Aleksi Kytö

React Native: parhaat käytännöt

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

19.5.2024

Tiivistelmä

Tekijä:	Aleksi Kytö
Otsikko:	React Native: parhaat käytännöt
Sivumäärä:	32 sivua
Aika:	19.5.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mobile Solutions
Ohjaajat:	Lehtori Ilkka Kylmäniemi

Mobiilisovelluskehityksessä parhaat käytännöt ja toimintatavat muuttuvat teknologian ja työkalujen kehittyessä. Mobiilisovelluksia kehittäessä on hyvä huomioida suurilta osin sovelluksen toimintaan ja kehitykseen vaikuttavat asiat, kuten tässä projektissa käsitellyt projektin hallintaa helpottavat yleiset kirjoitus- ja rakennesäännöt, sekä sovelluksen toimintaan vaikuttavat asiat, joita ovat muun muassa tilanhallinta ja suuren tietomäärän esittäminen oikealla tavalla.

Tässä tutkimusprojektissa tutustutaan aluksi JavaScriptiin, joka on oleellinen osa React Native -mobiilikehitystä, sillä React Native pohjautuu Reactiin, joka on Facebookin JavaScript-kirjasto. Tämän jälkeen käydään läpi projektinhallinnan ja käytettävyyden kannalta tärkeitä asioita, kuten TypeScript, koodin formatointi ja linter, tyylien käyttö ja tiedostojen sekä kansioiden rakenne ja nimeäminen. Lisäksi projektissa tutkitaan sovelluksen tilanhallintaan liittyviä vaihtoehtoja, perehdytään Expon käyttötarkoituksiin ja tarkastellaan tiedon esittämiseen käytettäviä vaihtoehtoja, jotka tässä tapauksessa ovat FlatList ja FlashList.

Tutkimusprojektissa havaittua tietoa on hyvä käyttää osana mobiilisovelluskehitystä, mutta samalla on suositeltavaa olla avoimena myös uusille vaihtoehdoille, sillä teknologia ja työkalut kehittyvät jatkuvasti.

Avainsanat:	React Native, mobile development, best practices

Abstract

Author:	Aleksi Kytö
Title:	React Native: best practices
Number of Pages:	32 pages
Date:	19 May 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Mobile Solutions
Supervisors:	Ilkka Kylmäniemi, Senior Lecturer

In mobile application development, best practices and ways of working change as technology and tools evolve. When developing mobile applications, it is good to pay attention to the things that affect the operation and development of an application, such as the general writing and structure rules discussed in this thesis that facilitate project management, as well as the things that affect the operation of the application, such as space management and presenting a large amount of data in the right way.

This thesis first discusses JavaScript, which is an essential part of React Native development because React Native is based on React, which in turn is Facebook's JavaScript library. Subsequently, things which are important for project management and usability are presented/evaluated, such as TypeScript, code formatting and linter, the use of styles, and the structure and naming of files and folders. In addition, the project analyzes/investigates options related to the application's space management and Expo and goes through the options used to present information, which in this case are FlatList and FlashList.

The information gained in the final year project can be used as part of mobile application development, but at the same time it is recommended to be open to new emerging options as well because technology and tools are constantly developing.

Keywords: React Native, mobile development, best practices

Sisällys

1 Johdanto

Mobiilisovelluskehitykseen on luotu vuosien aikana useita eri ohjelmointikehyksiä, ja tekniikka kehittyy koko ajan. Uusien ja uudistuvien ohjelmointityökalujen myötä käytännöt saattavat lisääntyvien ja muuttuvien ominaisuuksien myötä vaihtua, minkä vuoksi alalla on hyvä pysyä tietoisena kehityksen kulusta.

Tässä tutkielmassa on tavoitteena selvittää parhaat tämänhetkiset React Native -ohjelmointikehykseen liittyvät käytännöt, jotta React Nativella ohjelmoitaessa asiat sujuisivat mahdollisimman hyvin. Tämän tutkielman tavoite on myös, että tutkielmassa käsitellyt aiheet edesauttaisivat lukijan kykyä luoda selkeä kansiorakenne selkeillä nimityksillä, sekä se, että luotu sovellus toimisi tehokkaasti, että sovelluksen käyttäjäkokemus olisi mahdollisimman hyvä.

Tutkielmassa perehdytään JavaScriptiin, joka toimii React Nativen ohjelmointikielenä, React Nativen historiaan ja React Nativen eroon muihin ohjelmointikehyksiin verrattuna, React Native-ohjelmointiin ja sen käyttötarkoituksiin sekä siihen, mitkä ovat tämänhetkisen tiedon valossa parhaat käytännöt React Native -ohjelmointikehystä käytettäessä. Tutkittaviin aiheisiin kuuluu muun muassa tilanhallinta, joka on suuressa roolissa sovelluksen toimivuuden ja reaktiivisuuden kannalta, sekä Expo, joka käytössä mahdollistaa helpon projektin aloituksen tuoden mukanaan muita helpotuksia sovelluskehittäjän arkeen. Viimeiseksi tutkittavaksi aiheeksi on valittu virtuaaliset listat ja uutena vaihtoehtona Flash-List, jotka ovat oleellisia, mikäli sovelluksessa halutaan esittää suuria määriä dataa listan muodossa.

