jamk

Docean-ohjelmiston opastustoiminnallisuuden suunnittelu ja toteutus

Antti Tarvainen

Opinnäytetyö, AMK
Toukokuu 2021
Tietojenkäsittely ja tietoliikenne
Tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Tarvainen Antti

Docean-ohjelmiston opastustoiminnallisuuden suunnittelu ja toteutus

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu 2021, 49 sivua

Tietojenkäsittely ja tietoliikenne. Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: kyllä

Tiivistelmä

Opastuksesta on hyötyä käyttäjille kuin myös käyttäjien hankinnassa. Käyttäjälle voidaan näyttää mitä kaikkia toimintoja sovellus sisältää ja miten niitä käytetään. Opastuksen avulla voidaan tarjota käyttäjille tietoa, jota he eivät välttämättä löytäisi pelkästään kokeilemalla sovellusta, tai verrattuna toimintojen kokeilemiseen, opastus voi antaa tietoa nopeammin siitä, miten ne toimivat.

Tavoitteena työssä oli tutkia tietoa käyttäjän opastuksesta, opastuksen hyödystä eri tilanteissa ja opastuksen käytettävyydestä. Tavoitteena oli myös hankitun tiedon avulla suunnitella ja toteuttaa opastustoiminnallisuutta toimeksiantajan markkinoilla olevaan Docean-dokumentointipalvelun iOS- ja Android-alustojen mobiilisovellukseen. Opastustoiminnallisuuden lisäämisellä toimeksiantaja odotti saavansa vähennettyä työntekijöidensä käyttämien työtuntien määrää sovelluksen käyttökoulutukseen. Toimeksiantaja odotti myös opastustoiminnallisuuden kykenevän auttamaan ei niin tekniikkataitoisia ihmisiä pääsemään nopeammin ymmärtämään tarjoaman hyödyn dokumentoinnin työvaiheissa.

Tutkimusmenetelmänä oli tutkimuksellinen kehittämistoiminta. Kehittämistyössä käyttöliittymän mallit toteutettiin Adobe XD -ohjelmistolla ja ohjelmoinnin toteutus React Native -sovelluskehyksen avulla JavaScript-ohjelmointikielellä. Ohjelmointi suoritettiin Microsoft Visual Studio Code -tekstieditorilla ja ohjelmoinnin tukena varsinaisen tuotoksen testaamiseen käytettiin Xcode- ja Android Studio -ohjelmointiympäristöjä emuloimalla Android- ja iOS-laitteita.

Kehittämistoiminnan tuloksena toimeksiantajalle luovutettiin uusi React Native -koodikomponentti. Uudella koodikomponentilla voidaan opastaa toimintoja muun näkymän päälle tulevien vihjetekstien avulla. Vihjetekstin ympäröimä laatikko sisältää kolmion, jonka kärki osoittaa selkeyden vuoksi toimintoon, jota ollaan opastamassa.

Uuden koodikomponentin voi upottaa olemassa olevaan Docean-sovelluksen lähdekoodiin käyttöliittymäkomponenttien ympärille. Komponentti toimii täysin ilman muutoksia lähdekoodiin, lukuun ottamatta komponentin lisäämistä, jos halutaan ohjeistaa käyttäjälle yksittäistä toimintoa. Jos taas halutaan opastaa koko näkymän toiminnot peräkkäin, lähdekoodiin joutuu tekemään pieniä muutoksia. Tällä komponentilla toimeksiantaja voi helposti lisätä opastusta sitä tarvitsevien toimintojen yhteyteen.

Avainsanat (asiasanat)

käyttöliittymäsuunnittelu, käytettävyys, mobiilisovellukset, ohjelmistoala, ohjelmistotuotanto

Muut tiedot (salassa pidettävät liitteet)

-

jamk

Description

Tarvainen Antti

Design and implemention of tutorial functionality for Docean-application

Jyväskylä: JAMK University of Applied Sciences, May 2021, 49 pages

Information and Communications. Degree Programme in Information and Communications Technology. Bachelor´s thesis.

Permission for web publication: Yes

Language of publication: Finnish

Abstract

Tutorials are useful for both the users and for acquisition of users. They can show the user what kind of functionality the app provides and how they can be used. Tutorials can provide information for users that they may be unable to find otherwise just by trying out the app, or when compared to trying out the functionality, guidance can provide information faster about what the functionality does.

The aim was to research information about tutorials, the benefits of tutorials in different situations and usability of tutorials. Also the aim was to use the researched information for designing and implementing some kind of tutorial functionality for the clients Docean documentation applications mobileapp for iOS and Android platforms. The client expected the added tutorial functionality to lessen the work time their employyes needed to spend in training users. Also the client expected the tutorial functionality being able to help the less tech-savvy users to access information faster about how the app can help in a documentation flow.

Research method was research and development. In development the UI-models were made with Adobe XD software and the software implementation with React Native framework in JavaScript coding language. The actual writing of the code was done in Microsoft Visual Studio Code text editor and for testing the software in emulated Android and iOS devices, Android Studio and Xcode integrated development enviroments were used.

A new React Native code component was presented to the client as the result of development. The new code component could guide users by showing a tooltip on top of the view. To clarity what functionality was being guided, tooltip included a triangle whose tip pointed to the functionality.

The new code component could be embedded into the existing Docean-app source code by wrapping the UI-components with it. The component worked without making changes to the source code if it was used for guidance of a single functionality. And if the component was needed for a tutorial of a whole screen, it needed small modifications to the source code. With this component the client could easily add guidance for a functionality that needs it.

Keywords/tags (subjects)

UI-design, usability, mobilesoftware, software industry, software production

Miscellaneous (Confidential information)

-

Sisältö

Kuviot

1 Johdanto

Melkein jokaiselle tehtävälle mitä ihminen voi keksiä, on olemassa useampia mobiili- ja/tai verkkosovelluksia. Yhtenä keinona joukosta erottumiseen voi olla sovelluksen tekeminen muita helpommaksi ja intuitiivisemmaksi käyttää. Jos sovelluksella x tietyn asian tekemiseen kuluu 2 minuuttia ja sovelluksella y saman asian tekemiseen 5 minuuttia, kannattaako sovelluksen y käyttöä edes jatkaa? Varsinkin jos samaa asiaa toistetaan päivittäin useaan kertaan, kertyy hukkaan mennyttä aikaa vuosien vieriessä huomattava määrä. Jos tämä tuhlattu aika on vielä työntekijöiden palkallista aikaa, voisi sovelluksen vaihto tuoda säästöjä työnantajalle.

Opastoiminnallisuus voisi auttaa käyttäjiä sovellusta vaihtaessa tai ottaessa käyttöön näyttäen, miten sovellus toimii. Opastuksen lisääminen sovellukseen ei suorasti vaikuttaisi toimintojen suorittamisen nopeuteen, mutta sillä voidaan ohjeistaa käyttäjää käyttämään sovelluksen toimintoja paremmin tai kehittäjien tarkoittamalla tavalla, jolloin hukkaan heitetty aika vähenee. Tämän opinnäytetyön tavoite on siis suunnitella ja toteuttaa toiminnallisuutta Docean-mobiilisovellukseen, joka opastaa sovelluksen käytössä.

1.1 Toimeksiantaja

Toimeksiantajana toimii 2019 vuonna perustettu ohjelmistoalan yritys Document Ocean Oy. Document Ocean Oy koostuu neljästä henkilöstä, kahdesta ohjelmistokehittäjästä ja kahdesta myyntiammattilaisesta. Document Ocean Oy on erikoistunut mobiilidokumentointiin, dokumenttien digitaaliseen allekirjoittamiseen ja projektinhallintaan.

Toimeksiantajan päätuote on yrityksille, ensisijaisesti rakennus- ja sähköalan yrityksille suunnattu Docean-dokumentointisovellus. Docean-sovellus on tarkoitettu tehostamaan pakollisen laillisesti vaaditun ja myös kaikenlaisen muun dokumentoinnin tuottamista. Sovelluksen on tarkoitus vähentää dokumenttien laatimiseen, arkistoimiseen ja selaamiseen kuluvaa aikaa.

1.2 Opastustoiminnallisuuden kehittämisen motivaatio

Jos käyttäjä kokee sovelluksen käytön liian vaikeaksi ja tämän takia hitaaksi tai turhauttavaksi käyttää, voi asiakas huonoimmassa tapauksessa jättää sovelluksen ostamatta. Nykyisillä asiakkailla tämä voisi johtaa toiseen markkinoilla olevaan vaihtoehtoon siirtymistä tai tilauksen perumiseen.

Vastaava tilanne on saanut jo asiakkaan lopettamaan sovelluksen käytön. Sovelluksesta puuttui ohjeistus pankkitunnistautumistilanteessa mobiilitunnistautumisvaihtoehdon toimimattomuudesta, mutta kolmannen osapuolen pankkitunnistautumispalvelussa vaihtoehto mobiilitunnistautumiselle kuitenkin oli käyttäjälle näkyvissä ja käytettävissä.

Vaikka käyttäjä ei lopettaisi turhautumisen seurauksena sovelluksen käyttöä, vastaavat puutteet ohjeistuksessa voivat vaikuttaa negatiivisesti myös sovelluksen maineeseen sovelluskauppojen arvostelujen kautta.

Document Ocean Oy tarjoaa tällä hetkellä ratkaisuksi tähän ongelmaan maksua vastaan käyttökoulutusta yrityksille osana palvelun käyttöönottoa. Koulutuksen ollessa maksullinen palvelu varsinkaan useimmat pienistä yrityksistä eivät halua palvelua, vaan he ottavat sovelluksen käyttöön itsenäisesti, ongelmaa ei siis ole tällä ratkaistu. Document Ocean Oy:llä ei myöskään ole tällä hetkellä resursseja tarjota koulutusta ilmaiseksi.

Yritystä kohden käyttöönottoon kuluu yleensä n. kaksi tai kolme tuntia aikaa. Käyttöönotossa käydään läpi sovelluksen toiminnallisuuksia, opastetaan käyttämään sovellusta mahdollisimman tehokkaasti ja vastataan kysymyksiin. Käyttöönotto tarjoaa pientä lisätulon lähdettä, mutta se myös vähentää aikaa, jota käyttökoulutuksia pitävä myyjä voi käyttää myynnin ääressä. Käyttökoulutuksille ei myöskään ole kysyntää riittävästi, jotta niiden pitämiseen voitaisiin palkata erillisiä työntekijöitä myyjien asemasta.

Jos käyttökoulutusta vastaavaa ohjeistusta toteutetaan itse sovellukseen, käyttökoulutukset voidaan jättää kokonaan pois tai niiden kestoa voidaan pienentää. Opastoiminnallisuuden toteuttaminen helpottaa siis käyttäjiä ja myös vapauttaa lisää aikaa Docean-sovelluksen markkinoinnille.

1.3 Tutkimusmenetelmä

Työ toteutetaan tutkimuksellisena kehittämistoimintana. Tutkimuksellinen kehittämistoiminta on systemaattista toimintaa tiedon keräämiseen ja tiedon keräämisen tulosten käyttämiseen kehitystoiminnassa, esimerkiksi olemassa olevien järjestelmien parantamiseen (Tutkimus- ja kehittämistoiminta. N.d.).

Koska toimeksiannon tavoitteena on varsinaisesti toteuttaa opastoiminnallisuus, tätä tutkimusmenetelmää käytetään, jotta voidaan parhaiten toteuttaa Docean-sovellukseen uutta toiminnallisuutta tuotetun tiedon avulla.

2 Tietoperusta

Sovellusten opastoiminnallisuutta käytetään tuomaan esiin sovelluksen hyödyt ja opettamaan missä mitkäkin ominaisuudet sijaitsevat ja mitä ne tekevät. Opastoiminnallisuus voi kuitenkin huonoimmillaan turhauttaa uuden käyttäjän, mutta käyttäjä voi myös turhautua, jos hän ei tiedä miten tai mitä tehdä. (Rethinking Mobile Tutorials: Which Patterns Really Work? Neil T. 2021.)

2.1 Käytettävyys

World Wide Web Consortium on luonut saavutettavuusohjeet verkkosisällölle, joita voidaan soveltaa merkittävimmältä osin mobiilisovellukseen toteutettavassa opastoiminnallisuudessa. Uusin versio saavutettavuusohjeista on vuonna 2008 luotu 2.0 versiota täydentävä versio 2.1 (Web Content Accessibility Guidelines (WCAG) 2.1. 2019.).

W3C-organisaation perusti vuonna 1994 Tim Berners-Lee, jota on kutsuttu myös World Wide Webin keksijäksi. W3C-organisaation tavoite on luoda protokollia ja ohjenuoria, jotka varmistavat webin pitkäaikaisen kasvun. (About W3C. N.d.)

Verkkosisällön saavutettavuusohjeet on luotu tarjoamaan sovelluskehittäjille yhteinen standardi, jotka auttavat verkkosisällön tekemisessä saavutettavammaksi henkilöille, joilla on vammoja tai rajoitteita. Ohjeet auttavat myös tekemään verkkosisällöstä helpommin käytettävää ikääntyville

henkilöille ja parantamaan käytettävyyttä kaikille käyttäjille. (Web Content Accessibility Guidelines (WCAG) 2.1. 2019.)

WCAG 2.1 -onnistumiskriteeristö on dokumentoitu lausumina, jotka voidaan testata automaatiolla ja suositellusti myös ihmisen suorittamalla testillä. Kriteeristö on toteutuksen teknologiasta riippumaton. Erillisissä dokumenteissa kuitenkin kuvaillaan, miten kriteerien vaatimukset voidaan toteuttaa tietyillä teknologioilla. (Web Content Accessibility Guidelines (WCAG) 2.1. 2019.)

Kaikkia standardeja ei voida toteuttaa mobiilisovellukselle Docean-mobiilisovelluksen ollessa natiivisovellus ja standardin ollessa suunnattu verkkoselaimissa näytettävälle sisällölle. Tietokoneella käytettävien verkkosivustojen ja natiivisovelluksien välillä suurin eroavaisuus on selaimien tarjoamien yleisesti tunnettujen toimintojen puuttuminen, esimerkiksi sisällön suurennus prosenttimääräisesti. Ohjeet sisältävät erittäin paljon hiiren kursoriin ja näppäimistöön liittyviä parannuksia, joita ei voida mobiilisovelluksella toteuttaa. Ohjeet sisältävät myös paljon kohtia käyttäjän syötön helpottamiseen, mutta opastoiminnallisuus ei todennäköisesti tule sisältämään lomakkeita tai muuta vastaavaa syötteitä sisältävää toiminnallisuutta.

Jo opastoiminnallisuuden suunnitteluvaiheessa tulee ottaa huomioon kaikki edelliset kriteerit värimaailman valinnassa ja toiminnallisuuden suunnittelussa. Jos kriteereitä ei oteta huomioon suunnitteluvaiheessa, voi opastoiminnallisuuden muuttaminen kriteereitä vastaavaksi olla huonoimmassa tapauksessa aikamäärältään koko opastoiminnallisuuden toteutusta vastaava.

2.2 Käyttöönoton opastus

Ensimmäisen käyttökerran perehdytys on erittäin tärkeää, jotta käyttäjä saadaan palaamaan takaisin käyttämään sovellusta. Hyvällä käyttöönoton prosessilla vähennetään sovelluksen hylkäämisen määrää. Tekniikkataitoiset ihmiset eivät välttämättä tarvitse minkäänlaista perehdytystä sovelluksen käyttöönotolle, mutta muille jonkunlainen opastus voi olla erittäin tärkeä, jotta he pääsevät aloittamaan sovelluksen käytön mahdollisimman sujuvasti. (Mobile app onboarding: best practices and examples. 2020.)

Käyttöönoton opastuksen luominen on kuitenkin ongelmallista useista syistä. Perehdytyksen läpikäyminen vaatii käyttäjältä keskittymistä ja ylimääräistä aikaa eikä käyttäjä edes välttämättä

muista perehdytystä enää ohjeita tarvitessaan. Sovelluksen kehittäjiltä taas perehdytyksen luonti vie pois aikaa, joka voitaisiin käyttää käyttöliittymän parantamiseen. Jos aika käytetään käyttöliittymän parantamiseen, ei erillistä opastusta välttämättä edes tarvita. (Mobile-App Onboarding: An Analysis of Components and Techniques. Joyce A. 2020.)

Perehdytys pitäisi toteuttaa vain tietynlaisille sovelluksille. Sovellukset, jotka tarvitsevat käyttäjältä lisätietoa sovelluksen käytön räätälöintiä varten, voivat käyttää perehdytystä samanaikaisesti tietojen pyytämiseen ja ominaisuuksien esittelyyn. Paljon valtavirrasta poikkeavat tai muuten erittäin uniikit sovellukset voivat myös tarvita perehdytystä ensimmäisellä käyttökerralla. (Mobile-App Onboarding: An Analysis of Components and Techniques. Joyce A. 2020.)

Jos kuitenkin käyttäjälle näytetään sovelluksen ensimmäisellä käyttökerralla perehdytysnäkymä, täytyy käyttäjällä olla tiedossa, kuinka paljon opastusta on vielä jäljellä. Useimmat käyttäjät haluavat hypätä mahdollisimman nopeasti sovelluksen käyttöön ja tällöin käyttäjän on nähtävä, paljon hän joutuu vielä käyttämään aikaa opasnäkymässä. Opastus käyttöönotolle ei siis saa kestää liian kauaa. (Mobile app onboarding: best practices and examples. 2020.)

Kaikki ylimääräinen käyttöönotolle kuulumaton on parempi siirtää vaiheeseen, jossa niitä oikeasti tarvitaan, esimerkiksi mobiilisovelluksen vaatimien lupien, kuten sijaintitietojen käyttäminen tai ilmoitusten lähettäminen. Jos ensikäyttöä viivytetään vielä vastaavilla toiminnoilla, voi tämä jo käännyttää käyttäjän pois sovelluksesta. Parasta olisi, jos käyttäjä pääsisi kokeilemaan sovelluksen toiminnallisuutta ilman rekisteröitymistä tai ennen kuin sovellus vaatii suurempaa määrää profiilitietoja käytön jatkamista varten. (Mobile app onboarding: best practices and examples. 2020.)

2.3 Sovelluksen toimintojen opastus

Toimintojen opastukseen voi ottaa mallia peliteollisuudesta. Peleissä käytetään vain vähän tekstiä koska se tuhoaa pelin rytmityksen ja suurin osa pelaajista ohittaa pitkät tekstikappaleet, vaikka he tarvitsisivatkin opastusta pelin sisäistämiseen. (Rethinking Mobile Tutorials: Which Patterns Really Work? Neil T. 2021.)

Peleissä pelaaja yleensä opetetaan asteittain pelin ominaisuuksiin pelaamisen avulla. Pelaajalla voi esimerkiksi olla alussa vain muutama ominaisuus käytössä ja myöhemmin näitä avataan hänelle

lisää. Pelin vaikeustaso yleensä kasvaa edettäessä pelissä. Pelaajan ei heti alussa oleteta osaavan kaikista vaikeimpia toimintoja, vaan kun pelaaja on oppinut jo yksinkertaiset toiminnot ja pelin toimintamallin, pelaajalle avataan uusia mahdollisuuksia. Tämänkaltainen toiminnallisuus myös palkitsee käyttäjää.

