Tampereen ammattikorkeakoulu

Ohjelmistokirjaston tekeminen

React Native

Akio Ide

OPINNÄYTETYÖ
Lokakuu 2020

Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

IDE, AKIO:
Ohjelmistokirjaston tekeminen
React Native

Opinnäytetyö 31 sivua, joista liitteitä 0 sivua
Lokakuu 2020

Opinnäytetyön toimeksiantaja, Haltu oy, on ohjelmistotuotanto yritys. Haltu on erikoistunut muun muassa web- ja mobiilisovelluksien kehittämiseen sekä palvelimien ylläpitoon. Haltun toimeksianto oli toteuttaa kirjasto, joka parantaisi mobiilisovelluksien käyttökokemuksia. Yrityksen tarpeiden analysoinnin tuloksena opinnäytetyön tarkoitukseksi asetettiin luoda React Nativelle kirjasto, jolla on mahdollista lisätä animoituja aloitusruutuja. Opinnäyteprosessissa tutustuttiin erilaisiin tapoihin toteuttaa kirjastoja React Nativelle ja selvitettiin ongelmia, joita kirjaston luonnissa voidaan kohdata

Kun projektia toteutettiin, selvisi, että React Nativelle luodun kirjaston julkaisemiseen tarkoitettuja ohjeita ei löydy helposti, eikä niissä mainita vastaan tulevista ongelmista. Tästä syystä ongelmien selvittäminen on aikaa vievää, sillä harvoin niihin löytyi suoranaisia ratkaisuja. Näiden lisäksi kirjaston toteuttamisessa piti huomioida React Nativen oletuksena käytettävien natiivialustojen kielet ja versiot, jotta kirjasto olisi mahdollisimman helppokäyttöinen ja monella sovelluksella toimiva.

Lopputuloksena tehtiin React Nativelle kirjasto, jolla pystyy tuottamaan animoidun aloitusruudun pelkästään Androidille. Työmääräarvioon sisältyi myös toteutusta iOS-alustalle, mutta pelkästään Android tuen toteutus vastasi opinnäytetyön harjoitustyöhön varattua aikaa. Jatkokehityksessä kannattaa toteuttaa animoidun aloitusruudun toistoa myös muille alustoille. Jatkokehityksessä tulisi ottaa huomioon myös dokumentaation vähäinen määrä netissä sekä toteuttaa kirjasto yksi toimiva osa kerrallaan kirjastokokonaisuuden sijaan.

Asiasanat: react native, ohjelmistokirjasto, API, aloitusruutu

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Software Production

IDE, AKIO:
Creating a Programming Library
React Native

Bachelor's thesis 31 pages, appendices 0 pages
October 2020

The commissioner of this thesis is a software production company Haltu oy. Haltu is specialized on web- and mobile applications, as well as administering servers. The commission from Haltu was to create a library that would improve the user-experience of mobile applications. After analysing the needs of the company, the objective of this thesis was defined as creating a programming library for React Native, which would make creating animated splash screens possible. The purpose of this thesis was to become familiar with different ways to create libraries with React Native, and to investigate possible problems that might come up.

Upon creating the project, it became clear that instructions for publishing a library created for React Native were not available, and the few that were did not mention possible problems the user might encounter. Because of this, solving problems was time consuming since there rarely were any direct answers to them. In addition, when creating the library the versions and programming languages which native platform React Native uses by default had to be considered, so the library would be as easy to use as possible, and would work on different applications.

As an end result for this thesis, a library that can create animated splash screens only for Android was made for React Native. The workload estimate did also include implementation for iOS-devices, but creating only the Android-support for the library took all the time that was reserved for the practical work of this thesis. The library could be developed further to include the implementation also for iOS-platforms. The amount of documentation on the Internet should also take into consideration when developing further, as well as working on the library one part at a time instead of creating the whole library at once.

Key words: react native, programming library, API, splash screen

SISÄLLYS

ERITYISSANASTO

Y-sukupolvi	Sukupolvi, joka on syntynyt noin 1980-luvun ja 1990-luvun puolivälin välillä.
Android studio	Googlen omistama mobiilikehitysympäristö, jolla voi kehittää Android-sovelluksia.
Xcode	Applen omistama mobiilikehitysympäristö, jolla voi kehittää iOS-sovelluksia.
WP	WP (Windows Phone) on Microsoftin kehittämä mobiilikäyttöjärjestelmä.
DOM	DOM (Document Object Model) on puu-tietorakenne olioista.
State	Tieto, joka ilmaisee komponentin tilaa. State:n arvot muuttuvat komponentin sisällä.
Props	Tieto, joka luovutetaan eteenpäin hallinnoimaan komponentin tilaa.
Cordova	Apache Software Foundationin omistama ohjelmistokehys, joka mahdollistaa alustariippumattoman mobiilisovelluksen kehittämisen.
Ionic	Ohjelmistokehys, jonka pohjana on Apache Cordova. Ionicin erikoisuus on tehdä hyvä käyttökokemus ja käyttöliittymän vuorovaikutus sovellukseen.
Flutter	Googlen kehittämä ohjelmistokehys, joka mahdollistaa alustariippumattoman mobiilisovelluksen kehittämistä käyttäen Dart-ohjelmointikieltä.
AST	AST (async syntax tree) on välikieli, jolla on puutieto rakenne. AST koostuu ainoastaan oleellisesta asioista kuten muuttujat ja muuttujien arvot tietyissä olosuhteissa.
Tarball	Tiedostot, joita on pakattu.
Kotlin	JetBrainsin kehittämä ohjelmointikieli. Kotlinilla kirjoitetut koodit käännetään Javaan sovellusta käynnistettäessä.
Android SDK	Android-sovelluskehitys paketti, joka sisältää Android kehitykseen tarvittavan virheen etsintä ohjelman, kirjaston ja rajapinnan.

Gradle	Projektin rakennus skripti, jolla voi rakentaa kehitysympäristöä.

1 JOHDANTO

Älypuhelinten käyttö on kasvanut vuosi vuodelta. Vuoden 2019 tilaston mukaan yli 40 prosenttia koko maailman väestöstä omistaa älypuhelimen (O'Dea, 2020), ja y-sukupolvesta jopa 90 prosenttia (Vogels, 2019). Yhtenä vaikuttavana tekijänä älypuhelinten laajamittaiseen leviämiseen ovat olleet eri mobiilisovellukset, joilla voi helpottaa jokapäiväistä elämää tai tuoda nautintoa käyttäjän arkeen. Älypuhelinten leviämisen myötä mobiilisovelluksien merkitys on kasvanut myös markkinoinnin näkökulmasta.

Älypuhelimen alustana käytetään pääosin Android- ja iOS-käyttöjärjestelmiä, minkä vuoksi alustariippumattomat kehityskehykset ovat esiintyneet kustannustehokkaana vaihtoehtona kehittää mobiilisovelluksia. Muuten samaa sovellusta jouduttaisiin kehittää useaan kertaan eri kielellä ja alustalla, että se saataisiin julkaistua kaikille laitteille.

React Native on yksi suosituimmista alusta-riippumattoman sovelluksen kehityskehyksistä. React Native on mahdollistanut mobiilisovelluksien kehityksen JavaScriptia- ja HTML-ohjelmointikieliä muistuttavalla tyylillä, mikä antoi web kehittäjille mahdollisuuden tutustua mobiilisovelluksien kehitykseen. JavaScriptillä oli jo entuudestaan valtavia määriä kirjastoja, joita oli nyt mahdollista käyttää myös React Nativessa. Tästä syystä React Nativen suosio nousi räjähdysmäisesti, eikä se tule laskemaan vielä vuosiin.

Avoimet lähdekoodit ovat lisääntyneet kovaa vauhtia, minkä vuoksi ennustetaan kaiken ohjelmoinnin tapahtuvan pelkästään valmiiksi kirjoitettuja koodeja käyttämällä muutaman vuoden päästä. Kirjastot ja kehykset auttavat säästämään aikaa kehittäjiltä, jolloin kehittäjät voivat keskittyä muihin työtehtäviin. Omaan sovellukseen sopimattomat kirjastot voivat kuitenkin aiheuttaa lisätyötä, ja siksi sopivan kirjaston löytäminen tulee olemaan arvostettava kyky tulevaisuudessa. (innovative, 2017)

Tässä opinnäytetyössä on tarkoituksena tutustua React Nativen ohjelmistokirjaston luontiin ja syventää tietämystä eri kirjasto tyyppiin. Tavoitteena on toteuttaa

React Nativella toimiva ohjelmistokirjastoa ja oppia tunnistamaan hyviä lähdekoodeja kirjaston luojan näkökulmasta.