Parhaita käytäntöjä miettiessä ja etsiessä on hyvä pitää mielessä, että asioiden toteuttamiseen on lähes aina useita erilaisia tapoja, tässä tutkielmassa ei esitetä mitään tapaa parhaana, vaan mahdollisesti yhtenä parhaista. Lopulta tavat käytännön toteutukseen on hyvä valita omien mieltymysten ja ennen kaikkea projektin vaatimusten ja tarpeiden perusteella. Käytäntöjä valitessa ja tutkiessa tutkielmassa on pyritty löytämään yleisesti eniten suositeltuja ja suosituimpia ta-

poja toteuttaa asioita, mutta koska ei ole olemassa yhtä yleispätevää React Nativella tehtävän mobiilisovelluskehityksen opasta, tässä käsiteltävä tieto koostuu erinäisistä verkkosivuista ja artikkeleista, joista saa riittävän selkeän kuvan parhaista käytännöistä. Tarkoitus ei ole vielä pureutua täysin kaikkeen mahdolliseen toimintaan React Native -ohjelmointikehyksessä, vaan poimia muutama oleellinen asia osaksi tutkielmaa.

Työ perustuu mielenkiintoon mobiilisovelluskehitystä kohtaan sekä haluun kehittyä mobiilisovelluskehittäjänä ja tietää aiheesta mahdollisimman paljon. Tavoite on myös voida kehittää React Nativella mahdollisimman optimaalisia sovelluksia mahdollisimman optimaalisin tavoin käyttäen parhaita olemassa olevia ja tiedettyjä käytäntöjä.

2 React Nativen perustyökalut

React Native on avoimen lähdekoodin käyttöliittymäohjelmointikehys, jonka on kehittänyt Meta Platforms. React Native pohjautuu Reactiin, joka on Facebookin JavaScript-kirjasto, jolla kehitetään käyttöliittymiä, mutta Reactista poiketen React Native ei ole suunniteltu verkkoselainkehitykseen vaan mobiilikehitykseen. Sana Native nimessä viittaa mahdollisuuteen kehittää natiivisti, eli alustakohtaisesti hahmontavia mobiilisovelluksia niin iOS- kuin Android-pohjaisiin laitteisiin. [Eisenmann 2017.]

React Native sai alkunsa Facebookin hackathon-tapahtumaan kehitettynä projektina kesällä 2013, jolloin Jordan Walke ja Christopher Chedeau kokosivat tiimin React Nativen kehittämiseen. Tammikuussa 2015 React Nativen ensimmäinen versio julkaistiin React.JS Conf -konferenssissa, ja saman vuoden maaliskuussa React Nativen lähdekoodi vaihdettiin suljetusta avoimeksi. Tammikuussa 2015 lähdekoodi myös julkaistiin kaikkien nähtäville sekä muokattavaksi GitHubiin. [The history of React Native: Facebook’s Open Source App Development Framework 2020.]

2.1 JavaScript

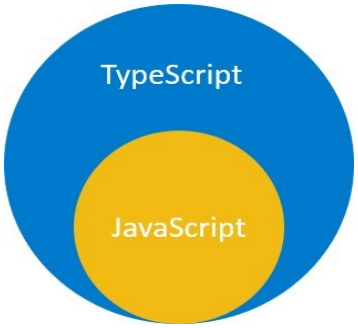
JavaScript on alkujaan Netscapen kehittämä, lähinnä verkkoympäristössä käytettävä dynaaminen tekstipohjainen ohjelmointikieli. JavaScriptiä voi käyttää sekä selainpuolen että palvelinpuolen ohjelmoinnissa, mikä näin ollen mahdollistaa interaktiivisuuden sovelluksessa tai verkkosivuilla. [History of JavaScript 2023.]

JavaScript on dynaamisesti tyypitetty, tulkattava oliopohjainen komentosarjakieli. JavaScript antaa mahdollisuuden luoda interaktiivisia, dynaamisesti päivittyviä elementtejä, joiden avulla käyttäjän on mahdollista tehdä haluttuja toimintoja ja näin ollen saada paras mahdollinen käyttökokemus [What is JavaScript? 2024]. Toisin sanoen JavaScriptillä saa luotua käytettävyyttä.