> *”Et voi pudottaa uusia pelaajia tulitaistelun keskelle ja odottaa heidän nauttivan kokemuksesta. Useimmat pelaajat kuolisivat jo ennen kuin he tajuaisivat miten ampua heidän aseillaan ja puolustautua” (Rethinking Mobile Tutorials: Which Patterns Really Work? Neil T. 2021.)*

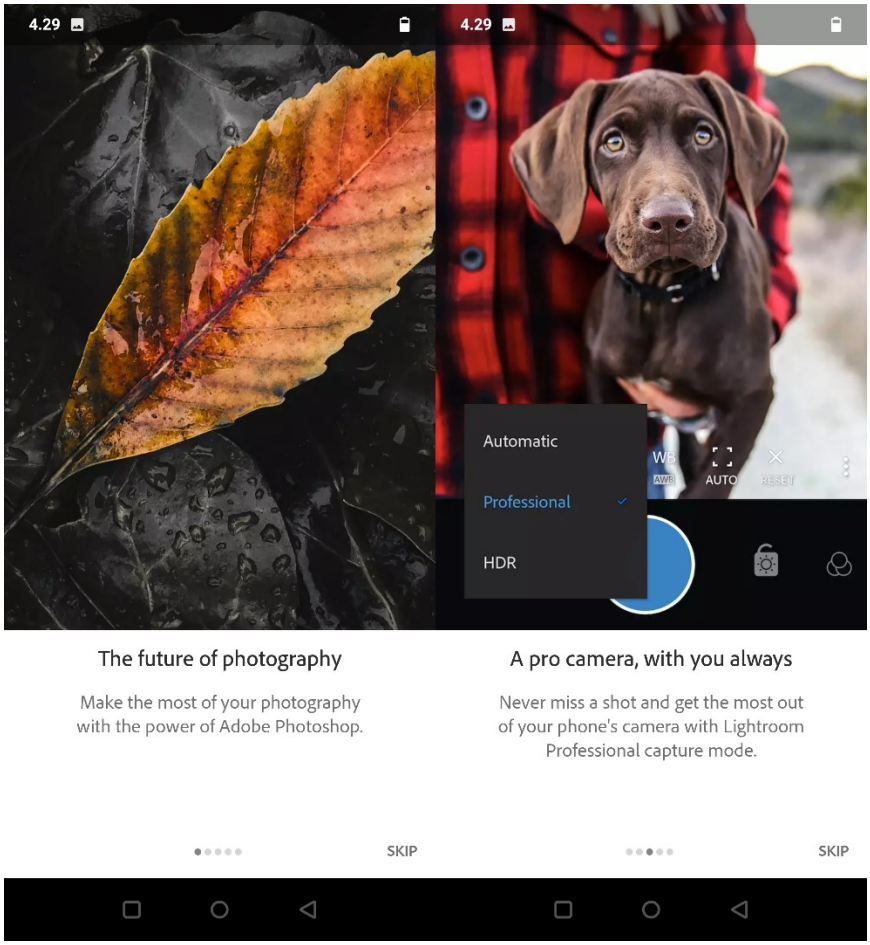
Jos käyttäjälle opetetaan kaikki heti alussa, hän todennäköisesti unohtaa tiedon ennen kuin sitä tarvitsee soveltaa. Tieto kannattaa tarjota käyttäjälle pienissä paloissa vain silloin kun sitä tarvitaan. (Rethinking Mobile Tutorials: Which Patterns Really Work? Neil T. 2021.)

Jos opastoiminnallisuus on tehty itse sovelluksesta erillään olevien opastusnäkymien avulla, jotka näytetään ennen sovelluksen toiminnallisuuksien avaamista, käyttäjä todennäköisesti ohittaa näkymät, jos se on tehty mahdolliseksi. Käyttäjällä ei myöskään mene sovelluksen toimintojen oppimiseen huomattavasti enemmän aikaa, vaikka käyttäjä ei ohittaisi opastusnäkymiä. Opastusnäkymien lukeminen siis vie turhaan aikaa käyttäjältä, minkä hän voisi käyttää sovelluksen tutkimiseen omatoimisesti. Käyttäjä voi jopa hahmottaa sovelluksen toiminnallisuudet vaikeammiksi käyttää, kuin mitä ne oikeasti ovat, jos ne on selitetty opastusnäkymissä. (Mobile Tutorials: Wasted Effort or Efficiency Boost? Joyce A. 2020.)

2.4 Markkinoilla käytetyt opastustavat

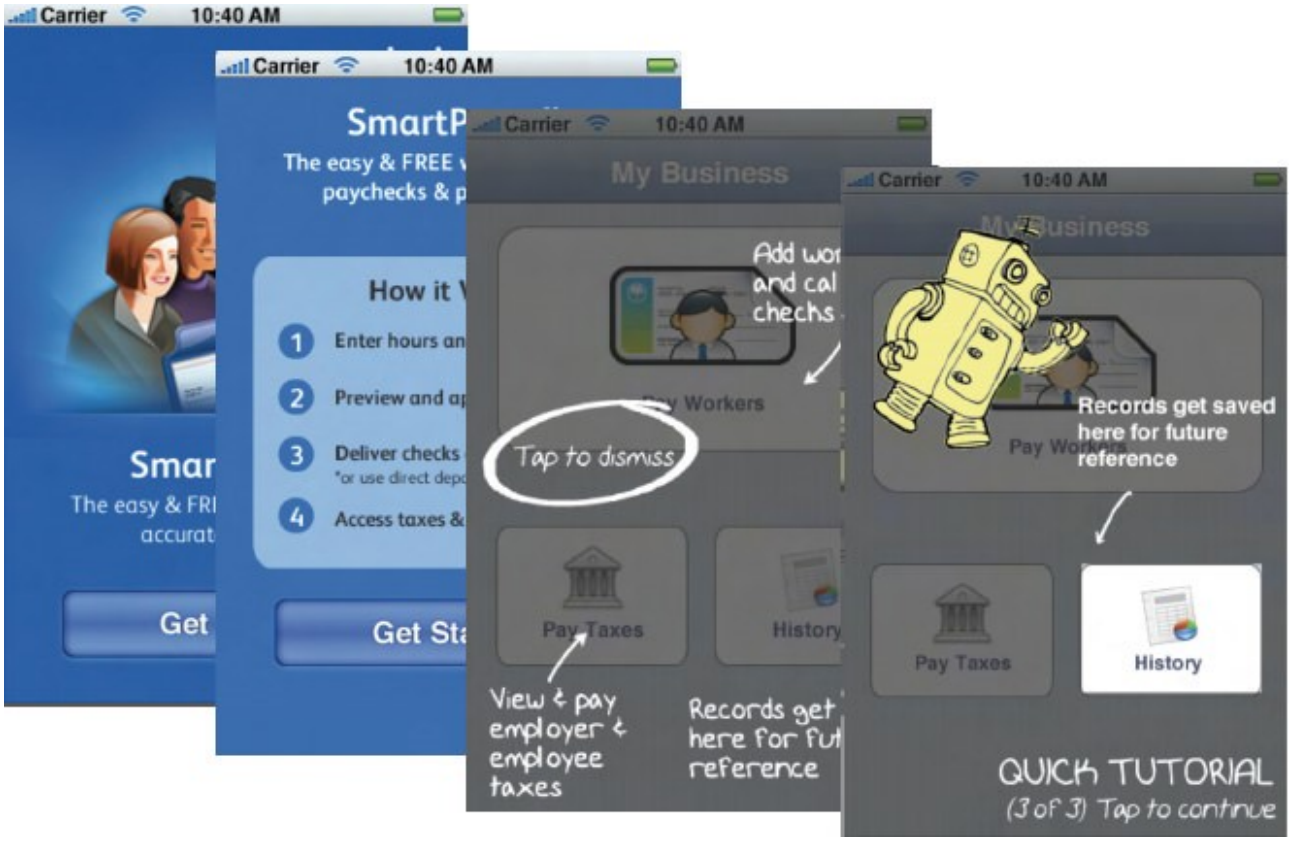
Opastamiseen olisi käytettävissä jo muistakin sovelluksista tunnettuja tapoja. Tunnettujen sovellusten opastustoiminnallisuuden tyyliä matkimalla voidaan olettaa osan käyttäjistä jo käyttäneen vastaavia toimintoja ja tottuneen näiden toimintatapoihin.

Ensimmäisenä tapana olisi esitellä toiminnot erillisissä näkymissä ennen varsinaiseen sovellukseen pääsyä, esimerkiksi Adobe Lightroom- mobiilisovellus käyttää vastaavaa opastustoiminnallisuutta (ks. kuvio 1). Erillisnäkymät sopisivat hyvin sovelluskauppoihin, missä sovellusta esitellään mahdolliselle käyttäjälle ennen ensimmäistä asennusta, mutta varsinaiseen opastukseen sovelluksen sisällä tämä ei ole hyvä vaihtoehto.



Kuvio 1. Adobe Lightroom sovelluksen käyttöönoton opastus (Adobe Lightroom. 2021).

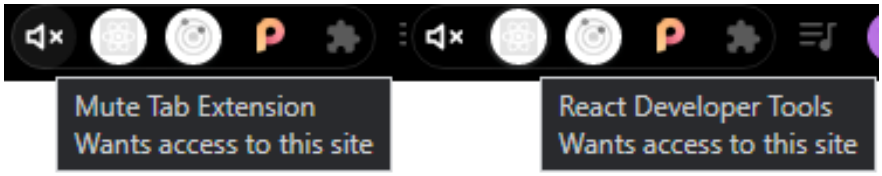
Vähemmän käyttöönottoa hidastavana vaihtoehtona olisi rekisteröitymisen jälkeen varsinaisen näkymän päälle asetettavat läpikuultavat ikkunat, jotka estävät toiminnallisuudet siihen asti, että käyttäjä käy ikkunat läpi. Ikkunat voisivat sisältää informatiivista tekstiä ja esimerkiksi korostaa toiminnallisuuden varsinaisesta näkymästä (ks. kuvio 2).



Kuvio 2. Snap Payroll -sovelluksen käyttöönoton opastus (Rethinking Mobile Tutorials: Which Patterns Really Work? Neil T. 2021).

Tämänkaltaista opastusta voisi käyttää koko näkymän toimintojen esittelemiseen, alaosan numeroinnin (ks. kuvio 2) indikoidessa kuinka monta näkymää on vielä käymättä läpi. Tällöin painamalla mistä tahansa siirtäisi opastusikkunan seuraavaan tilaan ja kun kaikki ominaisuudet on käyty läpi, painallus sulkisi opastusikkunan.

Samankaltainen tapa opastaa käyttäjää vähemmän häiritsevästi olisi näyttää vihjetekstejä (ks. kuvio 3). Tätä tapaa käytetään hyvin monessa sovelluksessa, ei pelkästään mobiilisovelluksissa. Vihjetekstit voisivat lisäksi osoittaa kyseiseen toimintoon selkeyden kannalta. Tämä tapa olisi myös nopein toteuttaa ja vähiten muutoksia nykyiseen sovellusrakenteeseen vaativa.



Kuvio 3. Google Chrome -selaimen lisäosien vihjelaatikot (Google Chrome. 2021).

Vihjeteksteillä opastamalla mitään toimintoja ei tarvitsisi poistaa käytöstä. Käyttäjä pystyisi sulkemaan vihjetekstin painamalla sen alueelta tai se sulkeutuisi tietyn ajan kuluttua omatoimisesti.

Jos vihjetekstejä käytettäisiin opastamaan koko näkymän toiminnallisuutta, niistä painamalla vihjeteksti siirtyisi aina seuraavan ominaisuuden yhteyteen ja päivittäisi tekstin. Molempia edellä mainituista tavoista voitaisiin käyttää koko näkymän tai yksittäisten toimintojen esittelemiseen.

2.5 Mallin luomisen arvo käyttöliittymäsuunnittelussa

Sovelluksen suunnittelussa voidaan käyttää hyödyksi käyttöliittymämallia. Malli on suunnitelma käyttöliittymästä ja sisältää ominaisuuksia, joita lopullinen sovellus tulee sisältämään. Malli voidaan tehdä staattisesti, jolloin se sisältää toiminnallisuuden vain visuaalisesti. Staattisessa mallissa siis voi olla painikkeita ja muuta vastaavaa, mutta niitä painamalla ei tapahdu mitään. Tällaista staattista mallia kutsutaan rautalankamalliksi. (What is a Mockup? Hufford B. 2021.)

Rautalankamallista yhtä astetta pidemmälle vietyä mallia kutsutaan nimellä mockup. Rautalankamalli on siis raaka luonnos käyttöliittymästä, se ei sisällä edes värejä ja sitä käytetään esittelemään käyttöliittymän elementtien asetelmaa. Mockup taas vastaa melkein täysin lopullista käyttöliittymän visuaalisuutta väreineen ja kuvineen. (What is a Mockup? Hufford B. 2021.)

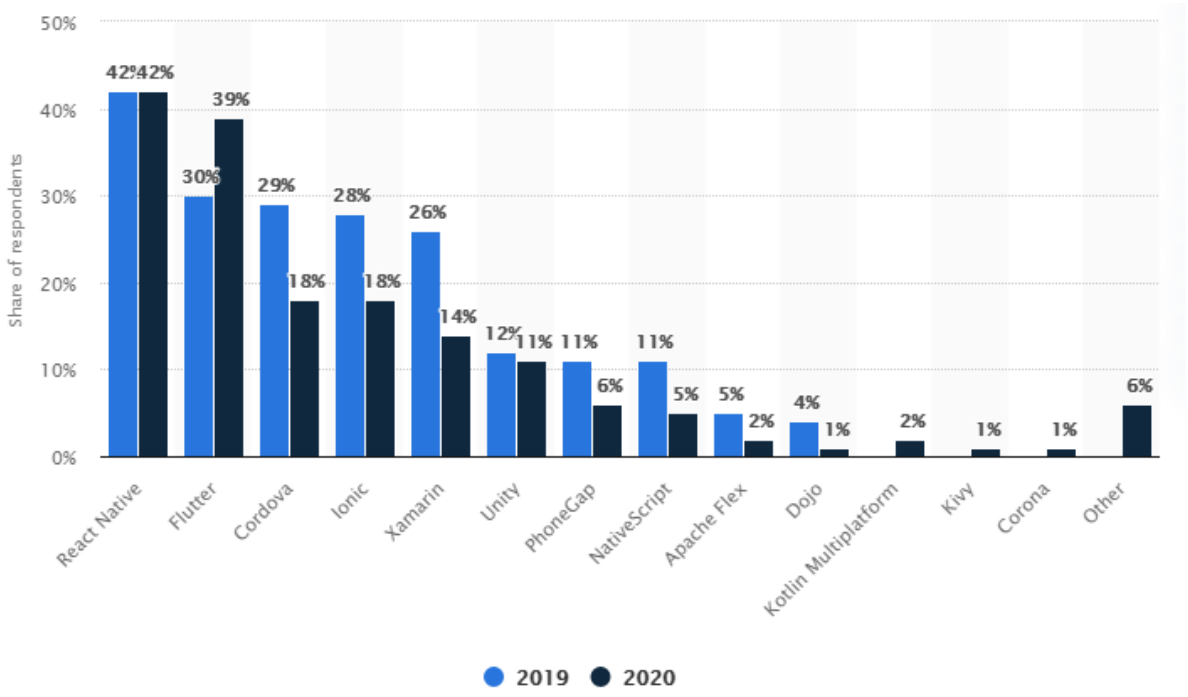
Mallin avulla voidaan hyväksyttää asiakkaalla käyttöliittymän visuaalinen ilme. Tämän jälkeen malliin voidaan tehdä myös visuaalisen ilmeen lisäksi täysi toiminnallisuus. Tällaista mallia kutsutaan prototyypiksi. Prototyyppi toimii samankaltaisesti, kuin oikeatkin sovellukset ja prototyyppia käyttämällä saadaan helposti palautetta asiakkaalta tai käyttäjiltä sovelluksen käytettävyydestä ja toimivuudesta ennen varsinaista panostusta sovelluksen ohjelmoimiseen. (What is a Mockup? Hufford B. 2021.)

Kaikista kallein ja hitain vaihe käyttöliittymäsuunnittelussa on prototyypin luominen. Prototyypille on varsinaisesti tarvetta vain silloin, jos käyttöliittymän interaktiivisuudesta halutaan palautetta. Prototyypin luominen voi viedä myös huomattavan määrän aikaa henkilöltä, joka ei ennen ole käyttänyt työkaluja prototyyppien luomiseen.

Kuitenkin kaikki nämä vaiheet nopeuttavat sovelluksen varsinaiseen ohjelmointiin vaativaa aikaa ja muutosten tekeminen näihin malleihin on vielä helppoa ja kustannustehokasta verrattuna muutosten toteuttamiseen jo ohjelmoituun sovellukseen.

2.6 Järjestelmäriippumaton sovelluskehitys

Erilaisia vaihtoehtoja mobiilisovelluksen kehittämiseen on useita. Suosituin kehittäjien käyttämä järjestelmäriippumaton sovelluskehys vielä vuonna 2020 oli React Native (ks. kuvio 4). Sovelluskehys on alusta ohjelmien kehittämiseen. Se tarjoaa perustukset, joiden avulla sovelluskehittäjä voi rakentaa ohjelmia, sovelluskehys voi esimerkiksi sisältää valmiita koodinpätkiä, joiden avulla voidaan prosessoida käyttäjän syötettä tai hallita fyysisiä laitteita. (Framework. 2013.)



Kuvio 4. Suosituimmat järjestelmäriippumattomat sovelluskehykset vuosina 2019 ja 2020 (Cross-platform mobile frameworks used by software developers worldwide in 2019 and 2020. Shanghong L. 2020).

Myös Docean-mobiilisovellus on rakennettu React Native -sovelluskehyksellä iOS- ja Android-alustoille. React Native -sovelluksissa voidaan käyttää samaa koodillista rakennetta, kuin React-sovelluksissa. React on JavaScript-kirjasto käyttöliittymien rakentamiseen pienien ja eristettyjen koodikappaleiden avulla, näitä koodikappaleita kutsutaan “komponenteiksi” (React. 2021). React Native taas on avoimen lähdekoodin sovelluskehys, jonka avulla voidaan kirjoittaa mobiiliapplikaatioita JavaScript-ohjelmointikielellä (React Native. 2021). React Native tarjoaa siis mahdollisuuden luoda mobiilisovelluksen melkein samaa lähdekoodia käyttämällä, kuin verkkosovelluksen luomisessa. React Native myös mahdollistaa saman lähdekoodin käyttämisen monella eri alustalla, esim. iOS, Android- ja Xbox-alustoilla (React Native. 2021).

Ilman React Native -sovelluskehystä mobiilisovellukseen kirjoitetaan komponentteja Android-alustalle Kotlin- tai Java-ohjelmointikielellä ja iOS-alustalle Swift- tai Objective-C-ohjelmointikielellä. Android-käyttöliittymä on käytössä esimerkiksi Samsung- ja Huawei-mobiililaitteissa ja iOS-käyttöliittymä taas kaikissa Applen kehittämissä mobiililaitteissa. React Nativen avulla komponentit voidaan kirjoittaa molemmille alustoille JavaScript-ohjelmointikielellä. React Native luo vastaavat iOS- ja Android-komponentit ajon aikana. (React Native. 2021.)

React Native nopeuttaa monialustaisen sovelluksen kehitystä huomattavasti; se mahdollistaa samanaikaisen julkaisun molemmille iOS- ja Android-alustoille ja tarjoaa silti lähes samankaltaista suorituskykyä, kuin mitä on mahdollista saada, jos ohjelman ohjelmoi erikseen eri alustoille.

React Native -sovelluskehyksellä luotu sovellus voidaan julkaista myös verkkosivustona käyttämällä esimerkiksi React Native For Web -ohjelmakirjasto (React Native for Web. 2021). Jos sovellusta kehitetään samaan aikaan mobiilialustoille sekä verkkoselaimiin, täytyy heti alusta asti tämä ottaa huomioon sovelluksen käyttöliittymän tyylien ohjelmoimisessa.

2.7 Alustaeroavaisuudet

Sovelluksen toimiessa oletuksena monialustaisena jotkut ominaisuudet eivät ole luontevia käyttää kaikilla alustoilla, jos käyttäjä on tottunut alustakohtaisiin UI-kokemuksiin. Esimerkiksi iOS- ja Android-alustojen UI-ohjenuorat eroavat toisistaan jopa huomattavissa määrin.

Ongelmana on myös eri alustojen fyysiset tai käyttöjärjestelmäkohtaiset eroavaisuudet ja näiden huomioiminen monialustaisessa ohjelmassa, esimerkiksi iOS-laitteista puuttuva takaisin palaamisen painike. Docean-sovellus on kehitetty tarkemmin Android-alustan UI-ohjenuorien mukaan, Android-alustan käyttäjämäärän ollessa huomattavasti suurempi kuin muiden. Käyttäjän, joka on käyttänyt esimerkiksi iOS-alustaa useita vuosia voi olla tämän takia hitaampi tottua sovelluksen toimintoihin.