Tämän opinnäytetyön tarkoituksena on tutustua React Native-ohjelmistokirjaston luontiin ja syventää tietämystä eri kirjastotyyppeihin. Tavoitteena on toteuttaa React Nativella toimiva ohjelmistokirjasto ja oppia tunnistamaan hyviä lähdekoodeja kirjaston luojan näkökulmasta.

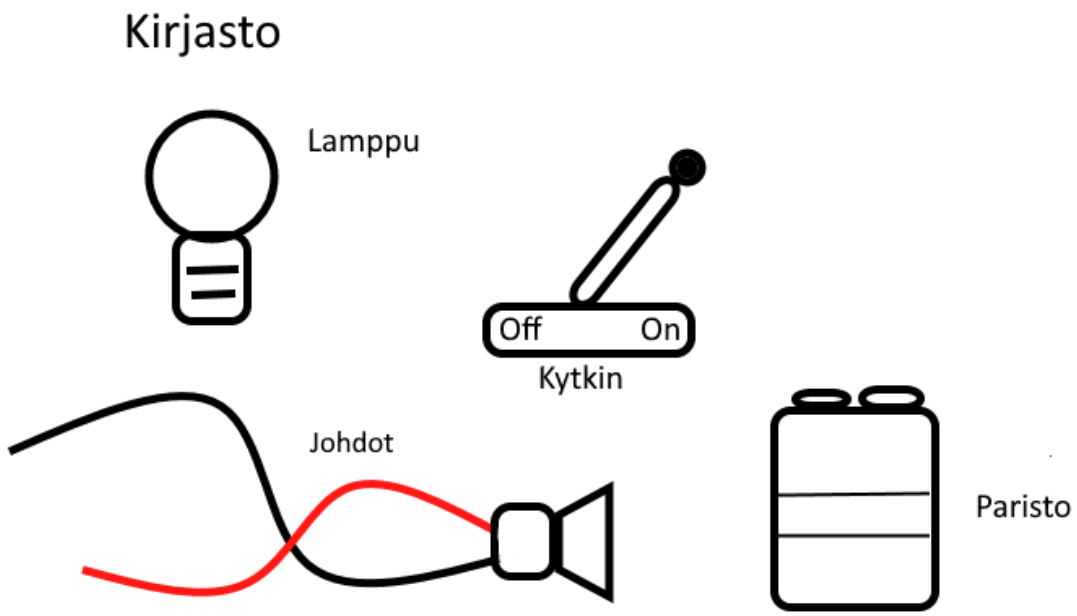
2 OHJELMISTOKIRJASTOT

Yhtenä ohjelmoinnin suurimpina edistäjinä ovat toimineet eri kirjastot, kehykset ja muut koodikokoelmat, joiden tärkeimpänä tehtävänä on auttaa luomaan uudelleenkäytettäviä koodeja valmiiksi. Kirjasto, kehys ja API usein sekoitetaan keskenään, mutta kullakin on olemassa selkeät eroavaisuudet.

2.1 Kirjasto

Kirjasto on kokoelma usein käytettävistä funktioista ja luokista. Kirjaston päätarkoitus on vähentää kehitysaikaa tekemällä usein käytettäviä luokkia ja funktioita valmiiksi, ettei kehittäjän tarvitsisi käyttää aikaa koodin kirjoittamiseen ja sen tarkistamiseen. Kirjasto auttaa myös pitämään koodin yksinkertaisena, jolloin koodin luettavuus paranee. Usein kirjasto ei toimi sellaisenaan sovelluksena, joten kyseessä on harvoin ohjelmistotiedosto. (Rapid API, 2020)

Kuva 1 on kuvaava esimerkki siitä, miten kirjastoa voidaan verrata tavarataloon. Kehittäjän pitäisi valmistaa palava lamppu ja hankkia tavaratalosta osia, kuten lamppu, johto, paristo ja kytkin. Kehittäjän ei siis tarvitse luoda lampun osia itse, sillä ne löytyvät jo valmiiksi luotuina, mutta osat eivät toimi sellaisenaan. Kehittäjän tehtävä on yhdistää osat kasaan ja saada valmistettua palava lamppu. (Web no hito, 2017)



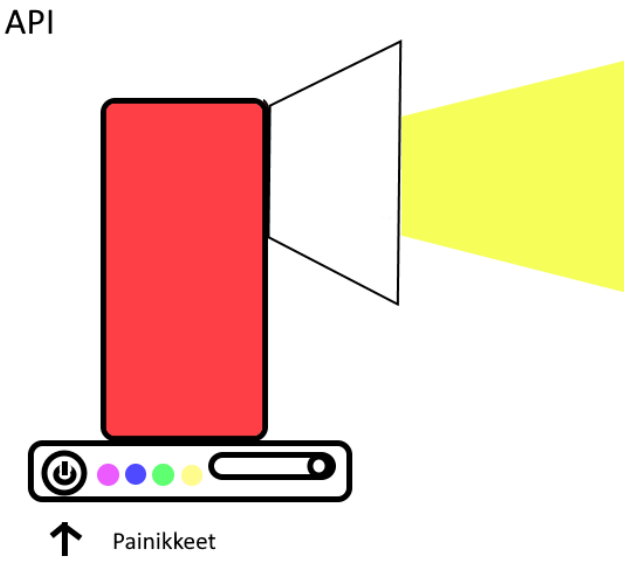
KUVA 1. Kirjaston vertaus lamppuun

Kirjaston luomiseen yksi nyrkkisääntö on tehdä mahdollisimman pieniä osia, jotta käyttäjän on mahdollista soveltaa niitä moneen eri käyttötarkoitukseen. Spesifiseen tarkoitukseen tehdyt osat pienentävät sen käyttötarkoituksia, ja käyttäjät voivat tunnistaa kirjaston huonona käyttökokemuksena. (Best Colleges Online, 2019)

2.2 API

API (Application programming interface) on käsite, joka mahdollistaa ohjelmien ja sovelluksien välisen vuorovaikutuksen. Ohjelmien yhtenä käyttötarkoituksena on julkaista kolmannen osapuolen käytettäväksi. Siitä huolimatta ohjelmaa ei välttämättä voida julkaista sellaisenaan, koska siihen voi liittyä tietoturvariskejä tai kehittäjä ei vain yksinkertaisesti halua. API:n pääideana on julkaista vain osa ohjelmasta kolmannen osapuolen käyttöön. Tällöin sovellus pystyisi käyttämään ohjelman ominaisuuksia, ja mahdollistaa että ohjelmat ja sovellukset pystyvät keskustelemaan keskenään. (Jacobson, Brail & Woods, 2011)

Jatkaen aiempaa lamppu-esimerkkiä, API:n voi ajatella kuvan 2 mukaisena valaisimena, jolla on eri ominaisuuksia. Valaisimella voi sytyttää valon, vaihtaa valon väriä ja säätää kirkkautta. Näiden ominaisuuksien käyttämistä varten valaisimella on eri painikkeita, joilla käyttäjä voi käyttää valaisinta tietämättä, miten valaisin toimii järjestelmällisesti. API vastaa painikkeisiin.