JavaScriptistä on luotu useita ohjelmointikehyksiä. Nämä ohjelmointikehykset ovat kokoelmia JavaScript-koodikirjastoista, jotka tarjoavat kehittäjälle valmiiksi kirjoitettua koodia käytettäväksi rutiiniohjelmointiominaisuuksiin ja -tehtäviin. Näistä ohjelmointikehyksistä yksi suosituimmista on React Native, jota tämä tutkielma käsittelee. [What is JavaScript used for? 2021.]

2.2 TypeScript-ohjelmointikieli

TypeScript on JavaScriptin syntaktinen superjoukko eli pohjimmiltaan sama asia kuin JavaScript, mutta hiukan muokattu versio. TypeScript käytännössä lisää syntaksin JavaScriptin päälle, jolloin kehittäjät voivat lisätä tyyppejä, mikä helpottaa niin sanottua tyypitystä sovelluskehityksessä (kuva 1).



Kuva 1. TypeScript verrattuna JavaScriptiin [What is TypeScript?].

TypeScriptin pääasiallinen tavoite on ottaa JavaScriptissä käyttöön valinnaiset tyypit, mikä auttaa välttämään ohjelmointivirheitä. TypeScript vaatii käyttäjältä tyypit, jonka vuoksi on käytännössä mahdotonta yrittää käyttää oikein asetettua tyyppiä väärässä paikassa. Tällä tavoin vältetään ohjelmointivirheitä ja voidaan varmistaa, että myös mahdolliset kehitystiimin muut jäsenet ymmärtävät, mistä tyypistä on muuttujien kohdalla kyse. Tärkeä ominaisuus TypeScriptissä on, että se tukee tulevia ES Nextin nykyisille JavaScriptmoottoreille suunniteltuja ominaisuuksia. Tämä tarkoittaa, että näitä ominaisuuksia on mahdollista käyttää, ennen kuin verkkoselaimet ja muut ympäristöt vielä tukevat niitä täysin. [What is TypeScript.]

2.3 Koodin formatointi ja lintterin käyttö

Koodin formatointi on koodin ulkomuodon muotoilua ja järjestämistä, mikä käytännössä tarkoittaa välilyöntejä, rivinvaihtoja ja sisennyksiä. Koodin formatoinnin tarkoitus on parantaa koodin ylläpidettävyyttä, luettavuutta ja sovelluskehittäjien koodin yhtenäisyyttä. Lintteröinti tarkoittaa koodin analysointia erinäisten ohjelmistotyökalujen avulla, minkä on tarkoitus löytää tyylivirheitä, toimintaa estäviä vikoja ja tekstillisiä virheitä koodista. [Kylmäniemi 2023.] Lintteröinti auttaa pitämään koodin laadun parempana ja vähentämään virheitä ennen kuin sovellus siirretään tuotantoympäristöön. Ero formatoinnin ja lintteröinnin välillä on siinä, että koodin formatointi keskittyy koodin ulkoasuun ja järjestämiseen, eikä

tarkasta koodia virheiden tai huonojen käytäntöjen varalta kuten lintteröinti [Kylmäniemi 2023]. Koodin formatointi parantaa koodin luettavuutta, ylläpidettävyyttä ja edistää sovelluskehittäjien kesken yhtenäistä koodityyliä, lintteröinti taas auttaa löytämään virheitä ja ongelmia koodista sitä kirjoitettaessa, joka edesauttaa koodin laadullisuutta ja vähentää tuotantoon pääseviä virheitä. Yksin tai varsinkin ryhmässä työskenneltäessä koodipohjan ylläpidon avuksi React Nativessa on hyvä käyttää koodin formatointiin ja lintteröintiin luotuja kirjastoja. Formatointiin suositeltu kirjasto on nimeltään Prettier ja lintterinä tämän kanssa käytetään usein ESLint-nimistä kirjastoa [Kylmäniemi 2023].

Prettier on koodin formatoija, joka automaattisesti muotoilee ja järjestää koodin ennalta määrättyjen sääntöjen mukaan. Prettierin avulla koodi pysyy luettavana ja helpommin ylläpidettävänä. ESLint on staattisen koodin analysointityökalu, joka auttaa korjaamaan ja tunnistamaan koodissa esiintyviä ongelmia. Se noudattaa ennalta määrättyjä sääntöjä, jotta koodi pysyy yhteneväisenä ja noudattaa parhaita käytäntöjä. [Lende 2023.]

Yksi mahdollinen haaste Prettieriä ja ESLintiä yhdessä käytettäessä on, että niillä on eri lähestymistapa koodin formatointiin, mikä voi johtaa niiden välisiin konflikteihin. Jotta Prettierin ja ESLintin yhteistyö onnistuisi helpommin, Prettieriin on olemassa ESLint-lisäosa, joka yhdistää Prettierin formatointisäännöt ESLintin sääntöihin ja varmistaa, että koodi on oikein formatoitu ja noudattaa standardeja. [Lende 2023.]