Opastustoiminnallisuuden täytyy ottaa huomioon käyttöliittymien eroavaisuudet alustakohtaisesti suurimman osan sovelluksesta ollessa käytössä samoilla ominaisuuksilla jokaisella alustalla hieman erilaisin käyttöliittymin.

3 Kehityskohteen esittely

Kehityksen kohteena on Docean-sovelluksen iOS- ja Android-alustoilla toimiva sovellus. Docean-sovellus on julkaistu iOS- ja Android-alustojen sovelluskauppoihin 2019 vuoden syksyllä. Sovellus toimii myös docean.fi verkkosivustolla samoilla ominaisuuksilla kuin mobiilisovellus. Verkkosivusto sisältää myös joitain lisäominaisuuksia, kuten käyttäjien hallinnan. Tämän työn kirjoittamisen aikana sovelluksella on latauksia sovelluskaupoista n. 300 kappaletta. Sovellus on käytettävissä vain sähköpostiosoitteella rekisteröityneille käyttäjille.

Docean-mobiilisovellus ja verkkosivusto on suunniteltu helposti käyttöönotettavaksi nykyaikaisia sovelluksia paljon käyttäneille. Sovellus käyttää paljon yleisesti käytössä olevia suunnitteluperiaatteita, muutamia poikkeuksia lukuun ottamatta.

Koska suurin kohderyhmä käyttää sovellusta rakennustyömailla erinäisissä olosuhteissa, lomakkeen täytön täytyy olla helppolukuinen ja nopeatoiminen. Sovellusta käytetään tällä hetkellä pakollisten lain vaatimien ja vapaaehtoisten esimerkiksi työn edistymisen esittävien dokumenttien laatimiseen.

Käyttäjälle on myös tarve ilmaista miten niin sanotut piilotetut ominaisuudet toimivat, jollei käyttäjä tutkimalla itse sovellusta näitä löydä tai edes tajua tehdä jotain tiettyä asiaa. Esimerkiksi selattavissa listoissa niiden kohtia voi painaa pohjassa, jolloin avautuu valikko minkä avulla kohtaa voi muokata tai poistaa sen kokonaan.

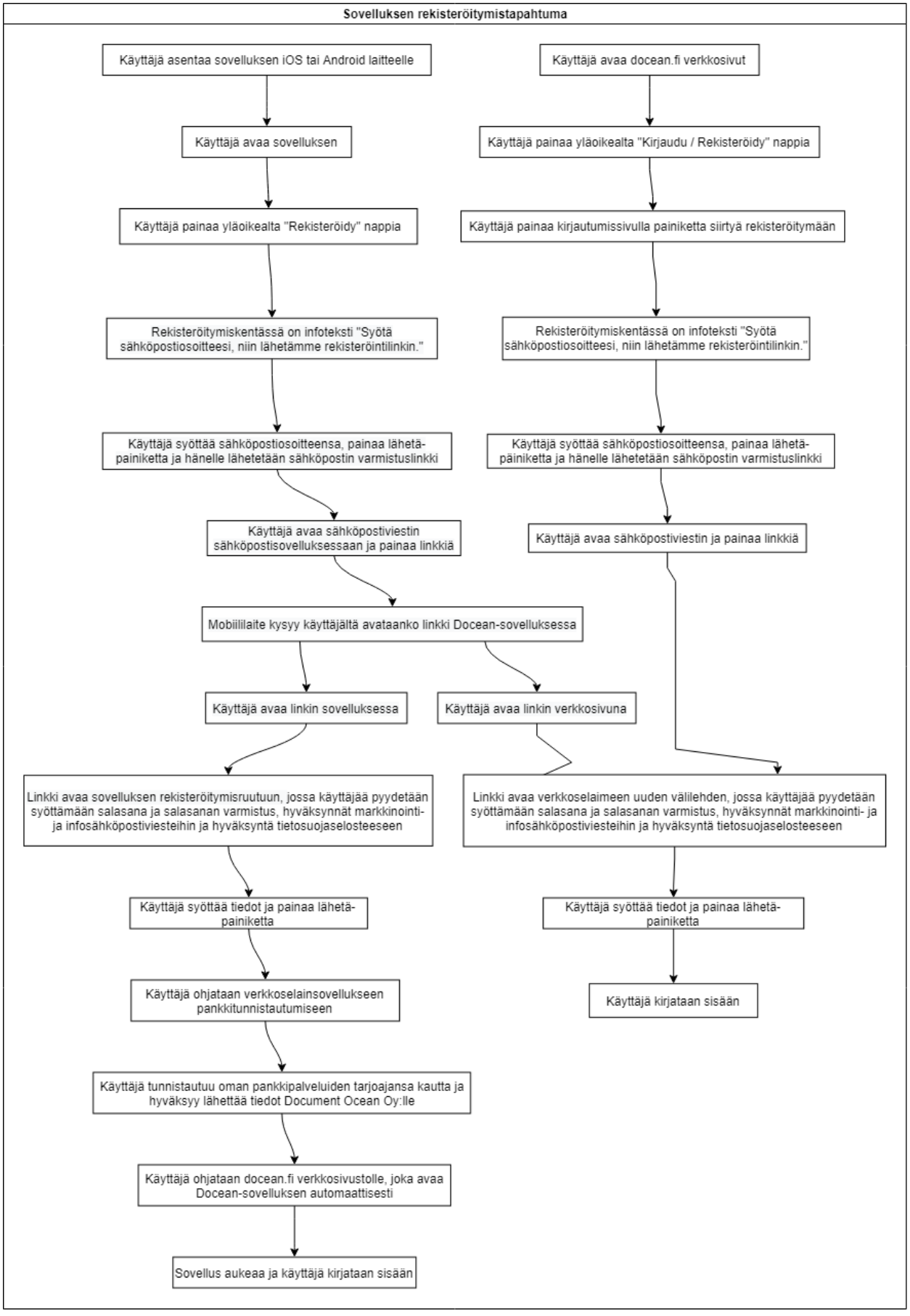
3.1 Tärkeimmät ominaisuudet

Edellä käydään läpi sovelluksen tärkeimpiä ominaisuuksia. Tässä työssä keskitytään vain sovelluksen mobiilidokumentointiosuuteen ja kaikkeen sen tarvitsemaan toiminnallisuuteen. Muut ominaisuudet, kuten käyttäjien hallinta ja projektiseurantatyökalu jätetään pois, jotta työn laajuus ei kasvaisi liian suureksi.

3.1.1 Rekisteröityminen ja kirjautuminen

Mobiilisovelluksessa yritysasiakkaan rekisteröitymistapahtuma (ks. kuvio 5) ja kirjautumistapahtuma poikkeaa valtavirran sovelluksista huomattavasti pankkitunnistautumisen takia. Pankkitunnistautuminen on arkipäivää nykyaikana, mutta varsinkin tunnistautumispalvelut, joiden tunnistautumistapahtumassa joudutaan avaamaan pankin tunnistautumissovellus, on jo tuottanut vaikeuksia käyttäjille. Käyttäjä joutuu valitsemaan tunnistautumispalvelun mobiililaitteen verkkoselaimessa, syöttämään käyttäjätunnuksensa, avaamaan tunnistautumissovelluksen, palaamaan takaisin verkkoselaimeen, antamaan suostumuksensa tietojen luovuttamiseksi Document Ocean Oy:lle ja vasta tämän jälkeen verkkoselain avaa automaattisesti Docean-sovelluksen.

Tunnistautumistapahtumassa on myös eroavaisuus Android- ja iOS-käyttöliittymien välillä. Käyttäjän pitää iOS-alustalla tietojen luovuttamisen hyväksymisen jälkeen valita ponnahdusikkunasta haluaako hän avata linkin Docean-sovelluksessa. Kirjautumistapahtuma ei onnistu, jos käyttäjä valitsee, että linkkiä ei saa avata Docean-sovelluksessa.



Kuvio 5. Docean-sovelluksen kirjautumistapahtuma mobiilisovelluksessa ja verkkoselaimessa.

3.1.2 Lomakkeiden luonti ja täyttäminen

Docean-sovellus käyttää sille varta vasten kehitettyä JSON-tietorakennetta (JavaScript Object Notation) lomakkeen täytön näkymän ja PDF-tiedoston luomiseen. JSON on universaalisti nykyaikaisten tukema kevyt tiedonsiirtoformaatti (Introducing JSON. N.d.). Formaatti on helppolukuinen niin ihmiselle kuin koneillekin (Introducing JSON. N.d.).

Lomakkeiden täyttämisen helpottamista varten sovellukseen on luotu lukuisia käyttöliittymäkomponentteja, esimerkiksi pudotusvalikko johon käyttäjä voi itse lisätä uusia vaihtoehtoja tai sähköpuolen urakoitsijoille räätälöity mittauksien syöttäminen ryhmäjohtotason pöytäkirjaan.

Jokainen erilainen käyttöliittymäkomponentti on tietorakenteessa tietyntyyppinen ”solu”. Tietorakenne määrittelee kuinka monta mitäkin käyttöliittymäkomponenttia lomakkeelle luodaan (ks. kuvio 6). Solut sisältävät määritellyn tyypin mukaan lisätietoa, jota tarvitaan komponentin luomiseen molempiin PDF-tiedostoon ja käyttöliittymään. Tietorakenteeseen määritellään myös alustavasti, kuinka monta allekirjoitusta lomakkeelle tarvitaan.

```
{
  row: 2,
  column: 1,
  valueKey: "6",
  width: 5,
  type: "textField",
  title: {
    font: "helveticaBold",
    text: { fin: "Keskustunnus:" },
  },
  direction: "row",
  borders: {
    top: { visible: false },
    left: { visible: false },
    bottom: { visible: true },
    right: { visible: false },
  },
},
```

Kuvio 6. Docean-sovelluksessa käytettävän JSON-tietorakenteen yksittäisen solun rakenne.

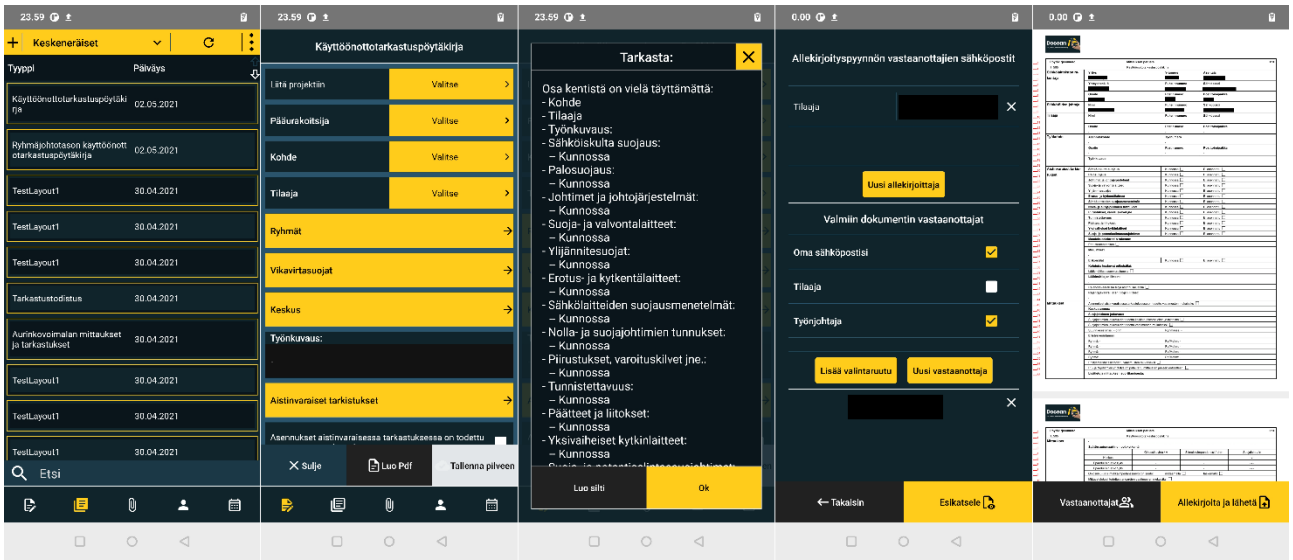
Lomakkeiden tietorakenteet, erillisliitteet ja lomakkeista luotu PDF-tiedosto arkistoidaan pilvipalvelimelle ja käyttäjä voi selata näitä kaikilla alustoilla. Lomakkeet on jaettu kategorioihin; keskeneräiset, valmiit, työntekijöiden ja vireillä olevat. Keskeneräiset-kategoria sisältää muokattavassa tilassa olevia lomakkeita ja muut kategoriat lukittuja lomakkeita, joita ei voi enää muokata, mutta

niistä voi luoda kopion tai revision, kopioida lomakkeen toiselle käyttäjälle keskeneräiseksi tai katsella niiden erillisliitteitä ja niistä luotua PDF-tiedostoa.

Vireillä olevat- kategoriassa käyttäjä näkee dokumentit, joiden lomake on täytetty valmiiksi ja se on lähetetty eteenpäin. Dokumentti on vireillä oleva, jos se on lähetetty allekirjoitettavaksi muille osapuolille, eikä allekirjoituksia ole vielä tehty loppuun tai jos valmiin dokumentin tai allekirjoituspyynnön lähetyksessä sähköpostilla on tapahtunut virhe. Sähköpostin lähetysvirheen takia käyttäjä voi pyytää vireillä olevan dokumentin uudelleenlähetystä sovelluksesta painamalla kyseistä dokumenttia.

Lomakkeiden täyttö toimii mobiilisovelluksella ilman verkkoyhteyttä reaaliaikaisen paikallistallennuksen ansiosta. Palvelimelle lomake tallentuu automaattisesti tietyin väliajoin tai lomakenäkymän painikkeella manuaalisesti.

Vastaanottajanäkymässä (ks. kuvio 7) käyttäjä voi valita valmiin dokumentin vastaanottajat ja myös allekirjoituspyyntöjen vastaanottajat, jos lomake on tarpeen allekirjoittaa muidenkin kuin tekijän.



Kuvio 7. Selausnäkymä, dokumentin täyttönäkymä, ilmoitus täyttämättömistä kohdista, vastaanottajanäkymä ja esikatselunäkymä Docean-sovelluksessa.

Vastaanottajanäkymässä on pikavalintana valintaruudut käyttäjän omalle, lomakkeen tilaajan ja käyttäjälle merkityn työnjohtajan sähköpostille. Pikavalintoja voi myös lisätä käyttäjäkohtaisesti uusia Lisää valintaruutu- painikkeella jos käyttäjä lähettää useasti samalle henkilölle vastaanotettavaksi lomakkeita.

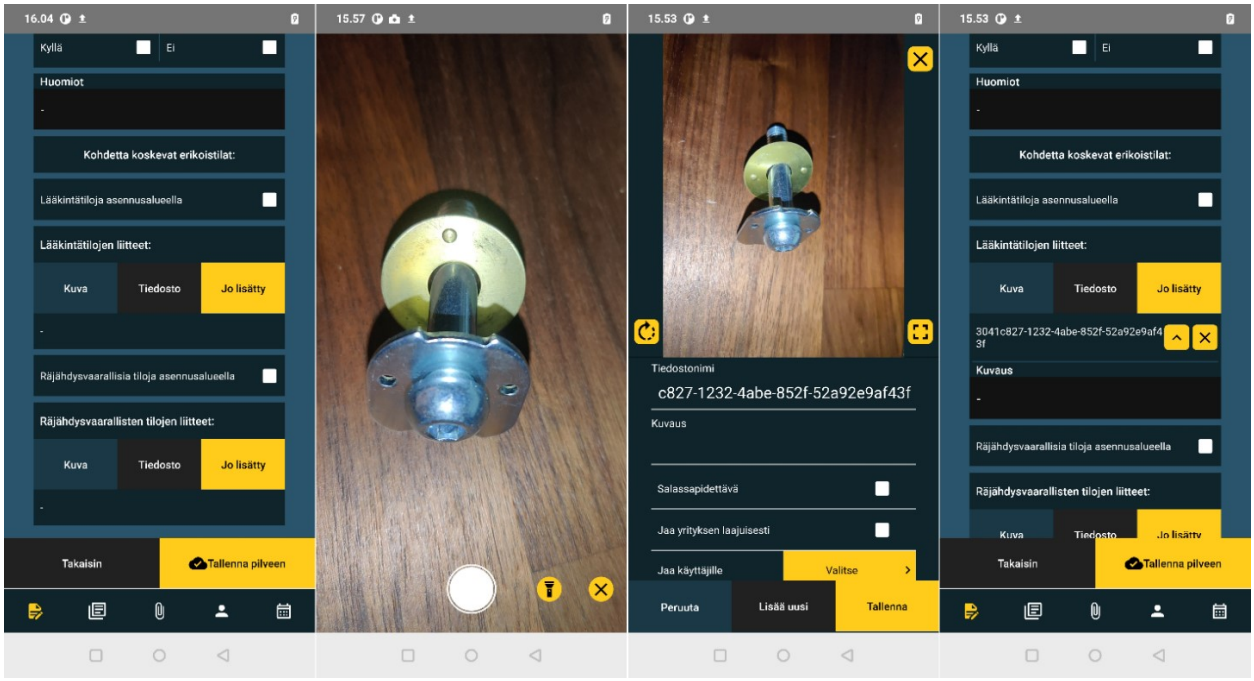
Esikatselunäkymässä käyttäjä voi selata lomakkeesta luotua PDF-tiedostoa ja siihen liitettyjä erillisliitteitä. Näkymän painikkeilla voi siirtyä Vastaanottajat-näkymään tai siirtää dokumentin valmiiksi, jolloin lomakkeesta riippuen lomake allekirjoitetaan digitaalisesti käyttäjän pankkitunnistautumistiedoilla tai vain lähetetään vastaanottajille.

Jos käyttäjä on lisännyt vastaanottajanäkymässä sähköposteja allekirjoittajille, heille lähetetään sähköpostilinkki allekirjoittaa lomake digitaalisesti, jolloin PDF-tiedosto lähetetään vastaanottajille ja allekirjoittajille vasta kun kaikki allekirjoittajat ovat allekirjoittaneet lomakkeen.

3.1.3 Liitteet

Lomakkeisiin voi liittää kaikentyyppisiä tiedostoja erillisliitteeksi. Kuvaliitteet voi lisätä joko pelkästään upotettavaksi lomakkeesta luotavaan PDF-tiedostoon tai ladata palvelimelle, jolloin kuvia voi käyttää uudestaan seuraavissa lomakkeissa.

Lomakkeen liitteen liittämisen käyttöliittymäkomponentissa on kaikilla alustoilla vaihtoehdot valita aikaisemmin lisätty liite tai tiedosto laitteen tiedostohakemistosta. Mobiililaitteilla on myös vaihtoehtona ottaa kuva Docean-sovelluksella ja liittää se dokumenttiin (ks. kuvio 8).