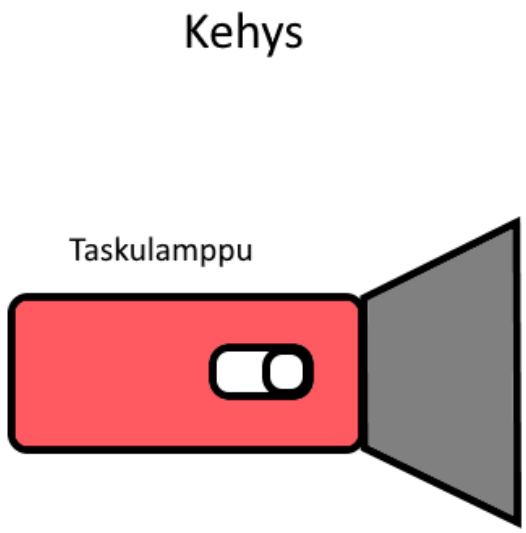


KUVA 2. API:n vertaus lamppuun

2.3 Kehys

Kehys, eli framework, viittaa kokonaisuudessa toimivaan malliin. Toisin kuin kirjastoissa, kehys voi toimia myös sovelluksena sellaisenaan. Kehyksen pyrkimyksenä on vähentää ohjelmien arkkitehtuurin käytettävää aikaa valmistamalla pohjan kehyksenä, jossa on ohjelmaan tarvittavat minimivaatimukset. Kehys on usein koottu useista eri API:sta ja kirjastoista. (Lerman & Miller, 2012)

Kehystä voi kuvata valmiina tuotteena. Aikaisempaan lamppu-esimerkkiin viitaten kehystä voi ajatella Kuvan 3 mukaisena taskulamppuna. Taskulamppu on koottu eri osista ja se toimii sellaisenaan valon lähteenä, vaikka käyttäjällä ei olisi sähkötekniikan tietämystä. Ulkoasun voi muokata tarvittaessa, mutta sisäisen järjestelmän muokkaaminen on vaikeaa. (Web no hito, 2017)



KUVA 3. Kehyksen vertaus lamppuun

Kehyksen yhtenä tärkeänä ominaisuutena on sen rajattu käytettävyys. Käyttäjän on pystyttävä käyttämään kehystä ilman kunnollista tietoa, miten kehys toimii. Tämän takia kehyksen käyttöä on mahdollista rajata tiettyihin käyttötarkoituksiin, jotta käyttäjät eivät tekisi virheitä niin helposti. (Lerman & Miller, 2012)

3 REACT NATIVE

React Native on alunperin Facebookin kehittämä JavaScript-ohjelmistokehys, joka on tarkoitettu mobiilisovelluksen kehittämiseen. Tavallisesti natiivi sovelluksien kehittämiseen on käytettävä palvelun tarjoamia työkaluja, jonka takia kehittäjän on tehtävä sama sovellus kullekin alustoille. Jos kehittäjän pitäisi tehdä sovellusta sekä Android- että iOS-alustalle, pitäisi hänen opetella käyttämään Android Studio- ja xCodea-kehitysympäristöjä. Alustoissa käytetään eri ohjelmointikieliä ja molemmissa on oma tapansa kehittää sovelluksia. React Nativella puolestaan pystyy kehittämään sovelluksia alustasta riippumatta käyttäen JavaScript-ohjelmointikieltä. (Eisenman, 2016)

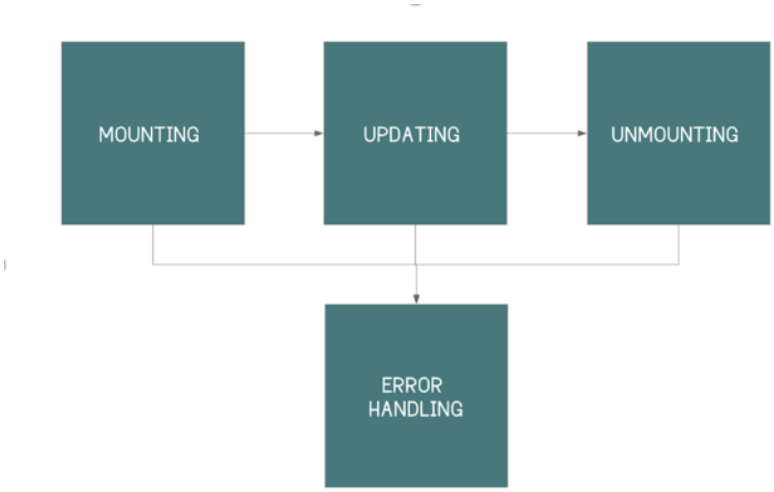
Kuten React Nativen virallisessa dokumentissa mainitaan, “React Nativella on mahdollista rakentaa mobiilisovelluksia käyttäen vain JavaScriptia. Se käyttää samaa tyyliä kuin React, ja antaa koota tyylikkään mobiili UI:n deklaratiivisilla komponenteilla. ” (Facebook) Komponenttien ja tyylien ohjeet kirjataan React Nativelle ominaisella tavalla, jotka muistuttavat hyvin paljon HTML ja CSS. React Nativen konseptina oli mahdollistaa mobiili sovelluksien kehittäminen myös web -kehittäjille ilman suurta osaamis kynnystä. React Nativella on mahdollista rakentaa Android-, iOS-, Web- ja WP-sovelluksia.

3.1 Elämänkaari

React Nativessa elämänkaari tarkoittaa prosessia, jossa komponentit luodaan ja poistetaan. Tämän prosessin aikana kutsuttavia funktioita sanotaan elämänkaari metodeiksi. Elämänkaari -metodit mahdollistavat komponenttien muuttumista sovelluksen ollessa käynnissä.

React Nativen elämänkaari voidaan jakaa kuvion 1 mukaisesti neljään osaan, jotka prosessoidaan Mounting, Updating ja Unmounting järjestyksessä. Näiden lisäksi on virheenhallinta-vaihe, jota käsitellään koko prosessin ajan. Mounting-vaiheessa alustetaan sovelluksessa näytettäviä komponentteja ja päivitetään DOM:a. Updating-vaihe tulee esille silloin kun props tai state sisältöä muutetaan. Komponentit renderöidään niiden mukaisesti, jolloin muutokset tulee näkyviin

käyttäjälle. Unmounting vaihe käynnistyy vain silloin kun komponentti poistetaan DOM:sta. (Kolodiy, 2019)



KUVIO 1. Elämänkaari jaettuna vaiheisiin(Kolodiy, 2019)

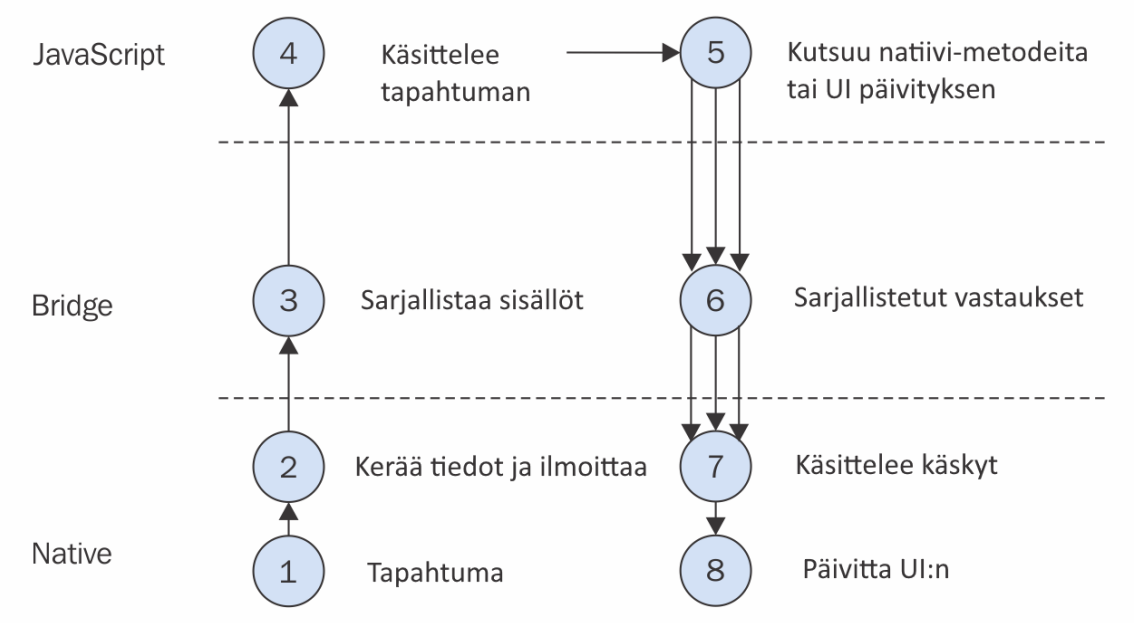
3.2 Silta

React Native on suhteellisen helppo käyttää, vaikka ei olisi tietämystä kehyksen toiminnallisuudesta, jos kehittäjä on kehittänyt Reactilla web sovelluksia. Kirjaston luomisessa on kuitenkin hyvä tietää etukäteen miten React Native toimii kokonaisuudessaan, sillä alustariippumattomuuden mahdollistamiseksi React Native on rakennettu omalaatuisella tavalla verrattuna muihin alustariippumattomiin kehyksiin.