ESLintin ja Prettierin käyttö voi siis auttaa löytämään virheitä ajoissa ja parantamaan koodin laatua. Sääntöjen räätälöinti puolestaan voi varmistaa tehokkuuden ja ylläpidettävyyden [Lende 2023]. Prettierin sääntöjä mietittäessä on suositeltavaa pitää ne yksinkertaisena ja mahdollisimman vähissä, toki samalla lisäten käyttöön kaiken tarpeellisen. Molempiin kirjastoihin löytyy hyviä olemassa olevia käyttöönotto-ohjeita, joiden avulla kirjastojen ja oikeiden sääntöjen lisääminen projektiin onnistuu varmasti.

3 Projektin alustus

Uutta mobiilisovellusprojektia aloittaessa yksin tai ryhmässä on mietittävä tiettyjä projektissa käytettäviä perussääntöjä. Näistä tutkielmassa käsiteltävänä ovat tapa käyttää tyylejä ja kansioiden sekä niissä säilytettävien tiedostojen nimeämis- ja rakennekäytännöt. Näitä sääntöjä noudattamalla projekti ja tiedostot pysyvät ymmärrettävyyden kannalta parempina, kaikki pysyy yhtenäisenä, asiat näyttävät selkeämmiltä ja työskentely on tämän seurauksena helpompaa. Tässä osiossa tutustutaan myös Expoon, joka on vaihtoehtoinen komentorivityökalu projektia aloitettaessa.

3.1 Tyylien käyttö

Tyylit ovat React Nativessa oleellisia, sillä niiden avulla määritetään, miltä sovellus ulospäin käyttäjälle näyttää. Tyylien sisällyttämiseen on muutama eri tapa: sovelluksen kehittäjä voi esimerkiksi lisätä ne kyseessä olevan sivun tai komponentin kanssa samaan tiedostoon. Vaihtoehtoinen tapa on tehdä tyyleille täysin oman kansion tai vain erillisen tyylitiedoston. Tämänhetkisen tiedon valossa on hyvä kuitenkin eritellä tyylit omaan tiedostoonsa, jolloin tiedostot pysyvät sopivan lyhyinä ja kaikki komponenttikohtaiset tyylit löytyvät helposti samasta paikasta (esimerkkikoodi 1). [Kundariya 2020.]

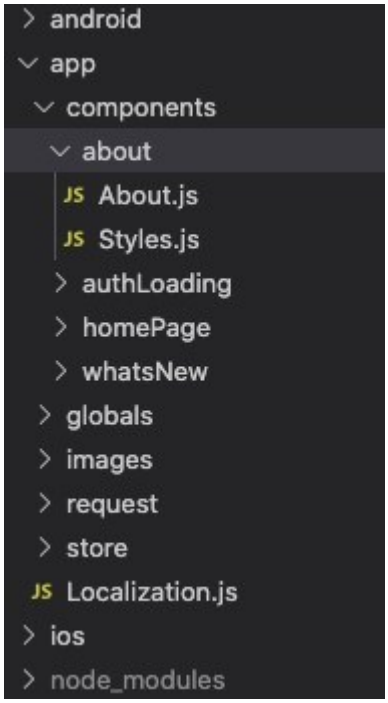
```
import ( productAmountstyles as styles ) from ‘./styles’

<View style={styles.container}>

          <Text style={styles.amountText}>{I18n.t(‘item-
          Screen.amount’}</Text>

          <View style={styles.quantityContainer}>

                    <View style={styles.quantityTextContainer}>

                              <Text style={styles.quanti-
                              tyText}>{quantity}</Text>

                    </View>

          </View>

</View>
```

Esimerkkikoodi 1. Tyylien käyttäminen koodin sisällä [Kundariya 2020].

Tyylien käyttöön liittyen yksi hyvä sääntö on välttää sisäisiä tyylejä, mikä tarkoittaa, ettei esimerkiksi yksittäisen tekstin tyyliä alettaisi muuttaa itse tekstikomponentin esittelyssä, vaan tämäkin tyyli haettaisiin tyylitiedostosta, jotta koodi pysyy yksinkertaisena ja helposti luettavana [Kundariya 2020].

3.2 Rakenne ja nimeäminen

React Nativella mobiilisovellusta kehitettäessä on oleellista pitää yhtenäinen linja nimeämisessä sekä kansiorakenteissa. Ensimmäinen oleellinen asia on luoda kaikki tiedostot app-kansioon, johon kaikille osioille tulisi luoda omat alikansionsa. Esimerkiksi komponenteille olisi olemassa komponentit-niminen kansio, jonka sisällä olisi kansio nappikomponentille, nimeltään nappiKomponentti (kuva 2).



Kuva 2. Esimerkki kansioiden ja alikansioiden käytöstä [Jabbar 2021].