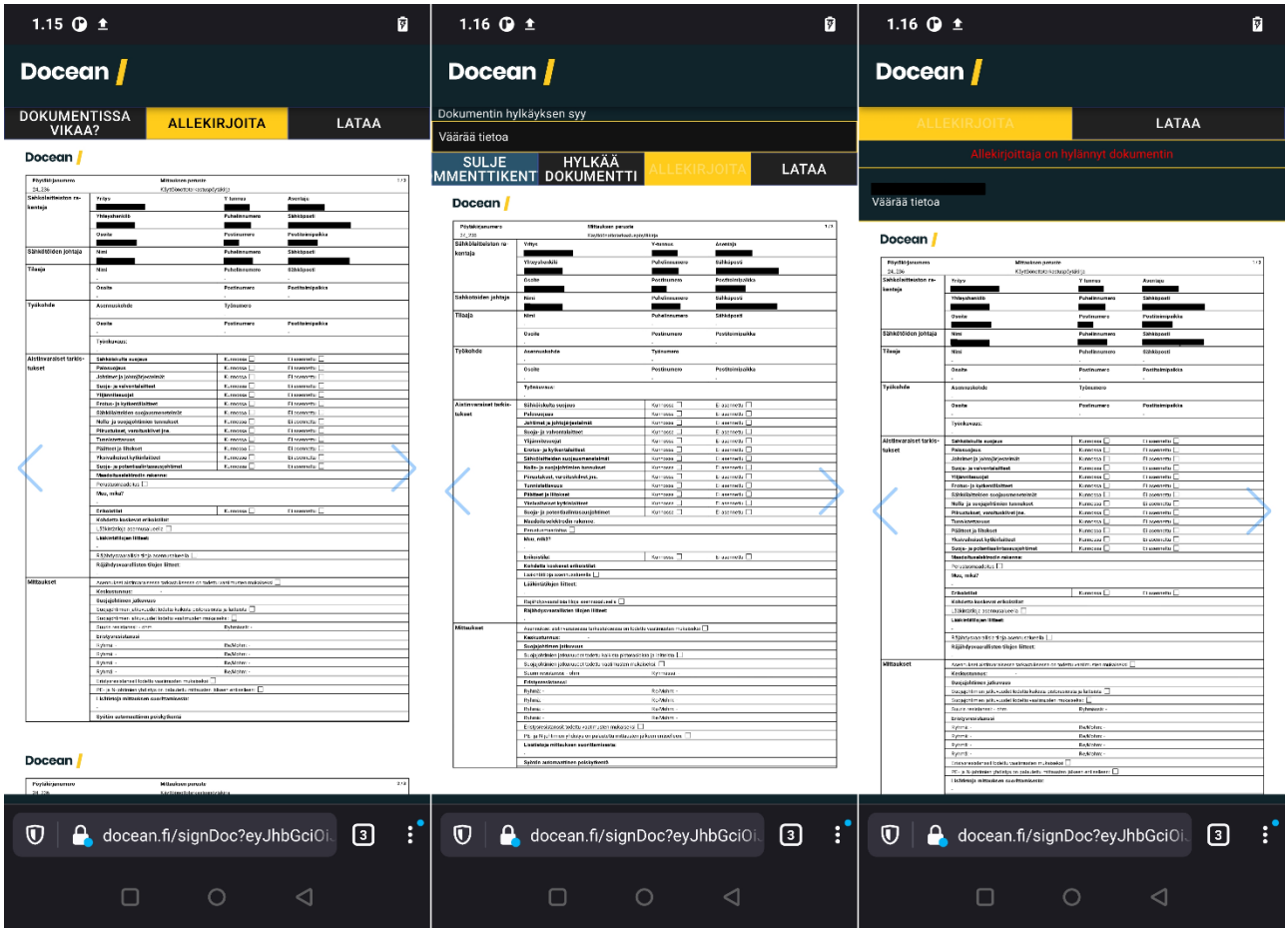


Kuvio 8. Dokumenttiin liitteen liittämisen prosessi Docean-sovelluksessa.

3.1.4 Digitaalinen allekirjoittaminen

Lomakkeesta luotua PDF-tiedosta käytetään hyväksi digitaaliseen allekirjoittamiseen. PDF-tiedostoon liitetään pankkitunnistautumisen avulla lainvoimainen kehittynyt digitaalinen allekirjoitus.

Allekirjoituspyynnön saaneet henkilöt pääsevät navigoimaan esikatselunäkymään, mikä sisältää normaalista esikatselunäkymästä poikkeavat painikkeet (ks. kuvio 9). Allekirjoittajat voivat hylätä allekirjoituspyynnön painamalla Dokumentissa vikaa-painiketta, syöttämällä syyn hylkäämiselle ja painamalla lopuksi Hylkää dokumentti- painiketta (ks. kuvio 9).



Kuvio 9. Allekirjoituspyynnön hylkäämisen prosessi allekirjoituslinkin avaamisen jälkeen, vasemmanpuolimmaisin näkymä näytetään käyttäjälle aina linkin avaamisen jälkeen.

Lomakkeen tekijälle lähetetään sähköposti sisältäen hylkäämisen syyn, hylkääjän sähköpostin ja lomakkeen tunnisteen. Lomakkeen tekijä näkee myös hylkäämisen tiedot sovelluksessa ja voi halutessaan ottaa yhteyttä hylkääjään ja/tai tehdä dokumentista kopion, korjata virheelliset tiedot ja lähettää dokumentin uudestaan allekirjoitettavaksi.

Allekirjoituslinkin saaneet näkevät hylkäämisen jälkeen allekirjoituslinkistä esikatselunäkymän, joka sisältää hylkääjien kommentit (ks. kuvio 9). Kommentteja voi lisätä, vaikka dokumentti on hylätty, jos allekirjoittajat löytävät muuta huomautettavaa. Dokumenttia ei voi enää allekirjoittaa, jos joku allekirjoittajista on sen hylännyt.

3.2 Vaatimuksien määritteleminen

Sovelluksesta saatujen tietojen ja tietoperustan avulla voidaan määritellä suunnittelussa ja toteutuksessa huomioon otettavia vaatimuksia. Toteutetun toiminnallisuuden onnistuneisuus on hyvä tarkistaa vielä vertaamalla tuotettua tulosta vaatimuksiin. Vaatimuksia voidaan määritellä viidessä eri luokassa; toiminnalliset vaatimukset liittyen yleisesti toiminnallisuuteen, käytettävyysvaatimukset liittyen toiminnallisuuden käytettävyyteen ja tekniset vaatimukset liittyen toiminnallisuuden tekniseen toteutustapaa. Vaatimuksille luodaan tunniste, jota voidaan käyttää teknisessä dokumentaatiossa ym. vastaavassa dokumentoinnissa. Toiminnallisille vaatimuksille käytetään etuliitettä functional_req_, käytettävyysvaatimuksille usability_req_ ja teknisille vaatimuksille tech_req_. Tunnisteet luodaan juoksevalla numeroinnilla niiden luokan mukaan etuliitteen kanssa.

Vaatimuksiksi voidaan määritellä teoriapohjassa esiin tulleen tiedon perusteella seuraavat:

- functional_req_1 opastuksen automaattinen avaaminen ensikertaa näkymään siirtyessä
- functional_req_2 opastuksen ohitus milloin tahansa
- functional_req_3 opastuksen käynnistäminen milloin tahansa
- functional_req_4 opastus koko näytön toimintoihin
- functional_req_5 opastus tiettyihin yksittäisiin toimintoihin.
- usability_req_1 normaalikokoisen tekstin esittämisen toteuttaminen vähintään 7:1 kontrastisuhteella (WCAG 2.1 1.4.6: 2008)
- usability_req_2 tekstin koon muuttaminen ilman avustavaa teknologiaa (WCAG 2.1 1.4.4: 2008)
- usability_req_3 välähdysten määrä enimmillään kolme kertaa sekunnissa (WCAG 2.1 2.3.2: 2008)
- usability_req_4 opastukseen liittyvien painikkeiden tarpeeksi suuri koko
- usability_req_5 opastuselementit on oltava erotettavissa normaalista toiminnallisuudesta
- tech_req_1 uusi toiminnallisuus on toteutettava uusina React-komponentteina
- tech_req_2 uudet React-komponentit on toteutettava React Hooks- ominaisuudella

Tech_req_1 toteutuneena tarkoittaa, että vihjetekstikomponenttia voi helposti uudelleenkäyttöää sovelluksen eri paikoissa. Tech_req_2 taas vaikuttaa lähinnä uuden toiminnallisuuden lähdekoodin kirjoitustapaan. Hooks on React version 16.8 uusi lisäys, joka mahdollistaa React-ominaisuuksien käytön ilman luokan kirjoittamista (Introducing Hooks. N.d.). Suurin osa Docean-sovelluksesta on kirjoitettu ilman luokkia. Toimeksiantaja siis asetti tämän vaatimuksen, jotta myös uudet komponentit on kirjoitettu samankaltaisesti kuin muutkin sovelluksen komponentit.

Kaikkien käytettävyysstandardin kriteerien täyttämiseen kuluisi huomattavasti enemmän resursseja, kuin tämän opinnäytetyön toteuttamiseen kannattaa käyttää. Tähän työhön kriteereistä valikoitui siis vain tärkeimmät ja kohtuullisen määrän aikaa vievät kriteerit. Valikoidut kriteerit liittyvät enimmäkseen visuaaliseen toteutukseen. Jos kuitenkin opastukseen löydetään suunnitteluvaiheessa tarvetta lisätä syötteitä, voi Document Ocean Oy toteuttaa syötteille kriteereitä vastaavat parannukset jatkokehityksessä.

Huononäköisillä käyttäjillä voi olla ongelmia nähdä tekstiä sen taustaväriltä. Tekstin ja sen taustavärin kontrastisuhteen toteuttaminen oikean suuruisena mahdollistaa sisällön näkyvyyden myös huononäköisille. Värinäön vaihtelut, kuten värisokeus, voivat myös vaikuttaa kontrastisuhteeseen ja tekstin luettavuuteen. 7:1 suuruinen kontrastisuhde on valittu korvaamaan käyttäjän värinäön puutteita ja kontrastinäön heikentymistä huononäköisillä käyttäjillä, jotka eivät käytä avustavaa sosiaalitekniikkaa. (Understanding Success Criterion 1.4.6: Contrast (Enhanced). N.d.)

Käyttäjille, joilla on lievä näkökyvyttömyys, tekstin koon muuttaminen voi olla kriittistä sovelluksen käytön mahdollistamiseksi. Yleensä selaimet vastaavat tästä toiminnallisuudesta, mutta jos kehittäjä käyttää selainalustoilla toimimatonta teknologiaa, tulee kehittäjän toteuttaa tekstin koon muuttamistoiminnallisuus itse. Kehittäjän pitää myös pitää huolta, että sisältö pysyy luettavissa, vaikka tekstin kokoa muokataan. Tekstiä pitää pystyä suurentamaan kaksinkertaiseksi alkuperäisestä leveydestään ja korkeudestaan. (Understanding Success Criterion 1.4.4: Resize text. N.d.)

Välähdysten määrää rajoittamalla vähennetään riskiä aiheuttaa käyttäjälle sairauskohtaus. Tämän avulla mahdollistetaan opastoiminnallisuuden käyttö myös käyttäjille, jotka sairastavat epilepsiaa tai muuta vastaavaa kohtauksia aiheuttavaa valonherkkyyteen liittyvää sairautta. (Understanding Success Criterion 2.3.2: Three Flashes. N.d.)

4 Suunnittelu

Suunnittelun tavoitteena on luoda mahdollisimman selkeä malli opastuksen ulkomuodosta. Tavoitteena on myös suunnitella opastuksen varsinainen toiminnallisuus ja mahdolliset lomakkeen tietorakenteeseen lisättävät opastuksen vaatimat muutokset. Opastuksesta ei luoda toimivaa prototyyppiä, vaan vain staattinen malli.

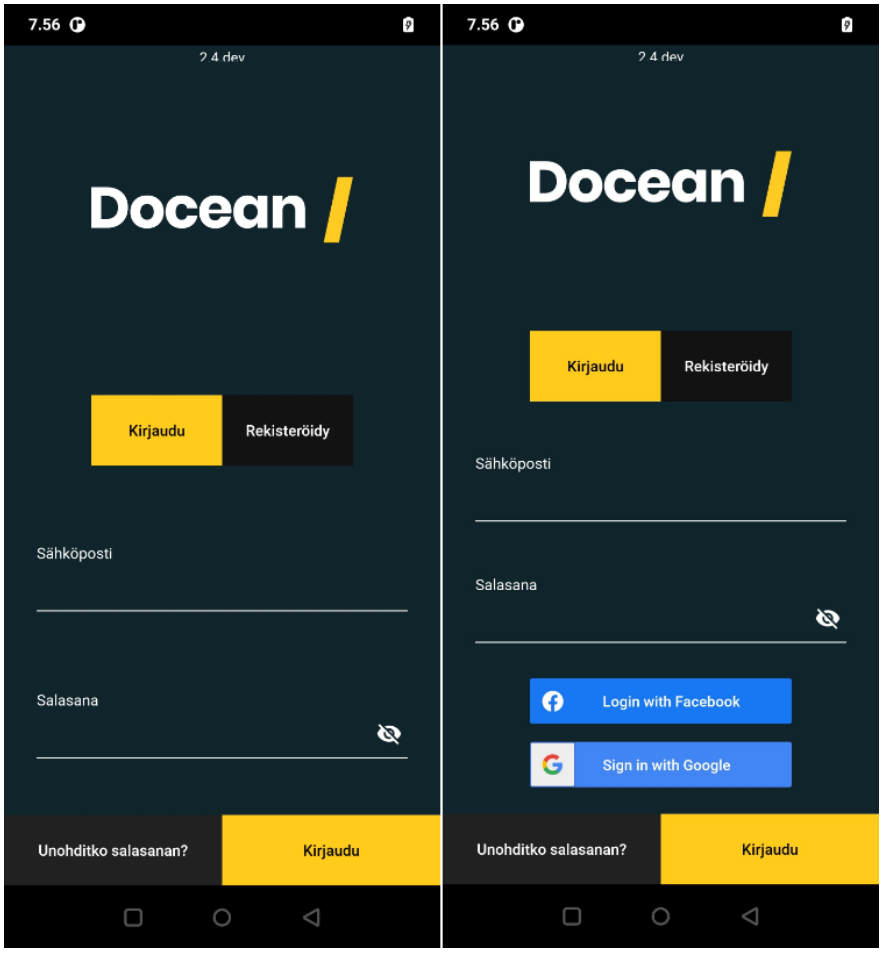
Mallin luomiseen käytetään Document Ocean Oy:llä muutenkin käytössä olevaa Adobe XD käyttöliittymäsuunnittelutyökalua. Työkalulla voidaan luoda yksinkertaisia staattisia rautalankamalleja käyttöliittymästä, mutta myös monimutkaisempien prototyyppien luominen onnistuu samalla työkalulla. Tietorakenteen muutokset ja mallista esille tuleva toiminnallisuus ja ulkomuoto hyväksytetään Document Ocean Oy:n edustajalla ennen toteutuksen aloittamista.

4.1 Ensimmäinen käyttökerta

Docean-mobiilisovelluksen perustoimintojen ollessa yksinkertaisia ja suhteellisen lähellä valtavirtaa, erillistä perehdytystä sovelluksen ensimmäisellä avauksella ei ole tarvetta tehdä. Ominaisuudet on esitelty sovelluskaupoissa, joiden informaation mahdollinen käyttäjä voi tarvitessaan selata läpi ennen sovelluksen asentamista ja vaikka mahdollinen käyttäjä asentaisi sovelluksen heti ilman lisätietojen lukemista sovelluskaupasta sen löytäessään, hän pääsee kokeilemaan sovelluksen ominaisuuksia heti nopean rekisteröitymisen jälkeen. Uusi käyttäjä voi rekisteröityä sovellukseen pelkän sähköpostiosoitteen avulla. Uutta käyttäjää pyydetään vain varmistamaan sähköpostiosoitteensa sinne lähetetyn linkin avulla ja syöttämään uusi salasana ja suostumus tietosuojaselosteeseen, jonka jälkeen hän pääsee jo tutustumaan sovellukseen.

Mahdollisimman nopeasti sovellusta kokeilemaan pääsemisen takia sovellukseen rekisteröitymistä olisi hyvä vielä keventää. Rekisteröitymisen keventämiseksi olisi hyvä lisätä sovellukseen kolmannen osapuolen kirjautumispalveluita, molemmille alustoille Facebook-kirjautuminen, Android-alustalle Google-kirjautuminen ja iOS-alustalle Apple-kirjautuminen (ks. kuvio 10). Käyttämällä kolmannen osapuolen kirjautumisvaihtoehtoja uuden käyttäjän ei tarvitse luoda uutta salasanaa Docean-palveluun, eikä varmistaa sähköpostiaan vaan hän voi rekisteröityä parhaimmillaan vain kolmella klikkauksella. Käyttäjän tarvitsisi esimerkiksi Google-kirjautumisen kautta vain hyväksyä Google-kirjautuminen ja Docean-palvelun tietosuojaseloste ja hän pääsee jo kokeilemaan sovellusta.

Kolmannen osapuolen kirjautumisen palveluja käyttämällä voidaan olla myös varmoja siitä, että käyttäjä osaa jo käyttää kyseisen palvelun käyttöliittymää ja näin Docean-sovellukseen ei tarvitse toteuttaa minkäänlaista opastusta. Tämä kuitenkin pätee vain koekäyttäjälle, joka ei joudu kirjautumaan pankkitunnistautumisen kautta.



Kuvio 10. Docean-sovelluksen nykyinen kirjautumisnäkymä ja malli näkymästä, johon lisätty kolmannen osapuolen kirjautumistavat.

Yritysasiakkaan käyttäjien sovellukseen kirjautuminen ja rekisteröityminen on digitaaliseen allekirjoitukseen käytettävän pankkitunnistautumisen takia paljon monimutkaisempi (ks. kuvio 5). Tunnistautumistapahtuman läpivientiä varten olisi hyvä lisätä ohjeistusta, mutta tunnistautuminen on ulkoinen OP-ryhmän tarjoama tunnistuksen välityspalvelu. Kaikki info täytyisi siis tarjota käyttäjälle ennen tunnistautumispalveluun siirtymistä.

Erilaisia tunnistautumispalvelun prosessin mahdollisuuksia on yhtä paljon, kuin Suomessa on pankkipalveluiden tarjoajia. Jotta käyttäjää voitaisiin ohjeistaa hyvin, informaatiota pitäisi tarjota jokaisesta erilaisesta tunnistautumisprosessista selkeästi ja kompaktisti. Tunnistautumisprosessit voivat myös muuttua huomattavasti riippuen pankkipalveluiden tarjoajin mielenjohteista. Näiden ongelmien takia päätettiin toimeksiantajan kanssa luottaa tunnistautumispalvelun tarjoajan hoitavan

käyttäjän ohjeistus tarpeeksi hyvin, jotta käyttäjä pystyy viemään tapahtuman onnistuneesti loppuun asti.

4.2 Sovelluksen toimintojen opastus

Erillisen opastuksen lisäämisen asemasta osa toiminnoista olisi parempi siirtää parempaan asiayhteyteen, esimerkiksi profiilinäkymän sivuvalikon Mittaukset ryhmittäin -valintaruutu, josta käyttäjä ei voi suoraan päätellä mitä sen painaminen vaikuttaa sovellukseen. Myös joidenkin painikkeiden tekstin vaihtamien kuvaavampaan tai joidenkin toimintojen visuaalisen ilmeen muuttaminen voi parantaa käyttäjän toimintojen ymmärtämistä, esimerkiksi selausnäkymän (ks. kuvio 7) oikean yläkulman asetusvalikon painikkeen kuvakkeen vaihtaminen kolmesta pisteestä rattaaseen, joka on yleisesti tunnetun asetusvalikon merkki.

Osan toiminnoista täytyisi toimia ilman erillistä opastusta; kirjautuminen, rekisteröityminen, unohtuneen salasanan vaihto ja digitaalinen allekirjoitus allekirjoituspyynnön linkistä. Nämä toiminnot ovat näkyvillä potentiaalisille käyttäjille ja jos ne ovat monimutkaisia käyttää, voi se vaikuttaa negatiivisesti Docean-sovelluksen maineeseen. Toimintojen pitäisi olla myös nykyisillekin käyttäjille nopeita käyttää, esimerkiksi unohtuneen salasanan palautuksessa käyttäjältä voi loppua kärsivällisyys, jos hän joutuisi vielä lukemaan opastusta. Jos siis näitä toimintoja jouduttaisiin opastamaan, ne on parempi toteuttaa täysin uudestaan tavalla, joka ei tarvitse opastusta. Nämä toiminnot kuitenkin ovat tällä hetkellä kirjautumisen ja rekisteröitymisen pankkitunnistautumista lukuun ottamatta helppoja ja nopeita käyttää, eikä opastusta toteuteta näille toiminnoille.

Mitä tulee opastoiminnallisuuden käynnistämiseen, helpoimpia vaihtoehtoja ovat opastusten näyttäminen käyttäjän siirtyessä näkymään ensimmäistä kertaa, tai opastuksen näyttäminen painikkeen avulla. Verkkoalustalla opastus voitaisiin avata käyttäjän siirtäessä hiiren kursorin toiminnon päälle, jolloin vihjeteksti tulisi näkyville eikä tällöin erillistä painiketta tarvittaisi, muuten kuin koko näkymän opastuksen käynnistämiseen. Mobiilisovelluksella kuitenkaan vastaavaa opastusta ei voi toteuttaa kiinteän näytöllä pysyvän kursorin puuttumisen takia ja opastuksen vapaaehtoinen käynnistäminen on pakko toteuttaa erillisen käyttöliittymäelementin avulla.