React Nativen lisäksi on olemassa muita kehyksiä, joilla voi kehittää native sovelluksia JavaScriptilla ja HTML:llä. Toteuttaakseen alustariippumattomuutta, React Native ei kuitenkaan käytä webview:ta, kuten Cordova:ssa ja Ionic:ssa eikä tee Flutterin tapaan koodista AST:ta sovellusta rakennettaessa, jota käännettäisiin native-koodiiksi. Sen sijaan React Nativella on sisäänrakennettu järjestelmä, jolla pääsee native-näkymiin ja -komponentteihin käsiksi. React Native käsittelee JavaScript- ja Native-säiettä, käyttäen siltaa. (Frachet, 2017)

Silta on käsite, joka luo kaksisuuntaisen ja asynkronisen yhteydenkahden säikeen välillä. React Nativessa silta toimii kahden säikeen välittäjänä, jonka tehtä-

vänä on käsitellä asynkroniset käskyt. Tämän ansiosta kuvion 2 mukaisella tavalla JavaScript säikeestä voidaan lähettää JSON-viestinä käskyjä nativeen, ja native-puoli voi lähettää sovelluksessa tapahtuneita tapahtumia JavaScript puolelle käsiteltäväksi. Asynkronisuus mahdollistaa näkymien sulavan hallinnan, koska mikään ei ole estämässä suoritusta kummassakaan säikeessä. (Frachet, 2017)



KUVIO 2. React nativen arkkitehtuuri (Novic, 2017)

3.3 Kirjastot React Nativessa

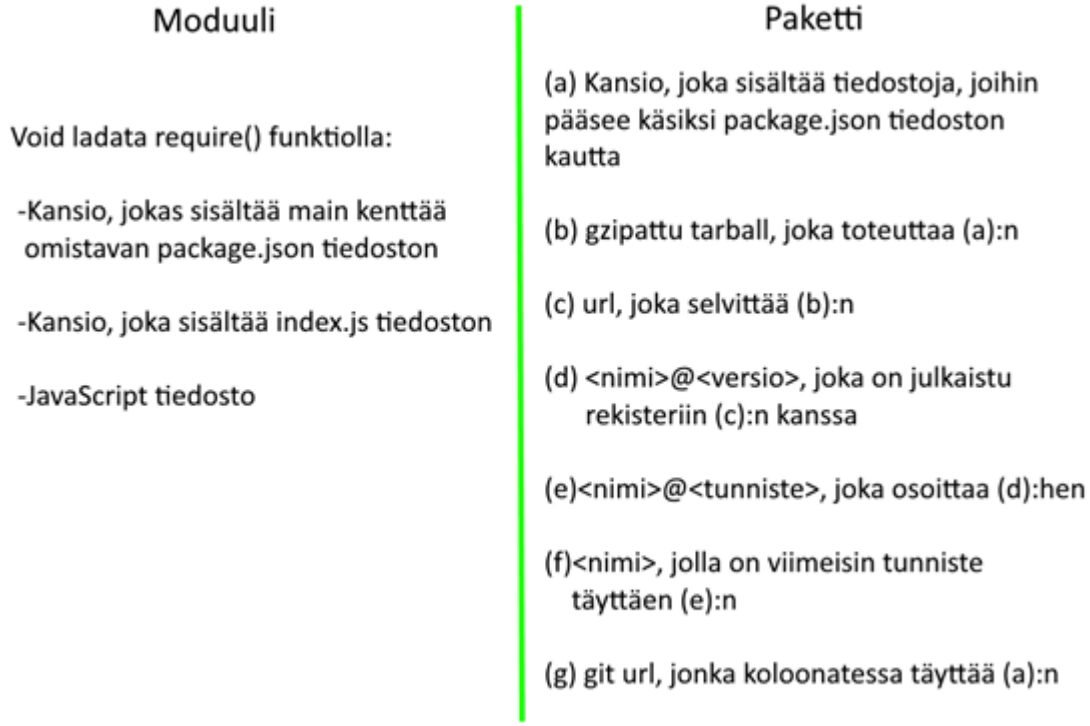
Kuten kaikissa kehitysympäristöissä myös React Nativeen on mahdollista tuoda ulkopuolisia kirjastoja edistämään projektin kehitystä. React Nativeen voi tuoda periaatteessa kaikki JavaScript kirjastot, sekä osa natiivi-puolta käsitteleviä kirjastoja.

3.3.1 Moduuli ja paketti

JavaScriptissä uudelleenkäytettävää koodia kutsutaan moduuleiksi tai paketeiksi. Näiden ansiosta kehittäjät pystyvät rakentamaan sovelluksia tehokkaammin ilman kunnollista tietämystä moduulin osa-alueesta. Yleisenä moduulin hyvänä aatteena on rakentaa pieni palikka, joka ratkaisee yhden ongelman. Erikoisia ongelmia voi ratkaista laajentamalla tätä pientä palikkaa. Moduuleita on olemassa erityyppisiä, sillä on olemassa sekä lokaalilla että serverillä käytettäviä.

Kuten npm paketinhallintajärjestelmän virallisessa dokumentissa sanotaan, ”Node.js:lla ja npm:lla on selkeä määritys paketeille ja moduuleille, vaikka näitä helposti sekoitetaan” (npm Docs). Moduulin määritteenä on tiedosto tai kansio, jota voi ladata node.js:ssä require() -funktiolla, eli kaikki JavaScript tiedostot luokitellaan moduuleiksi. React Nativen tapauksessa komponentit sekä itse sovellus voidaan luokitella moduuleiksi.

Kuvioon 3 on kerätty moduulin ja paketin suurimmat erot. Paketin tärkeimpänä määritelmänä on kansio, joka sisältää tiedostoja joihin pääsee käsiksi package.json tiedoston kautta. Paketti on suurempi kokonaisuus, sillä paketti sisältää useita moduulia. (Nakanishi, 2017)



KUVIO 3. Moduulin ja paketin ero

3.3.2 Natiivi moduuli

Vaikka React Native sisältää kaikki oleelliset toiminnot ja UI-komponentit, kaikkia ominaisuuksia ei ole kuitenkaan toteutettu, eikä jokaiseen nativen ominaisuuksiin voi vaikuttaa React Nativen puolelta. Tällöin kehittäjän on itse käsiteltävä natiiviien alustojen API:a. Tätä varten React Nativella on mahdollista luoda natiiveja moduuleja natiiveihin alustoihin. Natiivit moduulit ovat luokkia, jotka toteuttavat rajapintaa jolla on mahdollista käyttää silta-ominaisuutta. Natiivin moduulin funktioita on mahdollista kutsua React Native säikeeltä ja palauttaa arvo tai tapahtuma React Nativen käsiteltäväksi. Natiiveja moduuleja hyödyntämällä React Native voi käyttää vain natiivi-alustoissa käytettäviä API:a tai kirjastoja, kuten tietokannan hallinnointi ja kuvien prosessointi. React Nativen koodit toimivat JavaScript-virtuaalikoneella, jolloin joitain asioita on parempi hoitaa natiivin puolella. Tämän lisäksi natiiveilla moduuleilla voi tehdä uusia säikeitä natiivin puolelle, jonka ansiosta sovellus voi prosessoida suuria määriä tietoa ilman viivytystä. Viivytyksen vähentämisellä voidaan parantaa käyttäjäkokemusta sovelluksessa. (Fullstack.io)

