/Devpedia-CocoaApp/Storyboard.html

APPLE INC. 2018o. A Closer Look at Table View Cells [verkkoaineisto]. [Viitattu 2018-01-30]. Saatavissa: https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/TableView_iPhone/TableViewCells/TableViewCells.html

APPLE INC. 2018p. A Closer Look at Table View Cells [verkkoaineisto]. [Viitattu 2018-01-30]. Saatavissa: https://developer.apple.com/documentation/uikit/uicollectionview

APPLE INC. 2018q. UIDocumentPickerViewController [verkkoaineisto]. [Viitattu 2018-02-03]. Saatavissa: https://developer.apple.com/documentation/uikit/uidocumentpickerviewcontroller

APPLE INC. 2018r. providePlaceholder(at:completionHandler:) [verkkoaineisto]. [Viitattu 2018-02-03]. Saatavissa: https://developer.apple.com/documentation/fileprovider/nsfileproviderextension/1623483-provideplaceholder

COLLAN, Samu 2017-11-22. Toimitusjohtaja. [Palaveri.] Kuopio: East Dataconst Oy.

CONGRID OY 2018a. Congrid iOS-applikaatio [Verkkoaineisto]. [Viitattu 2018-01-13]. Saatavissa: https://itunes.apple.com/fi/app/congrid/id918818781?l=fi&mt=8

CONGRID OY 2018b. Congrid mobiiliapplikaatio [Verkkoaineisto]. [Viitattu 2018-01-13]. Saatavissa: http://www.congrid.fi/applikaatio/

DROPBOX INC. 2018. Company Info [Verkkoaineisto]. [Viitattu 2018-01-13]. Saatavissa: https://www.dropbox.com/news/company-info

EAST DATACONST OY 2018. East Dataconst [Verkkoaineisto]. [Viitattu 2018-01-13]. Saatavissa: http://www.eastdataconst.fi/

FLUENT PROGRESS RT OY 2018. Fluent Valokuvadokumentointi [Verkkoaineisto]. [Viitattu 2018-01-13]. Saatavissa: https://www.fluentprogress.fi/ratkaisualueet/rakentaminen/valokuvadokumentointi

INDUSTRIAL ITC OY 2018. Dokumenttien hallinta [Verkkoaineisto]. [Viitattu 2018-01-16]. Saatavissa: https://www.iitc.fi/fi/page/221

JAYESH Kawli 2016-05-28. Using UIActivityViewController (Swift) [verkkoaineisto]. [Viitattu 2018-01-27]. Saatavissa: http://jayeshkawli.ghost.io/using-uiactivityviewcontroller-swift/

LAINIALA, Pekko 2016. Testissä Apple iPhone SE: ”Pieni, mutta kallis puhelin”. [Verkkoaineisto]. [Viitattu 2018-01-18]. Saatavissa: https://muropaketti.com/mobiili/testissa-apple-iphone-se-pieni-mutta-kallis-pikkupuhelin/

LEHTO, Joonas 2016. Rakennustyön dokumentointi mobiilidokumentointipalvelulla. Tampereen ammattikorkeakoulu. Rakennustekniikka. Rakennustuotanto. Opinnäytetyö. [Viitattu 2018-01-20]. Saatavissa: http://urn.fi/URN:NBN:fi:amk-201604174456

NAUKKARINEN, Riikka ja ANTTONEN, Heidi 2010. Dokumenttien hallintajärjestelmän valinta Ovako Bar Oy Ab:lle. Saimaan ammattikorkeakoulu. Tietotekniikan koulutusohjelma. Opinnäytetyö. [Viitattu 2018-01-14.] Saatavissa: http://urn.fi/URN:NBN:fi:amk-201004277037

PIXACTOR OY 2018. Buildie [Verkkoaineisto]. [Viitattu 2018-01-13]. Saatavissa: http://www.buildie.fi/

RAKENNUSTEOLLISUUS RT RY 2018. Työturvallisuuskansio ja sen malliasiakirjat sekä lomakkeet [verkkoaineisto]. [Viitattu 2018-01-20]. Saatavissa: https://www.rakennusteollisuus.fi/Toimialat/Talonrakennusteollisuus/Hyotytietoa-tyomaille/Laatu-ymparisto-tyoturvallisuus/Tyomaan-tyoturvallisuus/Tyoturvallisuuskansio-pk-rakennusyrityksille1/Malliasiakirjat/

SALMI, Miikka 2016. Työmaan aikainen dokumentointi ja arkistointi saneerauskohteissa. Turun ammattikorkeakoulu. Rakennustekniikka. Tuotantojohtaminen. Opinnäytetyö. [Viitattu 2018-01-21]. Saatavissa: http://urn.fi/URN:NBN:fi:amk-2016121620808

STATISTA INC 2018. Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2017 [Verkkoaineisto]. [Viitattu 2018-01-21]. Saatavissa: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

VIITALA, Mika 2010. Dokumentinhallinta Pk-yrityksessä. Keski-Pohjanmaan ammattikorkeakoulu. Tietotekniikan koulutusohjelma. Opinnäytetyö. [Viitattu 2018-01-14.] Saatavissa: http://urn.fi/URN:NBN:fi:amk-201104264831