Vaatimuksina on opastuksen näyttäminen ensikertalaisille ja opastuksen käynnistäminen milloin vain, joten nämä molemmat pitää ottaa toteutuksessa huomioon. Staattisessa mallissa ensikertalaisille opastuksen näyttämistä ei oteta huomioon.

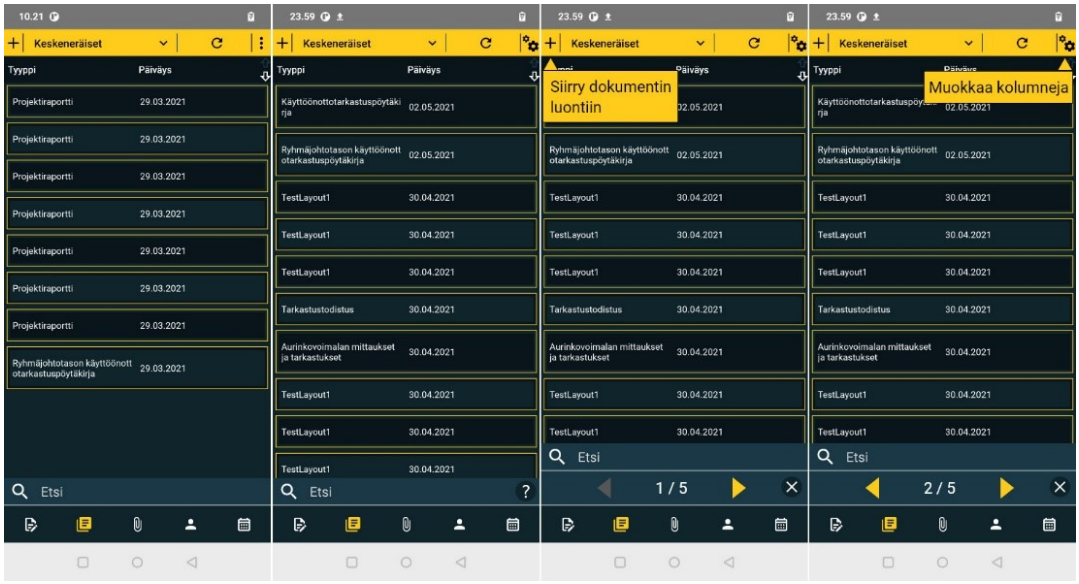
Sovelluksen seitsemästä näkymästä ainoastaan selausnäkymä sisältää niin paljon toimintoja, että koko ruudun toiminnot opastava toiminnallisuus olisi paras vaihtoehto. Selausnäkymä ei myöskään sisällä tilaa lisätä jokaiselle toiminnallisuudelle omaa painikettaan, jolla saa lisätietoa kyseisestä toiminnallisuudesta.

Muissa näkymissä suurin osa toiminnoista on itsestään selitteisiä, kuten perinteiset tekstikentät, mutta vaikeammin löydettävien tai käytettävien toimintojen yhteyteen on hyvä lisätä painike, jolla saa lisätietoa. Lisätietoa kaipaava toiminnallisuus voisi olla esimerkiksi Valitse vastaanottajat- näkymän Lisää uusi valintaruutu- painike (ks. kuvio 7).

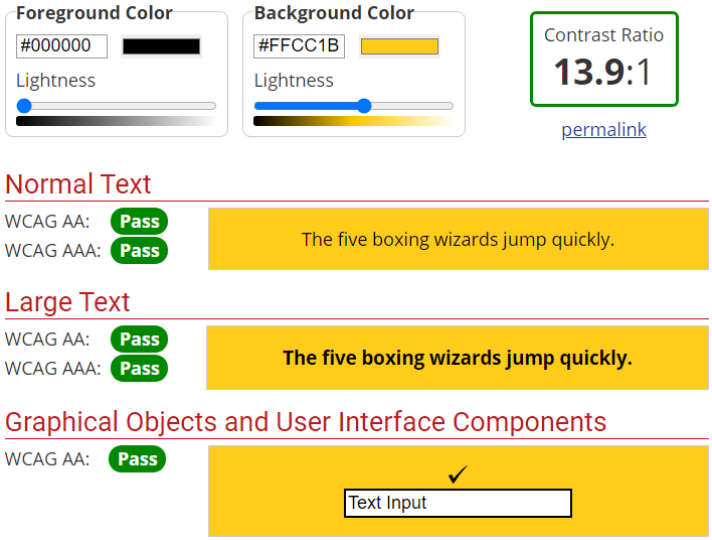
4.2.1 Kaikkien näkymän toimintojen opastuksen malli

Vihjetekstejä käyttävä opastus olisi todennäköisesti nopein toteuttaa sen yksinkertaisuuden takia ja sillä voidaan antaa käyttäjälle lisätietoa ilman huomattavaa häiriötä käyttökokemukseen. Vihjetekstit tulisi asettaa opastettavan toiminnan lähelle ja mahdollisesti vihjetekstin sisältävä laatikko voisi sisältää nuolen tai muun vastaavan elementin, joka osoittaa toimintoon.

Ensimmäisessä luonnoksessa (ks. kuvio 11) vihjetekstin laatikkoelementti on taustaväriltään sama kuin sovelluksessa muutenkin käytetty keltainen korostusväri #FFCC1B ja tekstin värinä musta #000000. Näiden värien kontrastisuhde (ks. kuvio 12) täyttää käytettävyysvaatimuksen usability_req_1.



Kuvio 11. Koko näkymän toimintojen opastuksen ensimmäinen malli.



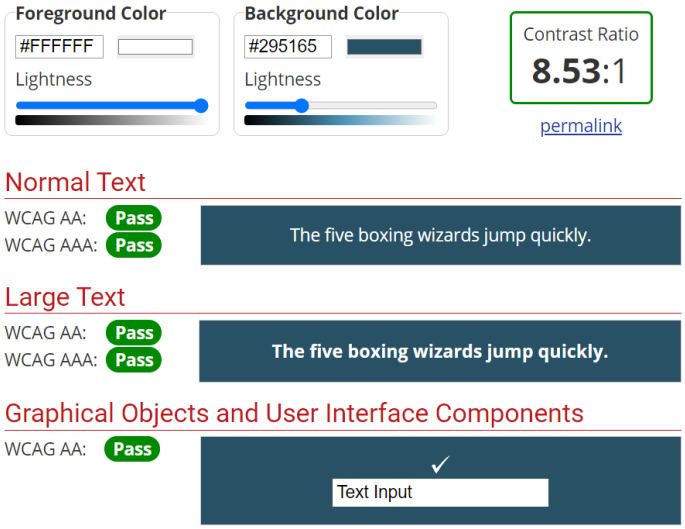
Kuvio 12. Docean-sovelluksessa käytetyn keltaisen korostusvärin ja mustan värin kontrastisuhde (Contrast Checker. 2021).

Luonnoksessa on lisätty selausnäkymän käyttöliittymän hakulaatikon oikealle puolella painike, jolla opastus voidaan käynnistää. Painike ei sisällä tekstiä, vaan kysymysmerkki-ikonin. Kysymysmerkki kuvastaa tunnetusti muissakin sovelluksissa toimintoa, jonka avulla käyttäjälle tarjotaan apua. Koko näkymän opastuksen ollessa esillä alareunan kiinteän navigointipalkin päällä voisi olla palkki,

joka indikoi, kuinka monta opastettavaa kohtaa on yhteensä ja monesko on tällä hetkellä näkyvillä. Palkin painikkeilla voisi siirtyä opastuksessa eteen- ja taaksepäin ja myös sulkea opastuksen.

Muuallakin sovelluksessa käytössä olevien värien käyttäminen vihjeikkunoissa ei kuitenkaan täytä vaatimusta usability_req_5. Opastukseen olisi hyvä käyttää sovelluksen värimaailmaan sopivaa, mutta siitä kuitenkin niin paljon eriävää, että käyttäjä tunnistaa käyttöliittymäelementin liittyvän opastukseen. Sovellus käyttää esimerkiksi joissain kohdissa onnistuneiden toimintojen ilmoittamiseen samanlaista käyttöliittymäelementtiä, kuin ensimmäisen luonnoksen vihjetekstielementti.

Toisessa luonnoksessa (ks. kuvio 13) keltaisen korostusvärin tilalle on vaihdettu sininen #295165, reunojen väriksi valkoinen ja myös paksunnettu reunoja. Näillä muutoksilla vihjelaatikot erottuvat selvästi muusta toiminnallisuudesta ja täyttävät silti vielä kontrastisuhteen vaatimuksen.

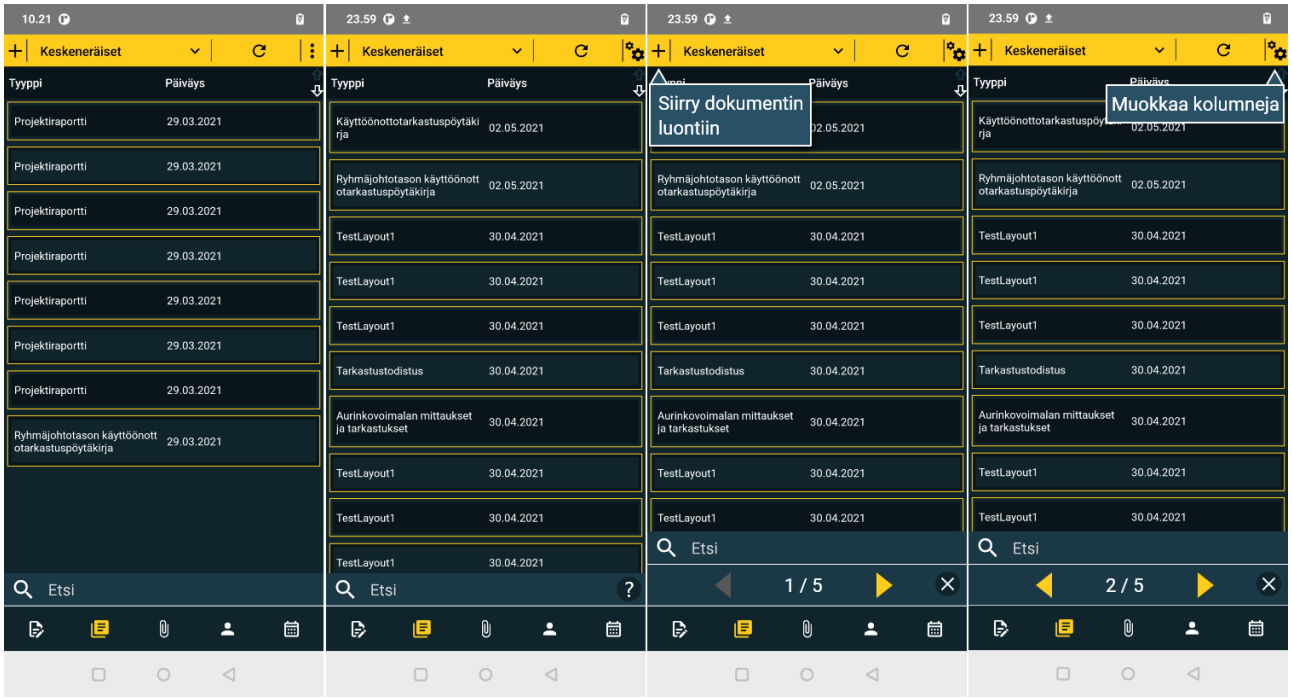


Kuvio 13. Vihjetekstin taustaväriksi keksityn sinisen korostusvärin ja valkoisen värin kontrastisuhde (Contrast Checker. 2021).

Opastuksen käyttöliittymäelementeille on muusta toiminallisuudesta erottuvuuden ja kontrastisuhteen lisäksi vaatimuksena painikkeiden tarpeeksi suuri koko. Tämä on vaatimuksena hieman epämääräinen, mutta tähän voidaan toteutuksessa käyttää jotain mobiilisovellukselle soveliasta ohjenuoraa.

Käytettävyysvaatimuksen usability_req_3 (välähdysten määrä enimmillään kolme kertaa sekunnissa) toteuttamiseen ei voida ottaa staattisella mallilla kantaa. Toteutuksessa täytyy vain ottaa huomioon, ettei opastuksen käynnistämisessä tai vihjeistä toiseen siirtymisessä tapahdu ylimääräisiä välähdyksiä. Vaatimukseen usability_req_2 (tekstin koon muuttaminen ilman avustavaa teknologiaa) ei toteuteta erikseen opastuksen yhteydessä mitään. Docean-sovelluksen asetuksissa tekstin koko voi muuttaa sovelluksen laajuisesti ja opastuksessa tullaan käyttämään tekstielementtejä, joiden koko muuttuu asetuksen avulla.

Mallin toisella iteraatiolla (ks. kuvio 14) on saatu täytettyä kaikki vaatimukset, jotka sillä voidaan esittää. Myös toimeksiantaja hyväksyi tämän mallin ilman muutoksia.



Kuvio 14. Koko näkymän toimintojen opastuksen toinen malli.

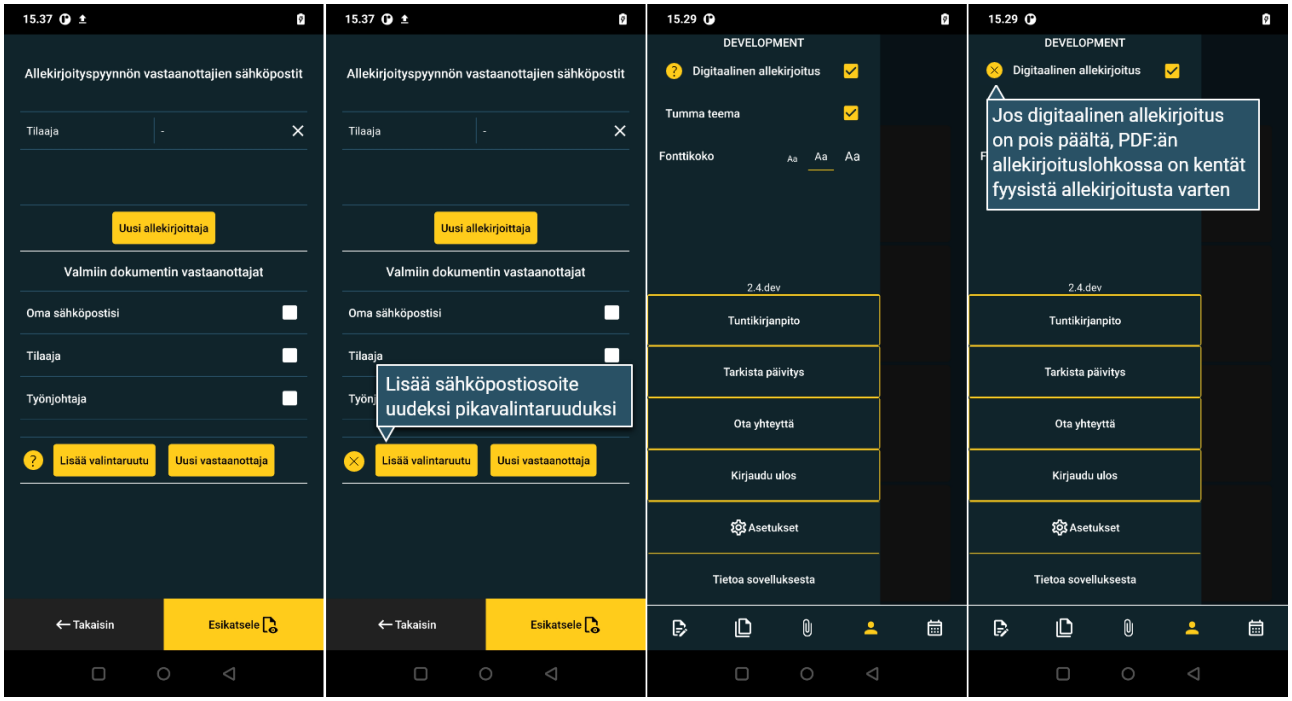
4.2.2 Yksittäisten toimintojen opastuksen malli

Yksittäisiä toiminnoille helpoin vaihtoehto toteuttaa on lisätä jonkin näköinen elementti toiminnon yhteyteen, joka tuo näytölle elementin, jossa on lisätietoa toiminnosta. Toisena vaihtoehtona olisi esimerkiksi pitkällä painalluksella toiminnosta opastuksen näytölle tuominen, mutta käyttäjä ei

välttämättä ikinä tajuaisi tätä ja pitkä painallus voi olla jo toiminnassa käytössä muulle asialle. Yksittäisenkin toiminnon opastuksen avaamiseen paras vaihtoehto on lisätä kysymysmerkkipainike toiminnallisuuden välittömään yhteyteen.

Ainoat sovellusta käyttäessäni havaitsemani opastusta kaipaavat toiminnot sovelluksessa ovat aikaisemmin jo mainittu Valitse vastaanottajat- näkymän Lisää uusi valintaruutu- painike ja profiilinäkymässä oleva Digitaalinen allekirjoitus- valintaruutu. Varsinkin Digitaalinen allekirjoitus- valintaruutu on sellainen, että käyttäjälle ei anneta minkäänlaista palautetta mitä on tapahtunut, kun sitä on painettu. Valintaruutu ei myöskään ole suoraan toiminnon yhteydessä, johon se vaikuttaa. Toiminnolla voidaan siis kontrolloida, allekirjoitetaanko lomakkeista luotuja PDF-tiedostoja digitaalisesti vai generoidaanko PDF-tiedostoon kentät fyysistä allekirjoitusta varten.

Vihjetekstielementti opastusta tarvitseville toiminnoille ja vihjetekstielementin avaava kysymysmerkkipainike olisivat samat, kuin koko näkymän opastuksen mallissa. Ainoat erot olisivat vihjetekstielementin sulkeminen myös sitä painamalla, kysymysmerkkipainikkeen elementtiin sopivat värit ja opastuksen tilan osoittamisen palkin puuttuminen alareunasta (ks. kuvio 15).



Kuvio 15. Yksittäisen toiminnon opastuksen ensimmäinen malli.

Yksittäisen toiminnon opastus on mallissa siis hyvin samankaltainen, kuin koko näkymän opastus. Näin käyttäjä erottaa heti, että näkymässä näkyvä elementti on vain opastusta varten. Mallia voidaan soveltaa kaikenlaisiin toimintoihin asettamalla kysymysmerkkipainike soveliaaseen kohtaan toiminnon käyttöliittymäelementtiin. Toimeksiantaja hyväksyi myös yksittäisen opastuksen mallin ilman muutoksia.

Osa yritysasiakkaista on esittänyt toivomuksen lisätä tiettyihin lomakkeen kenttiin opastusta lomakkeen täyttäjää varten. Tästä esimerkkinä sähköalan käyttöönottotarkastuspöytäkirjoista löytyvä kohta ”Kohteessa käytetty TN-S järjestelmää”. Lomaketta täyttävä sähköasentaja ei välttämättä muista mitä TN-S- järjestelmä tarkoittaa suoraan sen nimen perusteella, mutta pienellä ohjeistuksella voidaan muistuttaa asentajalle mitä kohdalla tarkoitetaan.

Tiettyjen lomakkeen kenttien opastusta ei voida luoda kiinteästi lähdekoodiin, koska lomakkeiden elementit luodaan täysin tietorakenteen määrittelemällä tavalla, ne on siis pakko lisätä jollain tavalla lomakkeiden tietorakenteeseen. Tietorakenteeseen voitaisiin lisätä tietoa, minkä perusteella vihjetekstielementti luodaan, jos sille on tarvetta. Tietorakenteeseen voitaisiin lisätä uusi ominaisuus nimellä hint. Hint voisi sisältää vihjetekstiä varten jo muuallakin tietorakenteessa käytettävän tekstiolion käännösominaisuuksillaan. Tekstiolio siis voi sisältää tietoa tekstin fontista, koosta, tekstin tasauksesta ja varsinaisen tekstin eri kielillä. Samaan hint olioon voi lisätä tulevaisuudessa muitakin tarvittavia tietoja, mutta tässä vaiheessa pelkkä tieto tekstistä riittää, jotta vihjetekstielementti voidaan luoda (ks. kuvio 16).

```
{
  row: 2,
  column: 1,
  valueKey: "6",
  width: 5,
  type: "textField",
  title: {
    font: "helveticaBold",
    text: { fin: "Keskustunnus:" },
  },
  direction: "row",
  borders: {
    top: { visible: false },
    left: { visible: false },
    bottom: { visible: true },
    right: { visible: false },
  },
},
```

```
{
  row: 2,
  column: 1,
  valueKey: "6",
  width: 5,
  type: "textField",
  title: {
    font: "helveticaBold",
    text: { fin: "Keskustunnus:" },
  },
  direction: "row",
  hint: {
    text: {
      fin: "Lorem ipsum dolor sit amet, consectetur adipiscing elit.",
    },
  },
  borders: {
    top: { visible: false },
    left: { visible: false },
    bottom: { visible: true },
    right: { visible: false },
  },
},
```

Kuvio 16. Tietorakenteeseen suunnitellut muutokset.