Nimeämissääntöjen mukaan kansioiden ja alikansioiden nimien tulisi aina alkaa pienellä kirjaimella, kun taas tiedostot kansioiden sisällä tulisi nimetä niin sanotulla PascalCase-tyylillä, jolloin jokaisen sanan ensimmäinen kirjain kirjoitetaan isolla kirjaimella. [Jabbar 2021.]

3.3 Expo

React Native -mobiilisovelluskehityksessä on käytännössä kaksi päävaihtoehtoa komentorivityökaluja valitessa, React Nativen oma React Native CLI sekä Expo [Expo vs React Native CLI 2023]. Expo on ilmainen avoimen lähdekoodin kehys Androidissa, iOS:ssa ja verkossa toimiville sovelluksille [Expo FAQ]. Expo yhdistää parhaat puolet mobiilin ja verkon välillä mahdollistaen sovelluksen skaalaukseen ja rakentamiseen monia tärkeitä ominaisuuksia, kuten livepäivitykset, sovelluksen välittömän jakamisen ja verkkotuen [Expo FAQ]. Expo-paketti voidaan asentaa lähes mihin tahansa React Native -projektiin [Expo FAQ].

Expon tarkoitus on tarjota yksinkertaistettu kehityskokemus poistamalla joitakin React Native -projektin perustamiseen ja määrittämiseen liittyviä monimutkaisia tekijöitä. Tämä tarkoittaa esimerkiksi nopeaa aloitusta, sillä Expon avulla sovelluskehittäjän ei tarvitse huolehtia laitekohtaisista asetuksista tai natiiveista koodiriippuvuuksista. Lisäksi Expo mahdollistaa niin sanotut Over-the-air-päivitykset, joiden avulla sovelluksesta on mahdollista julkaista käyttäjille päivityksiä ilman, että käyttäjän tarvitsisi ladata uutta versiota sovelluksesta. Expolla on myös tarjolla monta esirakennettua kirjastoa ja ohjelmointirajapintaa, joiden avulla on helppo lisätä sovellukseen esimerkiksi push-ilmoitukset, sovelluksen sisäiset ostot tai kartat. Yksinkertaistettu kehityskokemus Expossa tarkoittaa myös sitä, että jotain on täytynyt jättää pois. Yksinkertaistettuun kehityskokemukseen kuuluvat esimerkiksi rajoitetut natiivimoduulit, eli Expo estää pääsyn osaan natiivimoduuleista, minkä vuoksi sovelluskehityksessä voi kohdata rajoituksia tiettyjen kolmannen osapuolen kirjastojen integroinnissa tai pääsyssä syvälle laitteen ominaisuuksiin. Expoa käytettäessä on myös vähemmän mahdollisuuksia hallita kehitysasetuksia tai mukautettuja valintoja React Native CLI:hin verrattuna. Expo on hyvä vaihtoehto, mikäli sovelluksesta halutaan yksinkertainen ja nopea prototyyppi tai sovellus ei vaadi laajaa pääsyä laitteen ominaisuuksiin. [Expo vs React Native CLI.]

Expo SDK eli Expon ohjelmistokehityspaketti tarjoaa pakettimuodossa pääsyn laitteen ja järjestelmän toiminnallisuuksiin, joita ovat esimerkiksi kamera, kontaktit ja GPS-sijainti (esimerkkikoodi 2). Paketit ovat asennettavissa komennolla npx expo install, jota käytetään esimerkiksi kameran kohdalla kirjoittamalla npx expo install expo-camera, kaikki mahdolliset vaihtoehdot ovat löydettävissä Expon dokumentaatiosta. Expo SDK paketit toimivat kaikissa React Native sovelluksissa. [Expo Reference.]

```
import { Camera } from 'expo-camera';

import * as Contacts from 'expo-contacts';

import { Gyroscope } from 'expo-sensors';
```

Esimerkkikoodi 2. Expon ohjelmistokehityspakettien käyttöönotto [Expo Reference].

Valinta Expon ja React Native CLI:n välillä on lopulta riippuvainen sovelluksen vaatimuksista. Expossa ja React Native CLI:ssä on hyvät ja huonot puolensa, mutta molemmat toimivat hyvänä pohjana mobiilisovelluskehitykseen. Aloittelevalle mobiilisovelluskehittäjälle ehdotetaan Expoa sen yksinkertaisuuden ja helppouden vuoksi. React Native CLI taas on hieman monimutkaisempi, mutta samalla hieman monipuolisempi vaihtoehto osaavalle käyttäjälle.