Natiiveista moduuleista kuitenkin sanotaan, ettei niitä kannata käyttää, ellei ole aivan pakko, koska niiden ylläpitäminen on suuri haaste. Natiiveja moduuleja kuuluu tehdä kaikkiin alustoihin, jos sen toiminnan halutaan toimivan kaikissa alustoissa. Yleensä kyseessä ovat vain Android- ja iOS-alustat, mutta tarvittaessa myös WP. Kussakin alustassa on eri tyyppiset toiminnot ja niistä kaikista on pidettävä huolta. Sen lisäksi React Nativen teknisen eritelmän muutoksiin on reagoitava, jotta natiivien moduulien kutsuminen onnistuisi edelleen. Ongelmien sattuessa sen etsiminen ja korjaaminen on hankalaa, koska tähän tarkoitukseen ei ole mitään työkaluja, eli käytännössä virheen löytämistä varten sovellusta saatetaan joutua käynnistämään sekä React Nativen puolella
että kunkin alustojen kehitystyökalun puolella. (Fullstack.io)

4 PROJEKTI

4.1 Toimeksianto

Haltu oy on antanut toimeksiannon toteuttaa mobiilisovelluksien kehitykseen ohjelmistokirjaston, jolla voi parantaa sovelluksien käyttäjäkokemusta. Nykypäivänä sovelluskehityksessä on otettu aiempaa enemmän käyttäjäkokemusta ja käytettävyyttä huomioon. Käyttäjäkokemusta voi parantaa esimerkiksi lisäämällä siirtymis-animaatioita tai käyttämällä puhelimen värinää painikkeita painettaessa. Todellisuudessa käyttäjäkokemusta lisäävät ominaisuudet toteutetaan jälkikäteen tai ei ollenkaan. Käyttäjäkokemusta lisäävät ominaisuudet vaativat toteutuksen lisäksi suunnittelua ja harvat asiakkaat haluavat käyttää siihen ylimääräisiä resursseja. Toimeksiannolla haluttiin luoda mahdollisuus tuottaa käyttäjäkokemuksia parantavia tekijöitä kustannustehokkaasti.

Haltulla React Native on usein käytetty mobiilisovelluksen kehityskehys. React Nativen avulla on mahdollista tuottaa mobiilisovelluksia usealle alustalle samaan aikaan, jolloin sovelluksen kustannukset voidaan pitää pienenä. Tämän vuoksi kirjasto päätettiin tehdä React Nativelle.

Opinnäytetyön projektina toteutetaan ohjelmistokirjasto, jossa tarkoituksena on tutustua juuri React Native kirjaston luomiseen käytännönläheisesti. Yksittäisen moduulin tai natiivin komponentin luomiseen löytyy ohjeita, mutta kokonaisen kirjaston luomisen ohjeita oli harvinaisia. Kuitenkin niissä ei ole tuotu esille vastaantulevia ongelmia tai mahdollisia ratkaisuja.

4.2 Suunnittelu

4.2.1 Toteutettava ominaisuus

Toteutettavaksi toiminnaksi valittiin aloitusruutu. Yleisesti mobiilisovelluksissa aloitusruudulla tarkoitetaan näkymää, joka aukeaa heti sovellusta avatessa. Näkymä näytetään sen ajan kun sovellus lataa sisältönsä, jonka jälkeen sovellus

siirtyy varsinaiseen sisältöön. Perinteisesti alkuruutu pidetään hyvin yksinkertaisena ja sisältää lähinnä yrityksen tai sovelluksen logon, ja johon saatetaan lisätä yrityksen tunnuslause.

Aloitusruutu saattaa näkyä todella lyhyen ajan, jolloin se saattaa mennä käyttäjällä huomaamatta ohi, mutta siitä huolimatta aloitusruutua käytetään sovelluksen brändäämisessä. Kuten Material Design-dokumentaatiossa mainitaan, “Aloitusnäkymän näyttäminen voi vähentää pitkän ajan tuntemista ja sillä on mahdollisuus lisätä iloisen käyttäjäkokemuksen”(Google). Aloitusruutu antaa ensivaikutelman käyttäjälle sovelluksesta, ja hyvin toteutettuna helpottaa käyttäjän tuntemaan koko sovelluksen miellyttäväksi. Tämä toteutus on kuitenkin hieman kaksipiippuinen asia. Huonosti toteutetun aloitusruudun takia käyttäjä voi kokea sovelluksen päinvastaisesti huonona, vaikka sisältö olisikin erinomainen.

Aloitusruudun toteuttamiseen React Nativelle on jo olemassa kirjastoja, sekä kirjastoja, joilla voi tehdä animoituja aloitusruutuja. Näitä ei kuitenkaan ole kovinkaan montaa, ja missään kirjastossa ei ole yhtä tai kahta eri animaatiotyyliä enempää vaihtoehtoja. Kussakin kirjastossa on erilaisia vaatimuksia kirjaston käyttäjälle, jonka takia käyttäjät eivät pysty kokeilemaan jokaista kirjastoa lyhyessä ajassa löytääkseen omalle sovellukselleen sopivan animoidun aloitusruudun. React Nativessa hyvälaatuisen aloitusruudun toteuttamiseen vaaditaan jonkinlaista työtä kirjaston käyttäjältä, mutta jos yhdellä toteutuksella voi tehdä monenlaisia eri aloitusruutuja, kehittäjä voi keskittyä pääasiassa aloitusruudun valitsemiseen. Tästä syystä päätimme kehittää kirjaston, jolla käyttäjän on mahdollista toteuttaa useanlaisia animoituja aloitusruutuja vain yhden kirjaston avulla.

4.2.2 Kirjaston toteutustapa

React Nativessa yhtenä aloitusruudun toteuttamisen haasteena on valkoinen ruutu sovelluksen käynnistyessä. Natiivi näyttää valkoista ruutua sen ajan, kun React Native säiettä käynnistetään. Tämän vuoksi React Native ei pääse vaikuttamaan valkoiseen ruutuun pelkästään JavaScript moduulia tekemällä.

Ongelman voi kuitenkin ratkaista käyttämällä natiivia moduulia. Molemmille alustoille tehdään natiivi moduuli, jolla voi käynnistää aloitusruudun sisältävän ruudun. Natiiviin moduuliin rakennetaan funktioita, joilla voi käynnistää ja sulkea aloitusruudun. Käynnistys -funktiota kutsutaan pääasiassa natiivi säikeessä ja sulkemiseen tarkoitettu funktio kutsutaan React Native -säikeessä sitten, kun sovellus on saanut ladattua sisällön. Animointi tehdään natiivi-säikeen puolella, että saadaan toteutettua sulava animaatio.

Koska kyseessä on natiivi moduuli, animoitu aloitusruutu on toteutettava molemmille alustoille. Ennen natiivi moduuli-kirjaston toteutusta päätettiin tehdä toimiva aloitusruutu lähdekoodit molemmille alustoille, jonka tarkoituksena oli siirtää lopulta natiivi moduuli kirjastoon. React Native projektin natiivi koodit ovat oletuksena Androidin puolella Java ja iOS:n Objective-C, mutta alustavan selvityksen mukaan natiivi moduulia oli mahdollista toteuttaa Android puolen Kotlinilla ja iOS puolen Swiftillä, joten nämä kielet valittiin kehittämiseen.

4.3 Toteutus

4.3.1 Animoidun aloitusruudun toteuttamien

Projekti aloitettiin toteuttamalla Androidille pelkkä aloitusruutu. Androidissa yleinen tapa toteuttaa aloitusruutu on luoda aloitusruutua varten aktiviteetti, jossa näytetään aloitusruudun sisältö ja siirrytään siitä pääaktiviteettiin. Aloitusruudun aktiviteetissa on mahdollista pysäyttää sovellus hetkeksi, jotta käyttäjä ehtisi näkemään aloitusruudun. Muulloin aloitusruutu vain vilahtaisi näytöllä hetkellisesti, ettei ihmissilmällä ehtisi edes seurata sitä.

Ei ole kuitenkaan oikeaoppisen käytännön mukaista, jos sovelluksen sisältö vaatii lataamista. Material Design-dokumentissa mainitaan hyvän käyttökokemuksen toteuttavan aloitusruudun sellaisena, joka lataa sisällöt valmiiksi sen aikana kun aloitusruutua näytetään. Kun sovellus on saanut ladattua sisällön, aloitusruutu poistuu käyttäjän silmistä. (Google)

Tätä ei voida toteuttaa äskettäin mainitulla aloitusruudun toteutustavalla, koska pääaktiviteetin sisältö ei lataudu ennen kuin sovellus on siirtynyt pääaktiviteettiin.