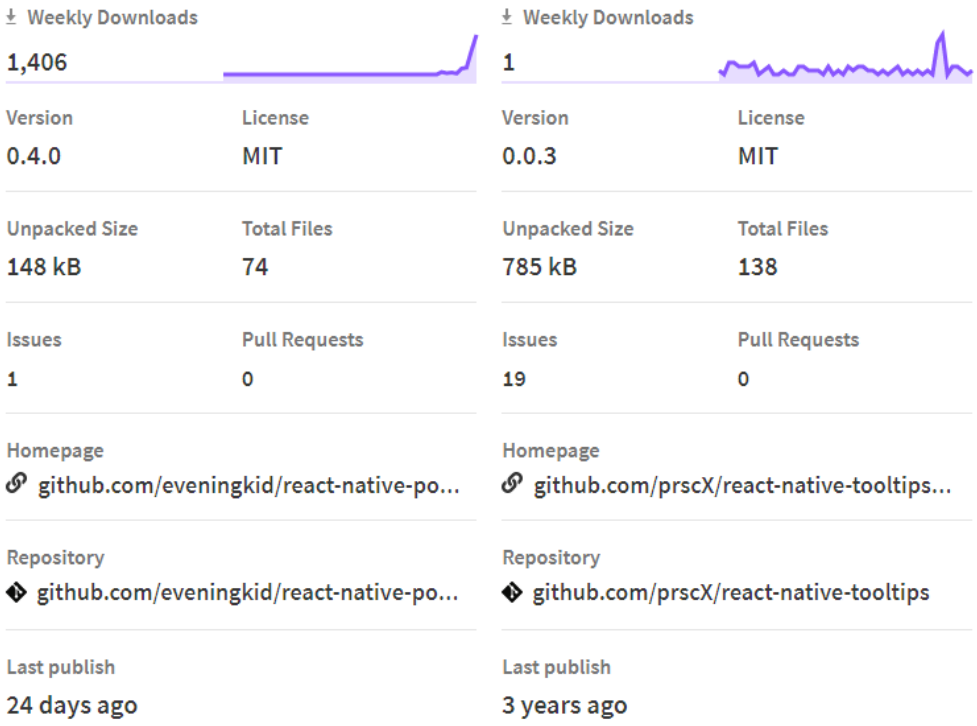
Tietorakenteeseen lisätyn tiedon perusteella voidaan lisätä nykyisen lomakkeen luonnin lähdekoodiin toiminnallisuutta, mikä luo lomakkeelle sen tarvitsemat vihjetekstit. Jos hint-ominaisuutta ei ole tietorakenteen solussa, niin sitä ei tarvitse luoda käyttöliittymään.

Mallien lisäksi toimeksiantaja hyväksyi tietorakenteeseen tehtävät lisäykset.

4.3 Avoimen lähdekoodin kirjastot

Suunnitteluvaiheessa kartoitetaan avoimen lähdekoodin kirjastot, joita voitaisiin käyttää nopeuttamaan opastuksen toteuttamista. Kirjastoja asennetaan React Native -sovellukseen npm JavaScript paketinhallintatyökalusta. Npm näyttää statistiikkaa paketeista paketin sivuilla (ks. kuvio 17) joista voidaan päätellä kannattaako kirjastoa harkita. Huomioon kannattaa yleensä ottaa kirjaston ylläpitäjien aktiivisuus, kirjasto voi olla täysin hylätty tai vielä kehitysvaiheessa. Tietenkin kirjasto voi olla täysin toimiva, vaikka se olisikin hylätty ja jos kirjaston lisenssi sallii lähdekoodin muokkaamisen ja kirjasto tarvitsisi vain joitain korjauksia, voi kirjasto olla varteenotettava vaihtoehto.

Aktiivisuutta voidaan tutkia statistiikasta ongelmien (issue) lukumäärän ja edellisen julkaisukerran (last publish) avulla (ks. kuvio 17). Jos ongelmia on kertynyt huomattava määrä ja edellisestä julkaisusta on kulunut aikaa, voidaan olettaa ylläpitäjien hylänneen kirjaston. Myös viikoittaisten latausten määrä on hyvä indikaattori siitä, onko kirjasto vielä suosittu vai ovatko kirjaston hyödyntäjätkin jo vaihtaneet toiseen kirjastoon tai tehneet oman.



Kuvio 17. React-native-popover ja react-native-tooltips avoimen lähdekoodin kirjastojen statistiikkaa (Popovers, tooltips for React Native. 2021; ReactNative: Native Tooltips (Android/iOS). Chauhan P. 2021).

Kirjaston koko (unpacked size) on myös erittäin tärkeä mobiilisovelluksen kannalta, kirjaston koko vaikuttaa suorasti sovelluskauppoihin lähetettävään sovelluksesta luotuun pakettiin. Sovelluskaupoilla on rajat, kuinka isoja sovellukset voivat olla, eikä kohtuuttoman suuria kirjastoja kannata käyttää, varsinkaan näin pieneen käyttötarkoitukseen. Npm-sivuston tarjoama purettu koko ei kuitenkaan suoraan kerro, kuinka paljon kirjasto vaikuttaa pakatun sovelluskauppoihin lähetettävän sovelluksen kokoon. Puretusta koosta voidaan kuitenkin päätellä, kuinka paljon kirjaston koko vaikuttaisi sovelluspakettiin.

Docean-sovellus käyttää jo react-native-elements kirjastoa, joka sisältää useita käyttöliittymäelementtejä, mukaan lukien vihjetekstikomponentin. Tämän kirjaston lisäksi vaihtoehtoja löytyi kaksi kappaletta; react-native-tooltips ja react-native-popable.

React-native-tooltips käyttää Android-alustalle tehtyä natiivikirjastoa ViewTooltip ja iOS-alustalle natiivikirjastoa SexyTooltip (ReactNative: Native Tooltips (Android/iOS). Chauhan P. 2021). Natiivikirjasto siis tarkoittaa esim. erikseen Android-alustan ohjelmointikielellä ohjelmoitua ja sille optimoitua kirjastoa. React-native-tooltips tarjoaa vain React Native- komponentin, joka käyttää alustan mukaan natiivikirjastoa generoimaan käyttöliittymäkomponentin. Hyvänä puolena tässä olisi natiivikirjastojen käyttäminen, jos natiivikirjastot olisivat luottettavan lähteen kirjoittamia. Tässä tapauksessa molemmat natiivikirjastot joudutaan myös tutkimaan ja varmistamaan niiden käyttökelpoisuus.

ViewTooltip sisälti 40 ratkaisematonta ongelmaa ja sen viimeisin julkaisu oli 31.1.2020 (ViewTooltip. Champigny F. 2020). ViewTooltip oli siis melko käyttökelvoton, varsinkin sen ollessa kirjoitettu Java-ohjelmointikielellä, jonka osaamista Document Ocean Oy:n työntekijät eivät tällä hetkellä omaa ja korjauksien tekeminen omakätisesti voisi viedä liikaa aikaa. React-native-tooltips ei ollut sovelias käyttöön.

React-native-popable taas oli suhteellisen uusi kirjasto, ensimmäiset tiedot sen viikottaisista latauksista oli viikolta 11.5.2020-17.5.2020. Viikoittaisia latauksia sillä oli viikolta 3.5.2021-9.5.2020 1406 kappaletta, kirjasto oli siis suhteellisen suosittu tämän työn kirjoitushetkellä. Kirjaston koko ei myöskään ollut suhteettoman suuri ja sen ongelmat oli korjattu nopeasti. Kirjoitushetkellä ratkaisemattomia ongelmia oli vain yksi kappale. Edellinen julkaisu kirjastosta oli 15.4.2021. (Popovers, tooltips for React Native. 2021.)

React-native-popable vaikutti lupaavalta kirjastolta npm-tilastojen perusteella. Seuraavaksi tarkisteltiin, toimiiko kirjaston käyttöliittymäelementti tarpeeksi hyvin. Kirjaston ylläpitäjä on tehnyt Expo-verkkosivulle demon kirjaston toiminnallisuudesta. Demon perusteella huomattiin vihje-elementin toimivan hyvin kaukana laitteen näytön reunoilta. Kuitenkin elementin, johon vihje-elementin pitäisi osoittaa, ollessa esimerkiksi näytön alalaidassa, vihje-elementti ei näkynyt kokonaan ja se tuli elementin päälle (ks. kuvio 18).

Kuvio 18. React-native-popable kirjaston demo. Vasemmalla muokkaamaton demo, oikealla ja keskellä elementin paikkaa siirretty (React-native-popable Sandbox. N.d.).

React-native-elements kirjaston vihje-elementissä ei tätä samaa vikaa ole. Huonona puolena on vihje-elementin ollessa näkyvillä muut toiminnot eivät toimi ennen vihjeen sulkemista. Kuitenkin react-native-elements kirjaston jo ollessa käytössä Docean-sovelluksessa on useita hyötyjä. Uutta kirjastoa ei tarvitse asentaa ja hallita tulevaisuudessa sen versioita. React-native-elements on myös erittäin suosittu kirjasto ja sitä päivitetään useasti.

Jos uutta kirjastoa ei asenneta, opastoiminnallisuuden päivitys voidaan tehdä Docean-sovelluksen sisältämällä CodePush-kirjastolla. Ilman CodePush-kirjaston käyttöä päivitys joudutaan julkaisemaan erikseen iOS- ja Android-sovelluskauppoihin ja odottamaan kauppojen hyväksyntää, ennen kuin käyttäjät voivat ladata päivityksen. CodePush-kirjaston avulla päivitys voidaan antaa käyttäjille ladattavaksi suoraan sovelluksesta käsin ilman ylimääräistä odottelua.

Näiden perustelujen takia toimeksiantaja antoi suostumuksen toteuttaa opastoiminnallisuuden react-native-elements tooltip-komponentin avulla.

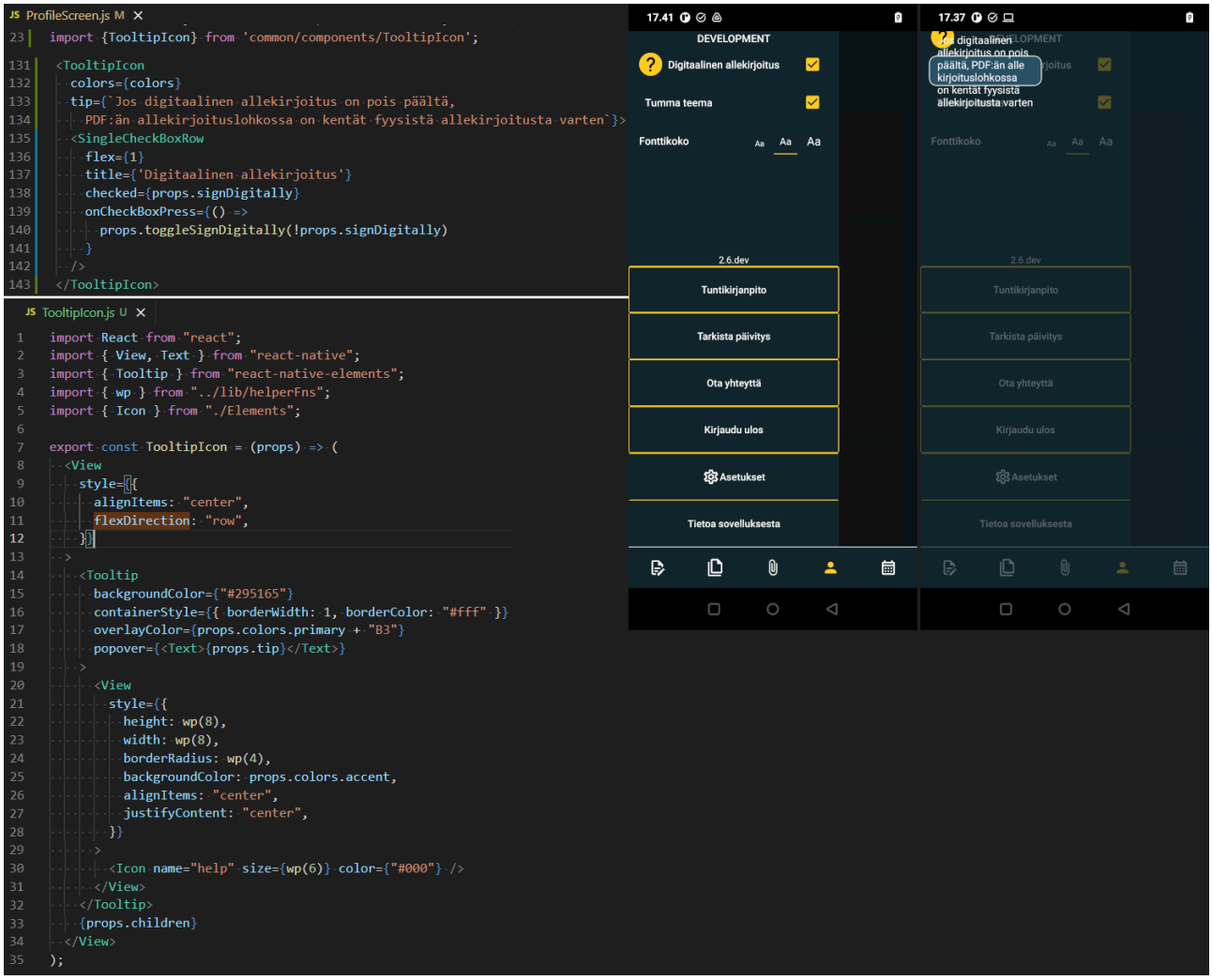
5 Toteutus

Toiminnallisuus toteutetaan JavaScript-ohjelmointikielellä React Native- ohjelmontikehyksellä Android- ja iOS-alustoille. Ohjelmointiin käytetään Visual Studio Code- tekstieditoria.

Toiminnallisuutta voidaan esikatsella paikallisesti käyttäen Android Studio- ohjelmointiympäristön Android-emulaattoria ja Xcode-ohjelmointiympäristön iOS-emulaattoria. Esikatselu onnistuu myös fyysisellä laitteella liittämällä laite USB-kaapelilla tai langattoman verkon kautta ohjelmointiympäristöihin.

Suurin osa Docean-sovelluksen käyttöliittymäelementeistä on jaoteltu yksittäisiin tiedostoihin, esimerkiksi painikkeelle oma tiedostonsa ja valintaruudulle omansa. Tätä tiedostorakennetta noudattaen myös vihjetekstielementille on hyvä luoda oma tiedostonsa.

Vihjetekstikomponentti olisi hyvä toteuttaa niin, että opastusta tarvitseville toiminnoille ei tarvitse tehdä mitään muutoksia. Tämä voitaisiin toteuttaa käyttämällä hyväksi React-kehyksen tarjoamaa children-toiminnallisuutta. Profiilinäkymän Digitaalinen allekirjoitus- valintaruutu on esimerkiksi toteutettu luomalla komponentti SingleCheckBoxRow ja asettamalla se profiilinäkymän käyttöliittymään koodilla: <SingleCheckBoxRow/>. React-kehyksen avulla valintaruudun ympärille voitaisiin asettaa vihjetekstikomponentti syntaksilla <TooltipIcon><SingleCheckBoxRow/></TooltipIcon>, jolloin SingleCheckBoxRow-komponenttiin ei tarvitsisi tehdä muutoksia. React tarjoaa tällä syntaksilla SingleCheckBoxRow-komponentin käytettäväksi TooltipIcon-komponentin sisälle props.children parametrilla (ks. kuvio 19).



Kuvio 19. React-native-elements-kirjaston Tooltip-komponenttia käyttävä toteutus lähdekoodissa ja visuaalisesti sovelluksessa.

Uusi komponentti yksittäisen toiminnon vihjetekstille toteutettiin tiedostoon TooltipIcon.js ja komponentti tuotiin tiedostoon ProfileScreen.js, jossa profiilinäkymä on toteutettu (ks. kuvio 19). Elementti voidaan sijoittaa nykyisen komponentin ympärille, jolloin komponentin vasemmalle puolelle luodaan kysymysmerkkipainike. Vihjetekstielementin luova kolmannen osapuolen kirjasto luo vihjetekstin, vihjetekstielementin sisälle asetetusta komponentista kopion ja sumentaa muun näytön. Vihjetekstin ollessa näkyvillä, kopio sisäkkäisestä komponentista siirtyy hieman ylöspäin (ks. kuvio 19), mutta tämä saatiin korjattua asettamalla vihjetekstikomponenttiin skipAndroidStatusBar-ominaisuus. Tämä ominaisuus pakottaa vihjeteksikirjaston olemaan ottamatta huomioon Android-käyttöliittymän tilapalkin korkeus elementin paikan laskennassa (Tooltip. N.d.).

Vihjetekstielementti ei myöskään ota huomioon pidemmän tekstin korkeutta. Tekstin korkeus jouduttaisiin laskemaan jollain tavalla ja asettamaan laskettu arvo vihjetekstielementin korkeudeksi. Kuitenkin ennen kuin tätä päästiin toteuttamaan, kirjaston toimintaa manuaalisesti testattaessa vihjetekstin avaaminen kaatoi aina sovelluksen. Virheilmoitukset näistä kaatumisista eivät antaneet mitään informaatiota siitä, mikä kaatumisia aiheutti. Myöskään React Native Elements- dokumentointiverkkosivuilla ei mainittu mitään vastaavasta ongelmasta. Vielä tässä vaiheessa, kun toteutukseen oli käytetty vain vähän aikaa, päätin vaihtaa pois React Native Elements- kirjaston tarjoaman komponentin käyttämisestä. Vaihtoehtoina oli ottaa käyttöön toinen avoimen lähdekoodin kirjasto tai tehdä vihjetekstikomponentin täysin itse. Itse tehty komponentti on paljon helpompi räätälöidä vastaamaan tarpeita ja myös sen kaikki aiheuttamat virheet on helpompi korjata, mutta sen tekemiseen voisi mennä enemmän aikaa, kuin uuden kirjaston sisällyttämiseen. Päätin kuitenkin tehdä komponentin itse, koska suunnittelussa tutkitut muut kirjastot eivät vaikuttaneet täysin toimivilta ja myös toimivan näköiset kirjastot voivat aiheuttaa suuria ongelmia.