4 Tilanhallinta

Tila on se tieto, jonka avulla komponentit toimivat, kuten niiden on tarkoitus toimia. Tila sisältää tiedot, joita komponentti tarvitsee ja tila kertoo komponentille tiedon siitä mitä käyttäjälle halutaan esittää. Tilanhallinta on prosessi, jolla hallinnoidaan tilan päivittämistä ja sen siirtämistä komponentilta toiselle. Datan kanssa työskentely ja sen monitorointi sovelluksessa voi olla haastavaa ja on asia, joka vaikeutuu sovelluksen koon kasvaessa, minkä vuoksi tilanhallinnalle ja tilanhallintakirjastoille on tarvetta. [Ikechi 2020.]

Komponentin tila on paikallinen tietylle komponentille, eli se on saavutettavissa ja muokattavissa vain kyseisessä komponentissa. Komponentin tilaa käytetään kyseiseen komponenttiin sidoksissa olevan datan hallintaan, esimerkiksi joidenkin käyttöliittymän elementtien näkyvyyteen.

Globaali tila taas viittaa dataan, joka on saavutettavissa ja muokattavissa useista komponenteista ympäri koko sovellusta. Globaalin tilan käyttäminen on

hyödyllistä hallinnoitaessa koko sovellukselle jaettua dataa, kuten käyttäjän todennuksen tilaa tai käyttäjän asetuksia.

Globaali tila ja komponentin tila ovat molemmat oleellisia ja niillä on hyötynsä ja haittansa riippuen tilanhallinnan tarpeesta sovelluksessa. Globaali tila voi yksinkertaistaa datan hallintaa ja tehostaa suorituskykyä mutta myös monimutkaistaa sovelluksen rakennetta, jos sitä käytetään väärin. Komponentin tila on yksinkertaisempi hallita ja ymmärtää, mutta se voi mahdollisesti johtaa tietojen päällekkäisyyteen ja epäjohdonmukaiseen toimintaan sovelluksessa. [Weatherston 2023.]

Tilanhallinta on tärkeä osa kaikkia sovelluksia. Kun sovelluksessa on useita komponentteja, jotka perustuvat eri alkuperään, mutta samalla jakavat samaa tilaa sovelluskehityksen aikana, voi olla haastavaa välittää relevanttia tietoa kaikille komponenteille. Tässä tilanhallinta ja siihen liittyvät työkalut ovat avuksi, sillä tilanhallinnan avulla voi tehokkaasti jakaa ja hallita tietojen kulkua toisistaan irrallisten komponenttien välillä. [Joshi 2023.]

4.1 Paikallinen tilanhallinta

Komponenttien on usein vaihdettava käyttäjälle näkyvää tietoa vuorovaikutuksen seurauksena. Esimerkiksi jos käyttäjä painaa nappia, jonka seurauksena tapahtuu sovelluksessa haluttu asia, on sovelluksen reagoitava napin painallukseen ja muutettava sisältöä vaaditulla tavalla. Tätä toiminnallisuutta varten komponenteilla on oltava tieto näytöllä tapahtuvista interaktioista ja siitä mihin tapahtumat vaikuttavat. [State: A Component’s Memory.]

React Nativen sisäänrakennettu useState Hook -erikoistoiminto antaa sovelluskehittäjän hallita paikallista komponenttitason tilaa. Se on hyödyllinen yksinkertaisten tilojen käsittelyyn komponentissa ilman ylimääräisten tilanhallintakirjastojen tarvetta [Jorge 2023]. Esimerkkikoodissa 3 on esitetty useStaten käyttöönottoa ja käyttöä komponentin sisällä.

```
import { useState } from 'react';

export default function Counter() {

            const [count, setCount] = useState(0);

            function handleClick() {setCount(count + 1)}

            return (

            <button onClick={handleClick}>You pressed me {count}
            times</button>

            );

}
```

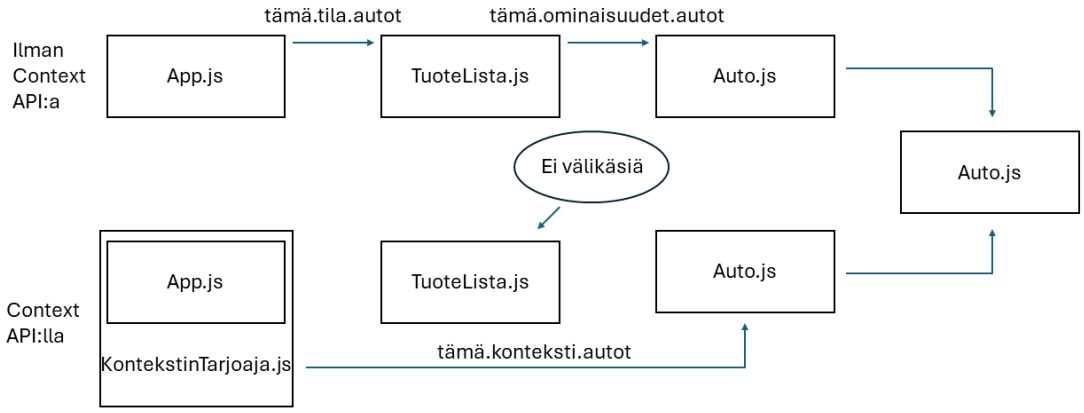
Esimerkkikoodi 3. useStaten käyttö komponentin sisällä [useState].