Ongelma voidaan ratkaista näyttämällä pääaktiviteetissa aloitusruutua sen ajan, kun sovellus latautuu. Päänäkymän ruudulle laitetaan koko aktiviteettia peittävä näkymä, joka on täsmälleen samanlainen kuin alkuruudussa, jotta käyttäjä ei huomaisisi ruudun vaihtumista. Kun sovellus on saanut ladattua sisällön, koko ruutua peittävä näkymä otetaan pois näkyviltä, jolloin käyttäjän eteen saadaan esiin päänäkymä, jossa on sisällöt valmiina. Android Studiolla on kirjasto, joka on tarkoitettu näkymän ja komponentin animoimiseen, joten on mahdollista lisätä animaatio siilloin kun koko näkymää peittävä ruutu poistetaan näkyvistä. (Medenjak, 2019)

Pelkän animoidun aloitusruudun toteuttaminen ei ollut hankala, mutta haasteen lisäsi se, että tavoitteena oli luoda React Nativen kanssa toimiva kirjasto. React Nativessa Android sovelluksen minimi SDK versioksi on asetettu 16 ja kohde versio on 29. Toisin sanoen kirjaston koodin pitäisi olla sellainen, joka toimii SDK versio 16 ja 29 välillä. Osa toiminnallisuuksista eivät siis toimi vanhassa versioissa, ja samalla pitää huomioida uusia ominaisuuksia. Aloitusruudun toteuttamisessa haasteena oli luoda koko näyttöä peittävä näkymä. (Facebook, 2020)

SDK 23 versiosta lähtien Android-sovelluksille lisättiin ominaisuus puolittaa ruutu, jolloin käyttäjä pystyi käyttämään kahta eri sovellusta samaan aikaan yhdessä ruudussa. SDK 22 ja sitä alemmilla versioilla riitti, että hakee pelkästään koko ruudun koon, mutta tämä ei toimi ruudun puolitusominaisuuden kanssa. Ratkaisuna tehtiin kaksi eri koodia.

Kuten kuvan 4 funktiossa, ensin tarkistetaan sovellusta käyttävän laitteen SDK versio, jonka jälkeen ajetaan eri koodit riippuen versiosta. Onneksi ruudun koon suhteen ei tarvinnut huomioida muuta kun SDK versio 23 ennen ja jälkeen, mutta joissain tapauksessa olisi saattanut vaatia enemmän eri koodeja eri versioon kohtaan.

```
protected void onLayout(boolean changed, int left, int top, int right, int bottom) {
    super.onLayout(changed, left, top, right, bottom);
    int leftBound;
    int topBound;
    int rightBound;
    int bottomBound;
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
        WindowInsets windowInsets = this.getRootWindowInsets();
        leftBound = -windowInsets.getSystemWindowInsetLeft();
        topBound = -windowInsets.getSystemWindowInsetTop();
        rightBound = this.getWidth() + windowInsets.getSystemWindowInsetRight();
        bottomBound = this.getHeight() + windowInsets.getSystemWindowInsetBottom();
    } else {
        Activity activity = (Activity) this.getContext();
        Window window = activity.getWindow();
        Rect rectangle = new Rect();
        window.getDecorView().getWindowVisibleDisplayFrame(rectangle);
        leftBound = -rectangle.left;
        topBound = -rectangle.top;
        rightBound = rectangle.right - rectangle.left;
        bottomBound = rectangle.bottom - rectangle.top;
    }

    Drawable splashDrawable = this.splashDrawable;

    splashDrawable.setBounds(leftBound, topBound, rightBound, bottomBound);
}
```

KUVA 4. Aloitusruudun toteuttamien React Nativelle

4.3.2 Natiivi moduulin tekeminen

Jotta Androidille tehtyjä funktioita voisi kutsua React Native-säikeestä, funktioista on tehtävä moduuli ja moduuleista tehdään paketti. Niiden toteuttamiseksi React Nativella on valmis luokka. Kuvan 5 mukaisella tavalla tehdään oma luokka, joka perii ReactContextBaseJavaModule:n. Tällöin @ReactMethod:lla varustetun funktion voi kutsua React Native-säikeessä.

```java
@ReactModule(name = SplashAnimationsModule.MODULE_NAME)
public class SplashAnimationsModule extends ReactContextBaseJavaModule {
    public static final String MODULE_NAME = "SplashAnimations";
    private final ReactApplicationContext reactContext;

    public SplashAnimationsModule(ReactApplicationContext reactContext) {
        super(reactContext);
        this.reactContext = reactContext;
    }

    @Override
    public String getName() { return MODULE_NAME; }

    @ReactMethod
    public void hide() {
        final Activity activity = getCurrentActivity();
        if (activity != null) {
            AnimatedLoadingScreen.hide(activity);
        }
    }
}
```

KUVA 5 Moduulin luominen Androidilla

Myös paketin tekemiseen vaaditaan oma luokka. Kuva 6 mukaisella tavalla luokan pitäisi toteuttaa ReactPackage rajapintaa, jossa pakettiin lisätään aiemmin tehty moduuli. Tarvittaessa pakettiin on mahdollista lisätä useita moduuleja. Paketin voi ottaa käyttöön lisäämällä paketti React Nativen aktiviteetti luokassa.

```java
public class SplashAnimationsPackage implements ReactPackage {
    @Override
    public List<NativeModule> createNativeModules(ReactApplicationContext reactContext) {
        return Arrays.<~>asList(new SplashAnimationsModule(reactContext));
    }

    @Override
    public List<ViewManager> createViewManagers(ReactApplicationContext reactContext) {
        return Collections.emptyList();
    }
}
```

KUVA 6. Paketin luominen Androidilla

4.3.3 Kirjaston julkaisemien

Android-tuen valmistuttua seuraava tehtävä oli tuoda androidille tehdyt toiminnot React Nativen kirjastoon. Toteutus päätettiin tehdä verkosta löytyvän Creating a

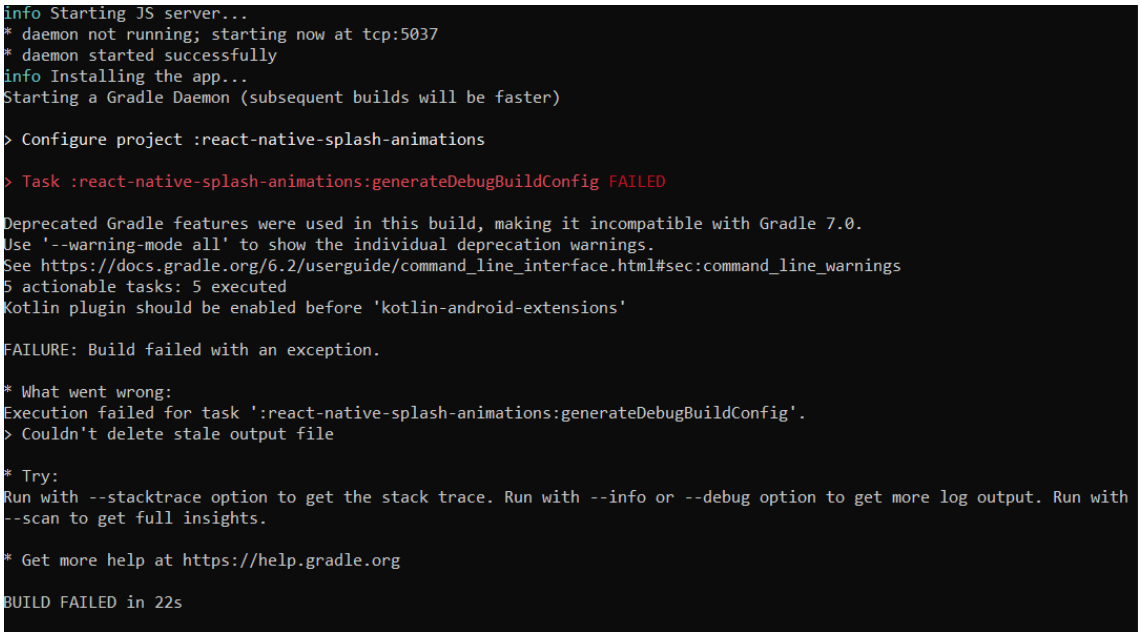
Native Module in React Native (Berlin, 2020) mukaisesti. Ohjeessa käytettiin react-native-create-library npm-pakettia, jolloin projekti tehtiin sen mukaisesti. Ongelmia ilmeni, kun react-native-create-library:lla luotuja kirjastoja ei voitu alustaa avaamalla ne pelkästään Android Studioon. Virheitä selvitettiin kohta kerrallaan, mutta jossain vaiheessa tuli vastaan seinä, josta ei päässyt ylitse. Kirjaston lähdekoodia tarkasteltaessa huomattiin kirjaston olleen päivitetty viimeksi kaksi vuotta sitten. Näin ollen projektille otettiin toinen kirjaston luontiin tarkoitettu paketti create-react-native-module, jota suositeltiin React Nativen virallisessa dokumentaatiossa (Facebook, 2020). Luotettavan lähteen lisäksi paketti oli päivitetty viimeksi kaksi kuukautta sitten. Create-react-native-module:lla luotu kirjasto pääsi alustusvaiheen läpi ja projekti oli mahdollista käynnistää. Projekti julkaistiin npm pakettina, jotta kirjaston toimivuutta voitaisiin testata käytännön läheisesti React Nativella luodulla sovelluksella.