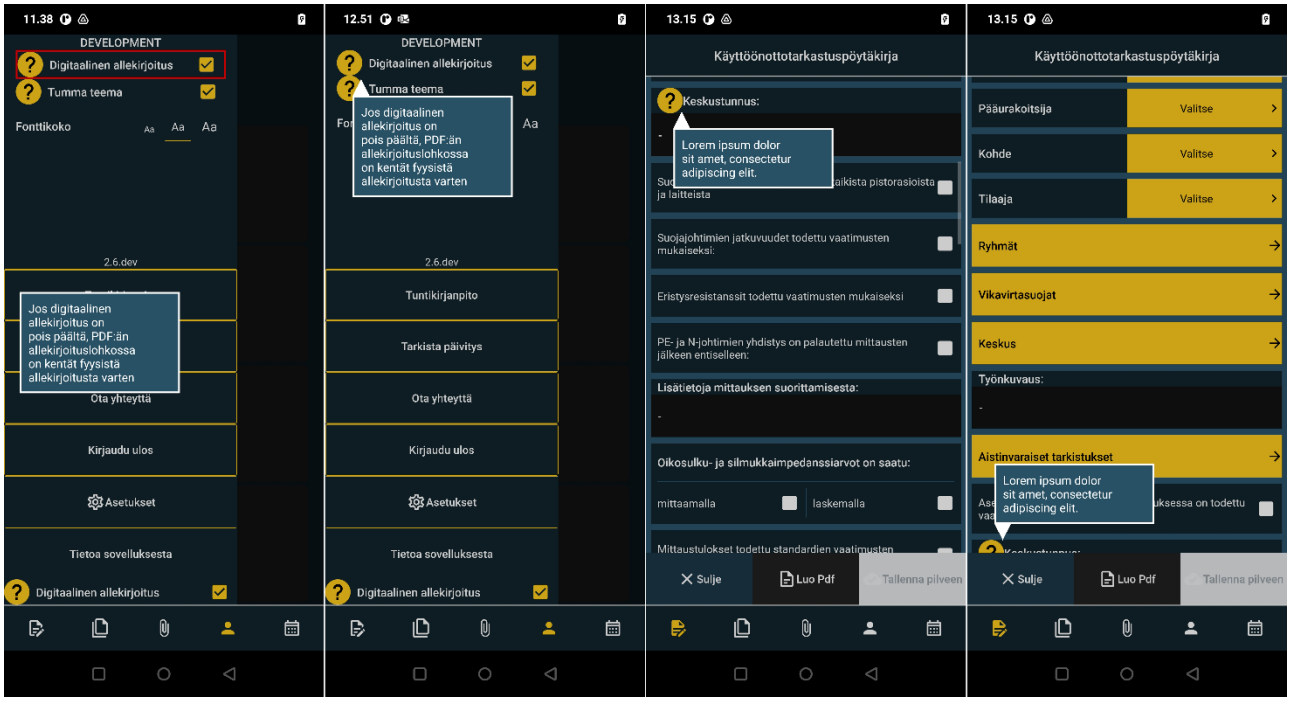
Uusi vihjetekstikomponentti toteutettiin uudeksi Tooltip-komponentiksi Tooltip.js tiedostoon (ks. liite 1). Pääkomponentin tiedoston pitämiseksi kompaktina luotiin toinen tiedosto TooltipModal.js (ks. liite 2) sisältämään esille tulevan vihjetekstielementin tyylit ja lähdekoodin. Komponentti kirjoitettiin käyttäen React Native- sovelluskehyksen tarjoamia komponentteja ja react-native-modal avoimen lähdekoodin kirjaston Modal-komponenttia. Modal-komponentin avulla vihjetekstilaatikko saatiin asetettua muun näkymän päälle.

Tooltip-komponentin sisällä lasketaan vihjetekstielementin sijainti näytöllä käyttäen React Nativen View-komponentin tarjoamaa measure-metodia. Measure ottaa selville pyydetyn View-komponentin korkeuden, leveyden ja sijainnin näytöllä ja palauttaa ne käytettäväksi (Direct Manipulation. 2021). Tällä tavalla saadaan selville vihjetekstiä tarvitsevan komponentin sijainti näytöllä ja voidaan asettaa vihjetekstielementti sopivaan kohtaan. Oletuksena vihjetekstielementti on vihjetekstiä tarvitsevan komponentin alapuolella ja komponenttiin osoittava kolmio vihjetekstielementin vasemmalla puolella elementin päällä. Vihjeteksti kasvaa oletuksena oikealle ja alaspäin.

Sijainnin laskemisessa on otettu huomioon näytön ala- ja oikeassa laidassa olevat elementit. Jos vihjetekstielementti ei mahdu kokonaan vihjetekstiä tarvitsevan komponentin alapuolelle, se ase-

tetaan komponentin yläpuolelle ja komponenttiin viittaava kolmio vihjetekstielementin yläpuolelle. Myös jos vihjetekstielementti ei mahdu kokonaan vihjetekstiä tarvitsevan komponentin oikealle puolelle, se asetetaan komponentin vasemmalle puolelle (ks. kuvio 20).

Yksittäisen komponentin toiminnan opastus (ks. kuvio 20) toteutettiin melkein täysin mallin kaltaiseksi (ks. kuvio 15). Tooltip-komponentin manuaalitestauksessa huomattiin React Nativen tarjoaman measure-metodin palauttavan välillä määrittelemättömiä arvoja. Varsinaista syytä tähän ei lähdekoodista löytynyt, joten vika oli todennäköisesti React Nativen measure-metodissa, johon korjauksia ei voi tehdä. Tämän takia komponenttiin lisättiin varmistukseksi logiikkaa, jos vihjetekstiä tarvitsevan komponentin oikeaa sijaintia ei saatu measure-metodilla. Varmistuslogiikan avulla vihjetekstielementti sijoitetaan näytön keskelle ja vihjetekstiä tarvitsevan komponentin ympärille asetetaan punaiset reunat (ks. kuvio 20).



Kuvio 20. Yksittäisen toiminnon opastuksen toteutuksen tulos.

Yksittäisen toiminnon opastusta varten itse toimintoon ei tarvitse tehdä mitään muutoksia ja näkymään, jonka sisällä toiminto on, tarvitsee vain lisätä toiminnon ympärille Tooltip-komponentti (ks. kuvio 21). Tässä yhteydessä Tooltip-komponentilla tarvii lähettää vain arvot tip, vihjetekstiele-

menttiin tuleva teksti ja icon, joka kertoo, että Tooltip-komponentti asettaa vihjetekstiä tarvitsevan toiminnon vasemmalle puolelle kysymysmerkkipainikkeen. Kysymysmerkkipainike avaa vihjetekstin ja mistä tahansa ruudulta painaminen sulkee sen. Tooltip-komponentti pitää itsensä sisällä tietoa, onko vihjeteksti näkyvillä vai ei.

```
<Tooltip tip={`Vaihda teema`} icon={true}>
  <SingleCheckBoxRow
    flex={1}
    title={'Tumma teema'}
    checked={
      !themeSelected || themeSelected === 'dark' ? true : false
    }
    onCheckBoxPress={() => {
      const newTheme =
        themeSelected === 'dark' ? 'light' : 'dark';
      props.toggleProfileTheme(newTheme);
      toggleTheme(newTheme);
    }}
  />
</Tooltip>
```

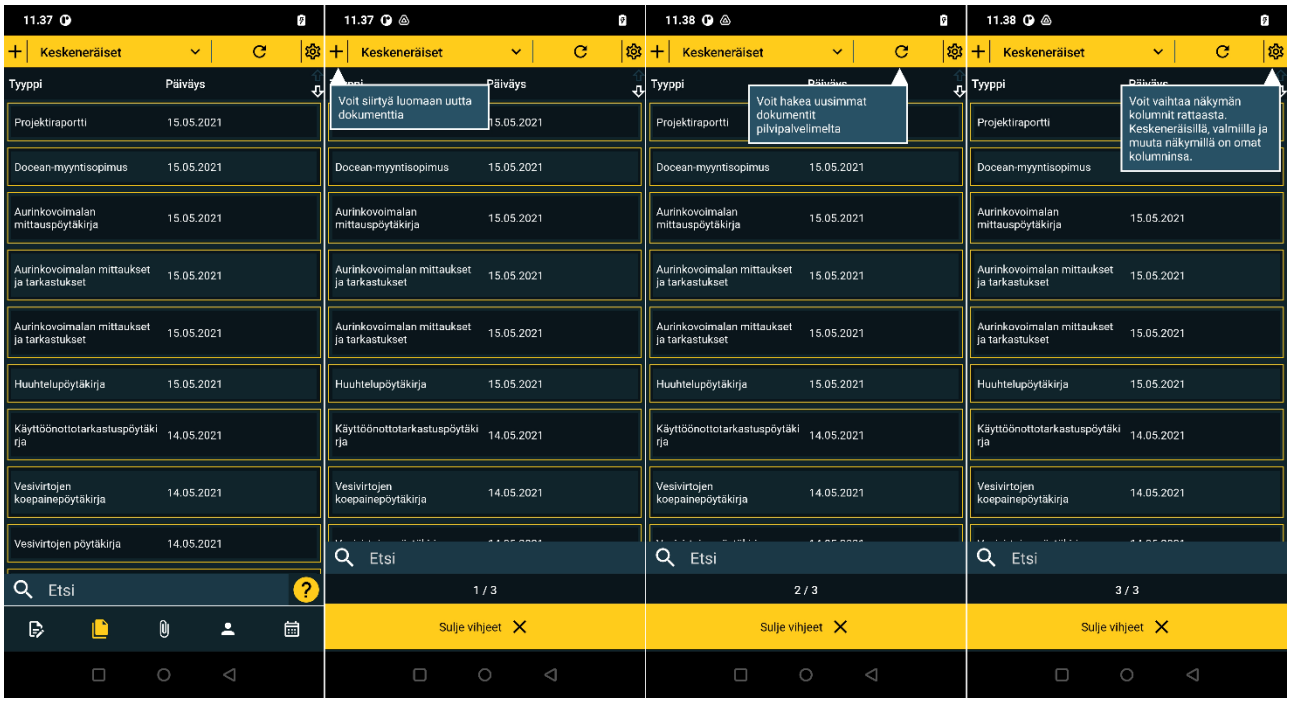
Kuvio 21. Opastusta tarvitsevan komponentin ympäröiminen lähdekoodissa Tooltip-komponentilla.

Tietorakenteesta luodun lomakkeen kenttien opastamiseksi tietorakenteeseen lisättiin suunnittelussa määritelty ominaisuus hint. Tätä apuna käyttäen lomakkeen lähdekoodiin erilaisille komponenteilla voidaan asettaa Tooltip-komponentti kentän ympärille. Jos kentälle on tietorakenteessa määritelty hint-ominaisuus, kysymysmerkki-ikoni on näkyvillä komponentissa. Jos hint-ominaisuutta ei ole määritelty, ikonia ei ole näkyvillä. Tämä on toteutettu Tooltip-komponentissa tarkistamalla, lähetetäänkö sille tip-arvo. Jos arvo lähetetään komponentille se palauttaa vihjetekstielementin ja sen sisälle asetetun komponentin ja jos arvoa ei lähetetä, se palauttaa vain sisälle asetetun komponentin. Koska melkein kaikki lomakkeen kentät ovat toisistaan eroavia, jokaiselle niiden komponenteille on asetettava Tooltip-komponentti sellaiseen kohtaa, jossa se ei aiheuta ongelmia lomakkeen täytölle tai visuaaliselle ilmeelle. Toimeksiannossa tämä toteutettiin vain textField-tyypin kentälle asettamalla kysymysmerkki-ikoni tekstikentän otsikon vasemmalle puolelle (ks. kuviot 20 ja 22).

```
<Tooltip tip={props.hint} flex={0} icon={true}>
  <Text
    style={[
      theme.boldText,
      { color: props.error ? "red" : colors.text },
    ]}
  >
    {props.error || props.title}
  </Text>
</Tooltip>
```

Kuvio 22. Lomakkeen tekstikenttäkomponentin ympäröinti Tooltip-komponentilla, lähettäen komponentille tietorakenteesta saadun hint-ominaisuuden.

Koko ruudun toiminnallisuuden opastus toteutettiin hyvin paljon sitä varten tehtyä mallia (ks. kuvio 14) vastaavaksi (ks. kuvio 23). Mallista eroten vihjeestä seuraavaan pääsee painamalla mistä tahansa Sulje vihjeet- painiketta lukuun ottamatta ja muutenkin vihjeiden ohituspainike on ulkomuodoltaan erilainen.



Kuvio 23. Koko ruudun opastuksen toteutuksen tulos.

Koko ruudun toiminnallisuuden opastamista varten näkymän, jota opastetaan, lähdekoodiin täytyy lisätä toiminnallisuutta (ks. kuvio 24). Toimeksiannossa toiminnallisuutta lisättiin vain selausnäkymään (ks. kuvio 23). Näkymän sisäiseen tilaan (state) täytyy lisätä näkyvillä olevan vihjetekstin numero helpNum jota muuttamalla voidaan asettaa näkyville seuraava vihje tai sulkea opastus. Näkymään lisättiin myös konstantti lastHelpNum, jonka avulla voidaan tarkistaa käyttäjän sulkiessa vihjeen, oliko se viimeinen vihje. Vihjeestä seuraavaan siirtymiseen luotiin uusi funktio continueHelpSequence, jota kutsutaan, kun käyttäjä sulkee vihjeen. Funktio kasvattaa helpNum muuttujaa yhdellä, jos nykyinen vihje ei ole viimeinen vihje ja jos vihje on viimeinen, funktio asettaa muuttujan arvoon null, joka sulkee koko opastuksen.

```
const lastHelpNum = 3;
const initialBrowseState = {
  helpNum: null,
const continueHelpSequence = () => {
  if (state.helpNum === lastHelpNum) {
    setState({
      helpNum: null,
    });
  } else {
    setState({
      helpNum: state.helpNum + 1,
    });
  }
};

const skipHelp = () => setState({ helpNum: null });
```

Kuvio 24. Selausnäkymään tehdyt Tooltip-komponentin vaatimat muutokset.

Koko näkymän toimintojen opastusta varten komponenttien, jotka tarvitsevat opastusta, ympärille pitää lisätä Tooltip-komponentti (ks. kuvio 25). Komponentille pitää lähettää arvot tip, flex, isVisible, hideModal ja skip. Tip-arvo on vihjetekstielementin sisälle tuleva teksti. Flex-arvoa tarvitaan pitämään käyttöliittymän näkymä samanlaisena, kuin ennenkin. Arvo isVisible määrää onko vihjetekstielementti näkyvillä. Arvon hideModal-arvon pitää olla funktio, jota kutsutaan, kun käyttäjä painaa mistä tahansa ruudulta, paitsi Sulje vihjeet- painikkeesta. Skip-arvoksi Tooltip-komponentti tarvitsee funktion, jota kutsutaan käyttäjän painaessa Sulje vihjeet- painiketta.

```
<Tooltip
  tip="Voit siirtyä luomaan uutta dokumenttia"
  flex={0}
  isVisible={state.helpNum === 1}
  hideModal={continueHelpSequence}
  skip={skipHelp}
>
  <Icon name={"plus"} size={wp(7)} color={"#000"} />
</Tooltip>
```

Kuvio 25. Koko näkymän opastuksen Tooltip-komponentti lähdekoodissa ja sen tarvitsemat arvot.

6 Pohdinta

Työn tavoitteena oli luoda opastoiminnallisuutta auttamaan uutta käyttäjää pääsemään nopeasti sisälle sovelluksen toimintoihin ja auttaa myös nykyisiä käyttäjiä, jotka eivät välttämättä ole olleet tietoisia sovelluksen tietyistä toiminnoista. Tavoitteen motivaationa oli saada käyttäjälle tarjottua tarpeeksi informaatiota sovelluksesta, jotta hän jatkaisi sovelluksen käyttöä. Toinen motivaatio oli saada vähennettyä tai kokonaan poistettua tarve pitää käyttökoulutuksia asiakasyrityksille.

Työn aiheeseen liittyen löydettiin, jopa kvantitatiivista tutkimustietoa, jota en odottanut löytyväksi. Tutkimustiedolla saatiin hyvää lisätietoa tietynlaisen opastuksen tarpeellisuudesta ja käyttäjäkokemuksesta. Tämä tutkimustieto auttoi opastuksen kohdistamisessa oikeisiin paikkoihin ja tietynlaisiin toimintoihin.

Tavoitteeseen päästiin hyvin käyttämällä apuna kerättyä tietoa, mutta kuitenkin toteuttamalla uudenlainen opastustoiminnallisuus onnistuneesti Docean-sovellukseen. Toimeksiantaja sai työn tuloksena käyttöönsä opastusta varten uuden käyttöliittymäkomponentin, jonka voi lisätä pienellä vaivalla mihin vain opastusta tarvitsevaan osaan sovelluksen lähdekoodissa. Työssä toteutettiin opastus vain osittain lopullisessa muodossaan. Kaikkea oikeaa vihjetekstiä ei lisätty, vaan joihinkin kohtiin jätettiin väliaikaiset tekstit. Jokaiselle lomakkeen komponentille ei lisätty vihjetekstielementtiä. Toimeksiantaja voi kuitenkin lisätä haluamansa vihjetekstit helposti toteutetun vihjetekstikomponentin avulla niihin kohtiin, joissa opastusta tarvitaan.

6.1 Toteutuneet vaatimukset

Toiminnallisista vaatimuksista toteutettiin melkein kaikki vaatimukset, opastus voidaan lisätä koko näkymälle tai yksittäiselle toiminnolle ja opastuksen voi ohittaa ja käynnistää milloin tahansa. Functional_req_1 jätettiin toteuttamatta, käyttäjä voi kuitenkin halutessaan käynnistää opastuksen milloin tahansa, joten opastuksen automaattinen käynnistäminen näkymään ensikertaa siirtyessä ei välttämättä ole pakollinen toteuttaa tulevaisuudessakaan.

Käytettävyysvaatimuksista toteutuivat kokonaan vaatimukset usability_req_1, 3, 4, ja 5. Usability_req_4 toteutettiin suhteuttamalla uuden komponentin painikkeet muuhun käyttöliittymään. Usability_req_2 toteutuu osittain sovelluksen profiilivalikon fonttikoon asetuksella, vihjetekstikomponenttiin täytyy vain lisätä jatkokehityksenä tyyli, joka vaihtuu valikon asetuksen avulla. Fonttikokoa ei voi muuttaa vaadittuun 200-prosenttiin asti, mutta kokoa voi kuitenkin muuttaa.

Tekniset vaatimukset toteutuivat kokonaan ilman poikkeuksia. Molemmat yksittäisen toiminnon ja koko ruudun toimintojen opastuksen vihjetekstielementti on luotu Tooltip-komponentiksi sovelluksen lähdekoodiin. Uusi komponentti on myös luotu käyttäen React Hooks- ominaisuuksia.

6.2 Ongelmakohdat

Opastuksen käytettävyyden määrittelyyn olisi ollut parempi käyttää mobiilisovelluksia varten luotua standardia, mutta Android-käyttöliittymän ohjenuorien ollessa Googlen laatimat ja iOS-ohjenuorien Applen laatimat, päätettiin käyttää W3C-organisaation web-kehitykseen luotua standardia. W3C-organisaatio voitaisiin luokitella luotettavammaksi lähteeksi, kuin voittoa tavoittelevat yritykset. Web-käyttöliittymille luotua standardia ei kuitenkaan voitu täysin käyttää hyödyksi mobiilisovellusta suunnitellessa, valitun standardin lisäksi olisi voinut käyttää jotain mobiilikäyttöliittymästandardia. Tällä olisi saatu lisää käytettävyysparannuksia opastoiminnallisuudelle.

Yleensä käyttöliittymätoteutuksissa ja myös esimerkiksi palvelintoteutuksessa tutkitaan ensin, onko korjattavaan ongelmaan jo olemassa ratkaisua tai avoimen lähdekoodin toteutusta ennen kuin ongelmaa lähdetään korjaamaan täysin alusta asti itse, pyörää ei siis tarvitse keksiä aina uudestaan. Tässä työssä toteutus aloitettiin myös etsimällä ensin kolmannen osapuolen kirjastoa,

joka hoitaisi vihjetekstin luomisen. Kirjastoja löydettiinkin ja niistä valittiin paras, mutta sen epävarman toiminnan vuoksi se hylättiin ja luotiin komponentti itse täysin alusta asti. Tämä vaikutti työn valmistumisen ajanhetkeen haitallisesti, mutta tuloksesta saatiin paremmin vaatimuksia vastaava ja tulevaisuudessa helpommin toimeksiantajan jatkokehitettävä.

Vihjetekstikomponentille ei luotu automaatiotestejä, se vain testattiin manuaalisesti kaikilla keksityillä sijainneilla. Voi siis olla, että vihjetekstikomponentti laskee väärin vihjetekstielementin sijainnin, jos sen opastusta tarvitseva komponentti on tietyllä kohdalla näyttöä. Pahimmassa tapauksessa kuitenkin vihjeteksti vain menee yli näytön rajojen, tämä ei siis aiheuta pahinta mahdollista lopputulosta, sovelluksen kaatumista.