Työkaluna useState on hyödyllinen, kun komponentin sisällä halutaan muistaa tietoa. Käytettäessä useStatea on muistettava, että sen käyttö on pidettävä vain komponenttikohtaisena, sillä globaaliin tilanhallintaan on olemassa omat vaihtoehdot. Alustettaessa useStatea on käyttötilanteen mukaan mietittävä, tarvitseeko se alustaa tietyllä tiedolla vai esimerkiksi määrittelemättömällä arvolla, eli jättäen sen alustamattomaksi. Alustamatta jättäminen on vaihtoehto, jos tietoa halutaan esittää vasta oikean tai halutun tiedon ilmaantuessa.

4.2 Context API

React Context API on kevyt tilanhallintatyökalu, joka antaa käyttäjälle tavan luoda globaaleja muuttujia, joita voi käyttää ja välittää komponenttien välillä ympäri sovellusta [Ronen]. React Context API -tilanhallintatyökalu on sovelluksissa vaihtoehtoinen tapa, kun on tarkoitus päästä samaan dataan useasta eri komponentista, jotka sijaitsevat sovelluksessa usealla eri tasolla sen sijaan, että tieto liikkuisi välitettävänä tietona emokomponenteilta lapsikomponenteille (kuva

3) [Ronen]. Context APIa käyttämällä sovelluskoodi pysyy siistimpänä ja virheellisten komponenttien välistä muuttujien välitystä on mahdollista välttää.



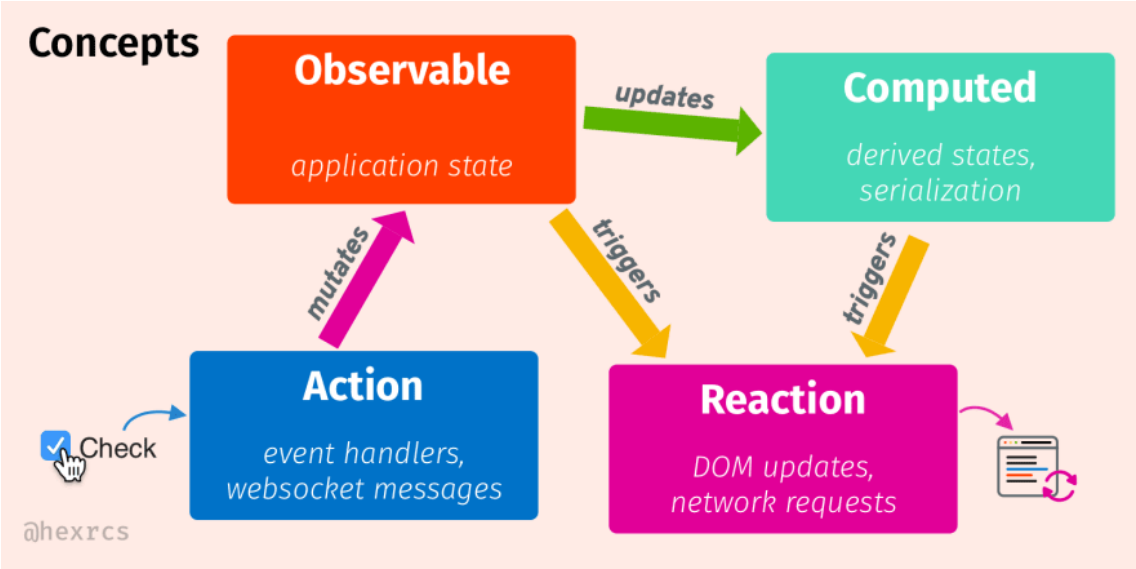
Kuva 3. Esimerkki komponenttien hierarkiasta, emokomponentilta lapsikomponentille tiedon välittäminen yläpuolella, React Context API alapuolella [Yordanov].

Context API koostuu kahdesta pääkomponentista, kontekstin tarjoajasta ja kontekstin kuluttajasta. Kontekstin tarjoaja on vastuussa tiedot sisältävän kontekstin luomisesta ja hallinnasta, jota välitetään komponenttien kesken. Kontekstin kuluttajaa taas käytetään kontekstin tietojen saavuttamiseksi, jolloin saadaan käyttöön myös komponentin tiedot. [Boateng 2023.]