Npm-paketin julkaisuun vaaditaan rekisteröityminen ja package.json -tiedosto. Tiedostosta täytyy löytyä tietyt tiedot, kuten sovelluksen nimi, versio ja pakettiin tarvittavat riippuvuudet. Näiden lisäksi komentorivillä pitää ajaa muutama komento, joilla alustetaan julkaisu-ympäristöä ja julkaistaan kirjasto verkkoon. Kirjasto menee julkaisuun, jos samannimistä kirjastoa ei ole olemassa. Tämän jälkeen kirjaston voi ladata projektiin verkon välityksellä. Npm:n maksullisena ominaisuutena kirjaston pystyisi julkaisemaan yksityisenä, ettei muut käyttäjät pystyisi lataamaan testivaiheessa olevaa kirjastoa, mutta ilmaisella käyttäjätilillä kaikki julkaisut ovat julkisia.

4.3.4 Testaus

Kirjasto julkaistiin npm-pakettina ja tehtiin uusi React Native-projekti, johon ladattiin kyseinen npm paketti, jotta voitiin kokeilla sen toimivuutta. Jälleen kerran kirjastolle tuli ongelmia vastaaan. React Native-sovellusta ei voitu käynnistää kirjaston takia ja antoi virheilmoituksen, joka nähdään kuvassa 7. Ongelma johtui Kotlinilla kirjoitetusta koodista. Vaikka alustavassa selvityksessä todettiin natiivi moduulin olevan mahdollista kirjoittaa Kotlinilla, se ei tarkoittanut Kotlinin toimivan React Native-projektissa sellaisenaan, vaan projektille piti tuoda Kotlin-laajennus, jotta projektin koodit olisivat toimineet. Kotlin-laajennus piti tehdä koko

React Native-projektille, eikä pelkästään paketin laajentaminen riittänyt. Ongelman todettiin olevan liian suuri vaiva kirjaston käyttäjälle, jolloin projektin oli pakko uudelleenkirjoittaa Javalla. Uudelleenkirjoittamiseen pystyttiin käyttämään apuna Android Studiossa olevaa toimintoa, jolla on mahdollista katselmoida Javaksi käännettyä Kotlin-koodia. Sitä ei kuitenkaan voinut käyttää suora sillä se sisälsi paljon ylimääräisiä muuttujia ja funktioita, joita piti itse poistella ja korjata toimivaksi.



KUVA 7. Virheilmoitus sovellusta käynnistyttäessä

Aikomusten mukaisesti, kirjasto ei enää vaatinut Kotlin laajennusta. Tämä ei kuitenkaan korjannut kaikkia ongelmia. Tällä kertaa virheilmoituksessa kerrottiin build.gradle-tiedoston olevan viallinen. Android Studio alustaa projektin kehitysympäristön käyttämällä build.gradle. Tavallisesti projektia ei voi käynnistää, jos Android Studio ei pysty suorittaa build.gradle:n kokonaan.

Kirjaston build.gradle-tiedostoa päivitettiin, mutta asia ei siltikään edennyt mitenkään. Monien eri kokeilujen jälkeen ei ongelma siltikään korjaantunut, jolloin päätettiin palata lähtökohtaan, ja tällä kertaa tehdä toimivaa kirjastoa. React Nativen virallisessa dokumentissa oli ohje miten tehdä natiivi moduuli, ja siinä käytettiin esimerkkinä toast (Facebook, 2020). Toast on hetkellisesti ruutuun ilmestyvä teksti, joka löytyy molemmista natiivi luokasta. Tarkoitus oli siis tehdä toimiva kirjasto täsmälleen ohjeen mukaisesti ja verrata nykyiseen, että ongelman lähde

saadaan löydettyä. Vaikka toast kirjaston tekeminen olisi epäonnistunut, siitä oltaisiin silti saatu jonkinlaisia tuloksia, sillä epäonnistuminen on myöskin tulos.

Lopputuloksena vastaan tuli edelleen sama ongelma. Gradle-tiedosto aiheuttaa edelleen ongelman sovelluksen käynnistyessä. Vinkin saamista varten kirjasto verrattiin jo olemassa olevaan kirjastoon, jossa havaittiin yksi iso ero. Toimivan kirjaston koko oli 54kB ja projektimme kirjasto oli 310kB, vaikka olemassa oleva kirjasto pitäisi olla paljon suurempi, koska se sisältää paljon enemmän toiminnallisuutta. Lähdekoodia tarkasteltaessa havaittiin eron johtuvan .npmignore-tiedostosta, jota projektissamme ei ollut ollenkaan. .npmignore-tiedostoon voi kirjata tiedostoja, joita ei ladata npm pakettia ladattaessa. Tähän kuului muunmuassa gradle-tiedosto, build-kansio ja muut tiedostot, joita tarvitaan kirjaston kehittämisessä, mutta ei kirjaston käyttöä varten. Toisin sanoen virheilmoitus johtui gradle tiedostosta, jota löytyi kirjastosta vaikka sen ei olisi kuulunut.

Toast-kirjasto pääsi vihdoin ongelmasta läpi, mutta vastaan tuli taas uusi ongelma, jossa kerrottiin toastin olevan jo toteutettuna React Nativeen. Näiden ongelman ohittamiseksi Module-luokkaan on lisättävä kuvan 8 mukaisesti canOverrideExistingModule() -funktio, joka sallii duplikoituneita natiiveja moduuleita.

```java
public class ToastLibraryModule extends ReactContextBaseJavaModule {
  private static ReactApplicationContext reactContext;

  private static final String DURATION_SHORT_KEY = "SHORT";
  private static final String DURATION_LONG_KEY = "LONG";

  ToastLibraryModule(ReactApplicationContext context) {
    super(context);
    reactContext = context;
  }

  @Override
  public String getName() {
    return "ToastLibrary";
  }

  @ReactMethod
  public void show(String message, int duration) {
    Toast.makeText(getReactApplicationContext(), message, duration).show();
  }
  @Override
  public Map<String, Object> getConstants() {
    final Map<String, Object> constants = new HashMap<>();
    constants.put(DURATION_SHORT_KEY, Toast.LENGTH_SHORT);
    constants.put(DURATION_LONG_KEY, Toast.LENGTH_LONG);
    return constants;
  }

  @Override
  public boolean canOverrideExistingModule() {
    return true;
  }
}
```

KUVA 8. Toast natiivi moduulin toteutus

Animoidun aloitusruudun kirjastoon lisättiin .npmignore-tiedosto, jonka jälkeen sovellus sai itsensä käynnistettyä. Kuvan 9 mukainen virhe tuli kuitenkin vastaan melkein heti sen jälkeen.

```
To reload the app press "r"
To open developer menu press "d"

[Sat Sep 26 2020 10:46:29.939] BUNDLE ./index.js

[Sat Sep 26 2020 10:46:31.543] ERROR  Error: Exception in HostObject::get(propName:SplashAnimations):
java.lang.IllegalArgumentException: Could not convert class com.splashanimations.HideAnimationType
[Sat Sep 26 2020 10:46:31.546] ERROR  Invariant Violation: Module AppRegistry is not a registered call
able module (calling runApplication)
[Sat Sep 26 2020 10:46:31.546] ERROR  Invariant Violation: Module AppRegistry is not a registered call
able module (calling runApplication)
```