Uutta komponenttia ei myöskään testattu kuin yhdellä laitteella molemmilla iOS- ja Android-alustoilla. Koska uusi komponentti tehtiin käyttäen React Nativen tarjoamia komponentteja, voidaan kuitenkin olettaa komponentin toimivan jokaisella laitteella. Samoja uuden komponentin käyttämiä React Nativen komponentteja on käytössä muuallakin sovelluksessa ja sovellusta on testattu usealla laitteella. Sama pätee uudessa komponentissa käytettyyn react-native-modal kirjaston Modal-komponenttiin. Oletuksena siis vihjetekstikomponentin tulisi toimia jokaisella laitteella, jota sovellus tukee, mutta tätä ei voida sanoa täydellä varmuudella.

Vihjetekstielementin sijainnin laskemiseen käytettävän measure-metodin palautumattomien arvojen ongelman tutkimiseen ei käytetty sen korjaamiseen vaadittua aikaa. Manuaalitestauksessa tätä tapahtui lähinnä profiilinäkymän sivuvalikon toimintojen vihjetekstielementeillä. Ongelmana olisi voinut olla sivuvalikon ominaisuus olla piilotettuna siihen asti, että käyttäjä avaa sen pyyhkäisemällä tai muulla tavalla. Sivuvalikon sisältämät komponentit kuitenkin ovat niin sanotusti luotu käyttöliittymään ja ne ovat vain näkymättömissä ennen valikon avaamista, varmuudella ei siis voida sanoa johtuiko ongelma sivuvalikosta. Tämän ongelman tutkimiseen ja korjaamiseen jouduttaisiin käyttämään enemmän aikaa, mutta sen ollessa ei-kriittinen ongelma vihjetekstielementtiin toteutetun varmistuslogiikan takia, ongelmaan päätettiin olla perehtymättä liikaa. React Native Elements- kirjaston Tooltip-komponentti mahdollisesti kaatoi sovelluksen tämän ongelman takia, jos tätä ei ollut huomioitu sen lähdekoodissa. Ennen varmistuslogiikan lisäämistä myös uutena luotu komponentti kaatoi sovelluksen samalla tavalla, kuin kirjaston komponentti.

6.3 Jatkokehittäminen

Toteutuksen tulokselle olisi hyvä luoda automaatiotestit. Automaatiotesteillä voitaisiin varmistaa opastuksen toimivuus myös tulevaisuuden päivityksissä ja useilla eri valmistajien laitteilla. Esimerkiksi Microsoft App Center Test- testiautomaatiopalvelulla voidaan testata sovelluksia sadoilla uniikeilla laitemalleilla (App Center Test. Cenovsky L., Gimm V., Green K., Renshaw T., Wester G., Wilson G. 2020). Docean-sovelluksen käyttöliittymän automaatiotestaukseen on käytetty Appium-työkalua ja myös opastukselle voitaisiin käyttää samaa työkalua.

Opastuksen tyylittely jäi vähäiseksi tässä työssä ja tyylit kirjoitettiin suoraan samaan tiedostoon logiikan ja käyttöliittymäelementtien kanssa. Tyylit olisi hyvä määritellä muun sovelluksen käyttämien tumman ja vaalean teemojen yhteyteen. Opastusta ei myöskään tyylitelty ollenkaan sovelluksen vaaleaan teemaan sopivaksi.

Opastus olisi myös hyvä lisätä Docean-sovelluksen verkkoselainversioon. Tässä työssä opastus toteutettiin pelkästään mobiilisovellukseen. Käytännössä saman Tooltip-komponentin pitäisi myös toimia verkkoselaimissa, koska sovellus käyttää muutenkin react-native-web avoimen lähdekoodin kirjastoa muuntamaan React Nativen komponentit verkkoselaimiin käyviksi. Ainoastaan react-native-modal kirjaston Modal-komponentti pitäisi vaihtaa verkkoselaimeen sopivaksi. Tämä kuitenkin on vain spekulaatiota ja esimerkiksi React Nativen measure-metodi ei välttämättä toimi ja toiminnolle jouduttaisiin toteuttamaan verkkoselaimen tukeva komponentin sijainnin määrittelevä metodi.

Myös työn aiheesta poikkeava mutta silti käyttäjäkokemusta parantavana ominaisuutena tulevaisuudessa Docean-sovellukseen kannattaisi toteuttaa ehdottamani kolmannen osapuolen kirjautumistavat. Tuttujen kirjautumistapojen avulla kirjautumistapahtumaa saataisiin erittäin paljon sujuvammaksi ja helpommaksi käyttäjille.

Toteutetusta opastuksesta olisi hyvä saada varsinaista käyttäjäpalautetta. Käyttäjäpalautteen perusteella opastusta voitaisiin jatkossa kehittää paremmin vastaamaan erilaisten käyttäjien tarpeita. Palautteen saaminen voi kuitenkin olla haastavaa sovelluksen vähäisen käyttäjämäärän takia

ja luonteeltaan työajalla käyttävänä sovelluksena. Kuitenkin yhden tai kahden hengen pienyrityksiltä palautetta voisi saada, asiakas joutuisi kuluttamaan hieman aikaa parannusehdotusten kirjaamiseen, mutta saisi vastineeksi paremman sovelluksen.

Lähteet

8 Design Guidelines for Complex Applications. Kaplan K. 2020. Artikkeli monimutkaisten applikaatioiden suunnittelun ohjenuorista. Viitattu 22.4.2021. https://www.nngroup.com/articles/complex-application-design/

About W3C. N.d. Lisätietoa W3C-organisaatiosta sisältävät verkkosivut. Viitattu 27.8.2021. https://www.w3.org/Consortium/

Adobe Lightroom. 2021. Adobe Lightroom sovellus Android Play -kaupassa. Viitattu 17.5.2021. https://play.google.com/store/apps/details?id=com.adobe.lrmobile&hl=fi&gl=US

App Center Test. Cenovsky L., Gimm V., Green K., Renshaw T., Wester G., Wilson G. 2020. Dokumentaatioverkkosivut Microsoft App Center Test- palvelulle. Viitattu 17.5.2021. https://docs.microsoft.com/en-us/appcenter/test-cloud/

Contrast Checker. 2021. Verkkosivu kahden värin välisen kontrastisuhteen laskemiselle. Viitattu 6.5.2021. https://webaim.org/resources/contrastchecker/

Core Components and Native Components. 2021. Opasverkkosivu React Native komponenteille. Viitattu 5.4.2021. https://reactnative.dev/docs/intro-react-native-components

Cross-platform mobile frameworks used by software developers worldwide in 2019 and 2020. Shanghong L. 2020. Statista-palvelun kaavio kehittäjien käyttyimmille järjestelmäriippumattomille sovelluskehyksille. Viitattu 17.5.2021. https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/

Direct Manipulation. 2021. React Native- sovelluskehyksen dokumentointisivu käyttöliittymäelementtien manipuloimiseen. Viitattu 16.5.2021. https://reactnative.dev/docs/direct-manipulation#measurecallback

Framework. 2013. Verkkosivusto teknologiasanastolle, alasivu sovelluskehyksen selitykselle. Viitattu 5.4.2021. https://techterms.com/definition/framework

Google Chrome. 2021. Google Chrome -verkkoselain versio 90.0.4430.212. Viitattu 17.5.2021. https://www.google.com/intl/fi_fi/chrome/

Instructional Overlays and Coach Marks for Mobile Apps. Harley A. 2014. Artikkeli mobiiliapplikaatioiden apu- ja infonäkymistä. Viitattu 22.4.2021. https://www.nngroup.com/articles/mobile-instructional-overlay/

Introducing Hooks. N.d. React-sovelluskehyksen dokumentointisivut Hooks-ominaisuudelle. Viitattu 16.5.2021. https://reactjs.org/docs/hooks-intro.html

Introducing JSON. N.d. Tietosivut JSON-tiedonsiirtoformaatista. Viitattu 17.5.2021. https://www.json.org/json-en.html

Mobile-App Onboarding: An Analysis of Components and Techniques. Joyce A. 2020. Artikkeli mobiilisovellukseen perehdytyksen komponenttien ja tekniikoiden analyysista. Viitattu 22.4.2021. https://www.nngroup.com/articles/mobile-app-onboarding/

Mobile app onboarding: best practices and examples. 2020. Artikkeli UI-kehityksen kannalta sovelluksen käyttöönoton opastukselle. https://uxplanet.org/mobile-app-onboarding-best-practices-and-examples-b0001816fda

Mobile Tutorials: Wasted Effort or Efficiency Boost? Joyce A. 2020. Tutkimus mobiilisovellusten opasohjelmien hyödyn määrästä. https://www.nngroup.com/articles/mobile-tutorials/

Popovers, tooltips for React Native. 2021. React-native-popable kirjaston npm-verkkosivusto. Viitattu 10.5.2021. https://www.npmjs.com/package/react-native-popable

React. A JavaScript library for building user interfaces. 2021. Verkkosivusto React-sovelluskehyksen dokumentaatiolle ja oppaille. Viitattu 30.3.2021. https://reactjs.org/

React Native. 2021. Verkkosivuston etusivu React Native-sovelluskehyksen dokumentaatiolle ja oppaille. Viitattu 30.3.2021. https://reactnative.dev/

React Native for Web. 2021. Github-tietolähde React Native For Web- sovelluskirjastolle. Viitattu 3.5.2021. https://github.com/necolas/react-native-web

React Native: Native Tooltips (Android/iOS). Chauhan P. 2021. React Native tooltips kirjaston npm-verkkosivusto. Viitattu 10.5.2021. https://www.npmjs.com/package/react-native-tooltips

React-native-popable Sandbox. N.d. React-native-popable-kirjaston demo expo-verkkosivustolla. Viitattu 17.5.2021. https://snack.expo.io/dmLOIiVHy

Rethinking Mobile Tutorials: Which Patterns Really Work? Neil T. 2021. Artikkeli kourallisesta olemassaolevista mobiilisovellusten opasohjelmista ja niiden toimivuudesta. https://www.smashingmagazine.com/2014/04/rethinking-mobile-tutorials-which-patterns-really-work/

Tooltip. N.d. Verkkosivu React Native Elements sovelluskirjaston dokumentointiverkkosivu Tooltip-komponentille. Viitattu 15.5.2021. https://reactnativeelements.com/docs/tooltip

Tooltip Guidelines. Joyce A. 2019. Artikkeli vihjetekstien ohjenuorista. Viitattu 22.3.2021. https://www.nngroup.com/articles/tooltip-guidelines/?lm=mobile-instructional-overlay&pt=article

Tutkimus- ja kehittämistoiminta. N.d. Tilastokeskuksen määritelmä tutkimus- ja kehittämistoiminnalle. Viitattu 17.5.2021. https://www.stat.fi/meta/kas/t_ktoiminta.html#tab2

Työelämän tutkiva kehittämistoiminta. N.d. Kuvaus työelämän tutkivalle kehitystoiminnalle Jyväskylän ammattikorkeakoulun oppimateriaalisivustolla. Viitattu 17.5.2021. https://oppimateriaalit.jamk.fi/yamk-kasikirja/tyoelaman-tutkiva-kehittamistoiminta/

Understanding Success Criterion 1.4.4: Resize text. N.d. WCAG 2.1 1.4.4:2018 standardin hyväksymiskriteerin lisätietosivut. Viitattu 28.4.2021.
https://www.w3.org/WAI/WCAG21/Understanding/resize-text.html

Understanding Success Criterion 1.4.6: Contrast (Enhanced). N.d. WCAG 2.1 1.4.6:2018 standardin hyväksymiskriteerin lisätietosivut. Viitattu 28.4.2021.
https://www.w3.org/WAI/WCAG21/Understanding/contrast-enhanced.html

Understanding Success Criterion 2.3.2: Three Flashes. N.d. WCAG 2.1 2.3.2:2018 standardin hyväksymiskriteerin lisätietosivut. Viitattu 28.4.2021.
https://www.w3.org/WAI/WCAG21/Understanding/three-flashes.html

Web Content Accessibility Guidelines (WCAG) 2.1. 2019. Virallinen suomenkielinen käännös verkkosisällön saavutettavuusohjeille (WCAG 2.1). Viitattu 27.8.2021. https://www.w3.org/Translations/WCAG21-fi/#perceivable.

What is a Mockup? Hufford B. 2021. Verkkoartikkeli, jossa avataan mockup-termiä. Viitattu 9.5.2021. https://cliquestudios.com/mockups/

Liitteet

Liite 1. Tooltip.js

```
import React from "react";
import { View, TouchableOpacity, StyleSheet } from "react-native";
import update from "immutability-helper";
import { TooltipModal } from "./TooltipModal";
import { Icon } from "./Elements";
import { wp, hp } from "../lib/helperFns";

const INITIAL_TOOLTIP_STATE = {
  isVisible: false,
  layout: { x: 0, y: 0, width: 0, height: 0, pageX: 0, pageY: hp(50) },
  componentLayout: { x: 0, y: 0, width: 0, height: 0 },
  tooltipLayout: { x: 0, y: 0, width: 0, height: 0 },
  caret: "top",
};
function tooltipReducer(state = {}, action) {
  switch (action.type) {
    case "set":
      return update(state, {
        [action.prop]: { $set: action.value },
      });
    case "setPosition":
      return update(state, {
        pageY: { $set: state.layout.pageY + action.pageY },
        pageX: { $set: action.pageX },
        caret: { $set: action.caret },
        xOverflow: { $set: action.xOverflow },
      });
    case "resetProp":
      return update(state, {
        [action.prop]: { $set: INITIAL_TOOLTIP_STATE[action.prop] },
      });
    default:
      throw new Error();
  }
}
export const Tooltip = (props) => {
  const [state, dispatchState] = React.useReducer(
    tooltipReducer,
    INITIAL_TOOLTIP_STATE
  );
  const view = React.useRef(null);

  const setState = (prop, value) => {
    dispatchState({
      type: "set",
```

```
    prop,
    value,
  });
};

const measureWrapper = (fn) => {
  if (view?.current) {
    view.current.measure((x, y, width, height, pageX, pageY) => {
      if (!pageY) {
        setState("layout", {
          x,
          y,
          width,
          height,
          pageX,
          pageY,
          undefinedPageY: true,
        });
      } else {
        setState("layout", { x, y, width, height, pageX, pageY });
      }
      if (fn) fn();
    });
  }
};

React.useEffect(() => {
  const tooltipHeight = state.tooltipLayout.height;
  const tooltipWidth = state.tooltipLayout.width;
  let _pageX;
  let xOverflow = false;
  if (state.layout.pageX + tooltipWidth > wp(98)) {
    _pageX = state.layout.pageX - tooltipWidth;
    xOverflow = true;
  } else {
    _pageX = props.icon
      ? state.layout.pageX
      : state.layout.pageX -
        (state.layout.width ? state.layout.width / 2 : 0);
  }
  if (state.layout.undefinedPageY) {
    dispatchState({
      type: "setPosition",
      pageY: hp(50) - tooltipHeight,
      pageX: 0,
      caret: "none",
      xOverflow,
    });
  } else if (state.layout.pageY + tooltipHeight > hp(90)) {
    dispatchState({
```

```
      type: "setPosition",
      pageY: -(tooltipHeight + (state.componentLayout.height ?? 0)),
      pageX: _pageX,
      caret: "bottom",
      xOverflow,
    });
  } else if (state.componentLayout.height) {
    dispatchState({
      type: "setPosition",
      pageY: state.componentLayout.height / 2,
      pageX: _pageX,
      caret: "top",
      xOverflow,
    });
  }
}, [state.layout, state.tooltipLayout]);

if (props.tip) {
  return (
    <View
      ref={view}
      onLayout={() => null}
      collapsable={false}
      style={[
        styles.tipWrapper,
        {
          borderWidth:
            state.layout.undefinedPageY &&
            ((props.icon && state.isVisible) || props.isVisible)
              ? 2
              : 0,
        },
      ]}
    >
      {props.icon ? (
        <TouchableOpacity
          onPress={() => {
            measureWrapper(() => setState("isVisible", true));
          }}
        >
          <View style={[styles.tipIcon, { backgroundColor: "#ffcc1b" }]}>
            <Icon name="help" size={wp(6)} color={"#000"} />
          </View>
        </TouchableOpacity>
      ) : null}
      <View
        style={{ flex: props.flex ?? 1 }}
        onLayout={(ev) => setState("componentLayout", ev.nativeEvent.layout)}
      >
        {props.children}
```

```
        </View>
        <TooltipModal
          caretRightSide={state.xOverflow}
          customBackdrop={!props.icon}
          skip={props.skip}
          isVisible={props.icon ? state.isVisible : props.isVisible}
          onTooltipLayout={(ev) => {
            setState("tooltipLayout", ev.nativeEvent.layout);
          }}
          tip={props.tip}
          caret={state.caret}
          top={state.pageY || state.layout.pageY}
          left={state.pageX || state.layout.pageX}
          hide={() => {
            if (props.icon) {
              setState("isVisible", false);
            } else {
              props.hideModal();
            }
          }}
          onHide={() => {
            dispatchState({
              type: "resetProp",
              prop: "pageY",
            });
          }}
        />
      </View>
    );
  } else {
    return props.children;
  }
};

const styles = StyleSheet.create({
  tipWrapper: {
    alignItems: "center",
    flexDirection: "row",
    borderColor: "red",
  },
  tipIcon: {
    height: wp(8),
    width: wp(8),
    borderRadius: wp(4),
    alignItems: "center",
    justifyContent: "center",
  },
});
```

Liite 2. TooltipModal.js

```
import React from "react";
import { View, Text, TouchableOpacity, StyleSheet } from "react-native";
import Modal from "react-native-modal";
import { Icon } from "./Elements";
import { wp, hp } from "../lib/helperFns";

const caretSize = 10;
export const TooltipModal = (props) => {
  return (
    <Modal
      backdropOpacity={props.customBackdrop ? 1 : 0.2}
      deviceHeight={hp(100)}
      isVisible={props.isVisible}
      onBackdropPress={props.hide}
      onBackButtonPress={props.hide}
      onModalHide={props.onHide}
      customBackdrop={
        props.customBackdrop ? (
          <TouchableOpacity
            onPress={props.hide}
            style={{ height: hp(95), zIndex: 1 }}
          >
            <TouchableOpacity
              onPress={props.skip}
              style={{
                zIndex: 2,
                position: "absolute",
                bottom: 0,
                left: 0,
                aspectRatio: 7 / 1,
                width: wp(100),
                backgroundColor: "#ffcc1b",
                alignItems: "center",
                justifyContent: "center",
                flexDirection: "row",
              }}
            >
              <Text style={{ color: "#000", paddingRight: 8 }}>
                Sulje vihjeet
              </Text>
              <Icon name="close" size={wp(7)} color={"#000"} />
            </TouchableOpacity>
          </TouchableOpacity>
        ) : null
      }
    >
      <View
        testID={"ToolTipWrapper"}
```

```
        onLayout={props.onTooltipLayout}
        style={{
          position: "absolute",
          width: wp(50),
          top: props.top,
          left: props.left,
          alignItems: props.caretRightSide ? "flex-end" : "flex-start",
        }}
      >
        {props.caret === "top" ? <View style={styles[props.caret]} /> : null}

        <TouchableOpacity onPress={props.hide}>
          <View style={styles.tipContainer}>
            <Text>{props.tip}</Text>
          </View>
        </TouchableOpacity>

        {props.caret === "bottom" ? <View style={styles[props.caret]} /> : null}
      </View>
    </Modal>
  );
};

const styles = StyleSheet.create({
  tipContainer: {
    backgroundColor: "#295165",
    padding: 8,
    borderWidth: 2,
    borderColor: "#fff",
  },
  top: {
    width: 0,
    height: 0,
    backgroundColor: "transparent",
    borderStyle: "solid",
    borderLeftWidth: caretSize,
    borderRightWidth: caretSize,
    borderBottomWidth: caretSize * 2,
    borderLeftColor: "transparent",
    borderRightColor: "transparent",
    borderBottomColor: "#fff",
  },
  bottom: {
    width: 0,
    height: 0,
    backgroundColor: "transparent",
    borderStyle: "solid",
    borderLeftWidth: caretSize,
    borderRightWidth: caretSize,
    borderTopWidth: caretSize * 2,
```

```
    borderLeftColor: "transparent",
    borderRightColor: "transparent",
    borderTopColor: "#fff",
  },
});
```