KUVA 9. Virheilmoitus .npmignore tiedoston lisäämisen jälkeen

Ongelma oli loppujen lopuksi suhteellisen yksinkertainen, mutta sen selvittäminen vaati aikaa, sillä kyseisestä virheestä ei löytynyt tietoa mistään. Virhe oli esiintynyt tavallisesti muissa asioissa kuin natiivi moduulin luomisessa. Virheraportissa esiintyvä com.splashanimations.HideAnimationType on enum-luokka,

jota virheraportin mukaan yritetään kääntää tavalliseksi luokaksi. Virheen korjaaminen on sinänsä yksinkertainen, sillä enum piti vain muokata luokka-muotoiseksi. Tämän ratkaisun löytäminen olisi kuitenkin haastavaa monelle, sillä tavallisesti enum kääntyy luokaksi Java-tiedostoja käynnistettäessä, eikä kehittäjän tarvitse tietää siitä. Missään natiivin moduulikirjaston luomisen ohjeessa ei mainittu, ettei enum-kirjastoa olisi voinut käyttää tai että se voi aiheuttaa ongelmia. Tämän korjauksen jälkeen animoidun aloitusruudun kirjasto lähti toimimaan juuri suunnitellulla tavalla.

5 POHDINTA

Opinnäytetyön alkuperäinen tavoite oli luoda kirjasto, joka toimii Android- ja iOS-ympäristöissä, mutta iOS-tuen toteutuksen suhteen oli luovutettava, sillä sen tekemiseen olisi vaadittu aikaa, jota projektiin ei enää ollut. Pelkästään Android-tuen toteutus vei koko kirjaston kehitykseen varatun ajan, vaikka natiivin Android-sovelluksen kehitys piti olla työn helpoin osuus. iOS:n natiivin moduulin tekeminen olisi luultavasti vienyt vähintään yhtä paljon aikaa, ellei jopa enemmän kuin mitä käytettiin Androidiin. Tämä johtuu siitä, että Objective-C olisi pitänyt opetella uutena ohjelmointikielenä, koska Swiftilla toteutettaessa kirjasto olisi luultavasti vaatinut Swift-laajennuksen asentamista käyttäjän projektiin.

Onnistuneen kirjaston toteuttamiseksi järkevintä olisi ollut toteuttaa ensin toimivaa kirjastoa, esimerkiksi React Nativen dokumenttia noudattaen. Toimivan kirjaston olisi voinut muokata siitä askel kerrallaan omaksi kirjastoksi, muokkaamalla Android ja iOS natiiveja moduuleja.

Opinnäytetyötä voidaan jatkokehittää esim. luomalla luomalla kirjastoon tuki eri alustoille. Jatkokehityksen tavoitteeksi voisi asettaa eri alustojen natiivien moduulien toteuttamisessa esiintyvien ongelmien selvittäminen. Toisena vaihtoehtona on selvittää tarkemmin kirjaston toteuttamista pelkällä moduulilla. Opinnäytetyössä ei käsitellä tarkasti moduulilla toteutettavien kirjastojen rajoituksista.

Tämän projektin ansiosta saatiin kuitenkin selville React Nativen ohjelmistokirjaston vaatimukset ja mahdolliset esteet, jotka saattavat tulla esille kirjastoa luotaessa. Tavallisen moduuli-kirjaston luominen vaatii ainoastaan React Nativen toiminnallisuuden tietämystä, mutta sillä ei välttämättä voi luoda kirjastoa joka olisi tarpeeksi laadukas tai jolla olisi tarpeeksi suoritustehoja. Näiden toteuttamiseksi natiivien moduulien käyttö on välttämätöntä, mutta natiivi-sovelluksien luomiseen vaaditaan tietämystä React Nativen lisäksi jokaiseen toteutettavaan alustaan liittyen. Lisäksi React Nativella oletuksena asetettujen alustojen minimi-version ja kohdeversion välit ovat suhteellisen suuria. Käyttäjän pitäisi huolehtia että kirjasto toimii jokaisella versiolla, ja laadun varmistamiseksi testaaminen vaatii paljon

budjettia. Siksi laadukkaan kirjaston luomiseen vaaditaan todella paljon resursseja ja aikaa, joka on suuruudeltaan samaa luokkaa kuin yksi kokonainen sovellusprojekti.

Vaikka natiivin moduulin tekeminen on työlästä ja aikaa vievää, tätä on kuitenkin suositeltavaa kehittäjille, jotka haluaisivat päästä seuraavaan tasoon. Kirjaston toteutus on todella opettavaista ja oppii arvostamaan muiden tekemiä kirjastoja ja niiden tekijöitä.

LÄHTEET

Berlin, R. 2019.Creating a Native Module in React Native. Luettu 07.08.2020 https://medium.com/wix-engineering/creating-a-native-module-in-react-native-93bab0123e46

Best Colleges Online. n.d. 100 Ways to Improve Usability in Your Library. Luettu 13.03.2020 https://www.bestcollegesonline.com/blog/100-ways-to-improve-usability-in-your-library/

Eisenman, B. 2016. Learning React Native: Building Native Mobile Apps with JavaScript. 2. painos. California: O'Reilly Media

Facebook,n.d. React Native, Luettu 08.05.2020 https://reactnative.dev/docs/native-modules-setup

Facebook, 2020, Native Modules Setup. Luettu 28.08.2020 https://reactnative.dev/docs/native-modules-setup

Facebook, 2020, React Native. Luettu 18.06.2020 https://github.com/facebook/react-native

Frachet, M. 2017. Understanding the React Native bridge component concept. Luettu 15.05.2020 https://hackernoon.com/understanding-react-native-bridge-concept-e9526066ddb8

Fullstack.io, n.d. Native Modules. Luettu 29.05.2020 https://www.newline.co/fullstack-react/react-native/p/native-modules/

Google, n.d. Launch screen. Luettu 07.08.2020 https://material.io/design/communication/launch-screen.html#usage

innovative, 2017, New Programs for the Future of Public Libraries. Luettu 08.03.2020 https://www.iii.com/blog/new-programs-for-the-future-of-public-libraries/

Jacobson, D., Brail, G., Woods, D. 2011. APIs: A Strategy Guide. 1. painos. California: O'Reilly Media

Kolodiy, M. 2019. React Native Component Lifecycle. Luettu 08.05.2020 https://www.netguru.com/codestories/react-native-lifecycle

Lerman, J., Miller, R. 2012. Programming Entity Framework: DbContext. 1. painos. California: O'Reilly Media

Medenjak, D. 2019. Adding Animated Splash Screens. Luettu 02.06.2020 https://blog.davidmedenjak.com/android/2019/05/17/animated-splash-screens.html

Nakanishi. 2017. npm towa nanika / Package to module no chigai. Luettu 22.05.2020 http://better-than-i-was-yesterday.com/what-is-npm/

O'Dea, S. 2020. Smartphone penetration worldwide as share of global population 2016-2020. Luettu 01.03.2020 https://www.statista.com/statistics/203734/global-smartphone-penetration-per-capita-since-2005/#:~:text=For%202019%20the%20global%20smartphone,penetration%20has%20reached%2041.5%20percent

Novic, V. 2017. React Native - Building Mobile Apps with JavaScript. 1. painos. California: O'Reilly Media

Npm Docs. n.d. About npm. Luettu 22.05.2020 https://docs.npmjs.com/about-npm

Rapid API, 2020. API vs Library (What's the Difference?). Luettu 30.10.2020 https://rapidapi.com/blog/api-vs-library/

Vogels, E. 2019. Millenials stand out for their technology use, but older generations also embrace digital life. Luettu 01.03.2020 https://www.pewresearch.org/fact-tank/2019/09/09/us-generations-technology-use/

Web no hito. 2017. JSFreemuwaaku to raiburarii ga gottya ni natteiru hanasi. Luettu 13.03.2020 http://webnohito.me/tips/js-framework-library/