Context APIa voi käyttää yleisen globaalin tiedon, kuten sovelluksen teematiedon tallettamiseen ja jakamiseen, jotta koko sovellus tietää käytössä olevan käyttäjän valitseman teeman. Muita hyödyllisiä käyttötarkoituksia ovat käyttäjän autentikointi ja kieliasetusten välitys.

4.3 MobX

MobX on tilanhallintakirjasto, jota on mahdollista käyttää kaikissa JavaScript-ohjelmistokehyksissä. MobX tarjoaa tavan varastoida ja päivittää sovelluksen tilaa, mitä käyttäen React Native hahmontaa komponentit. MobX mahdollistaa myös sovelluksen tilanhallintaa järjestelmällisellä ja reaktiivisella tavalla. MobX:n tavoite on tehdä tilanhallinnasta yksinkertaista ja intuitiivista hyödyntämällä tarkkailtavaa dataa ja automaattisia päivityksiä. [Ifeoluwa 2023.]



Kuva 4. Esimerkki MobX:n ydinkäsitteistä ja toimintaketjusta [Li 2019].

MobX:n ydinkäsitteitä on neljä (kuva 4), Observables, Actions, Reactions ja Computed Values. Observables on reaktiivisen ohjelmoinnin pohja, data, joka muuttuessaan laukaisee halutun reaktion. Mahdollisia reaktioita voivat olla esimerkiksi muutokset käyttöliittymässä tai uusi tietojen haku. Reaktiot ovat täysin määritettävissä. Tilanhallinnallisesti tämä tarkoittaa sitä, että sovelluksen tilamuuttujista tulee tarkkailtavia eli Observableja. Actions, eli toiminnot, ovat funktioita, jotka muokkaavat Observables-muuttujia. MobX:ssä ne ovat ainoa tapa muokata tilaa ja niiden avulla MobX varmistaa, että muutoksia seurataan ja oikeat reaktiot laukaistaan. Reactions, eli reaktiot, yhdistävät kaiken MobX:ssä. Ne ovat toimintoja, jotka käynnistyvät automaattisesti, kun Observablet, joista

reaktiot ovat riippuvaisia, muuttuvat. Computed Values, eli lasketut arvot, ovat dataa, jota voidaan käyttää informaation johtamiseen muista Observableista. Computed Values -arvot toimivat laiskasti tallentamalla tulokset välimuistiin ja laskemalla tulokset uudestaan vain, jos jokin taustalla olevista havainnoista muuttuu. Jos Computed Values -arvoja ei tarkkailla, ne keskeyttävät toimintonsa kokonaan. Computed Values -arvot auttavat vähentämään tallennettavan tilan määrää ja ovat hyvin optimoituja, minkä vuoksi niiden käyttö on hyvä vaihtoehto mobiilisovelluskehityksessä. [Ifeoluwa 2023.]

MobX:ää käytettäessä hyviä käytäntöjä ovat observableja muuttavien actions-toimintojen pitäminen samassa tiedostossa observable-muuttujien kanssa. Näin toimimalla vältytään samoihin asioihin liittyvien toimintojen hajauttamiselta projektin sisällä. Toisena oleellisena käytäntönä voidaan pitää pienten komponenttien käyttöä, koska observables-muuttujien muuttuminen aiheuttaa komponentin uudelleen hahmontamisen. Mitä pienempi komponentti on kyseessä, sitä vähemmän sovelluksen on piirrettävä uudelleen hahmontamisen tapahtuessa.

4.4 React Query

React Query mahdollistaa joustavan ja intuitiivisen lähestymistavan tiedonhallintaan React Native -sovelluksissa. Se on suunniteltu toimimaan monilla REST-ohjelmointirajapinnoilla tarjoamalla välimuistiin tallentamista ja muita suorituskykyyn liittyviä optimointeja, jotka voivat parantaa sovelluksen reagointikykyä. [Samman 2024.]

Hyödyntämällä React Queryn sisäänrakennettua välimuistia ja tiedonhallinnallisia ominaisuuksia on mahdollista vähentää yleisesti tiedonhallintaan käytettävän koodin määrää. React Query tarjoaa myös joukon Hook-erikoistoimintoja, joilla voi käsitellä mutaatioita, kuten tiedon lisäämistä, päivittämistä ja poistamista REST-ohjelmointirajapinnassa. Tämä edesauttaa ohjelmointirajapintaan liittyvien kyselyjen yksinkertaistamista, joka taas auttaa vähentämään ylimääräistä toistoa koodissa. [Samman 2024.]

```
function MyComponent() {

const notifyOnChangeProps = useFocusNotifyOnChangeProps()

const { dataUpdatedAt } = useQuery({

            queryKey: ['myKey'],

            queryFn: async () => {

            const response = await fetch(

            'https://api.github.com/repos/tannerlinsley/react-query',

            )

            return response.json()

            },

notifyOnChangeProps,

})

return <Text>DataUpdatedAt: {dataUpdatedAt}</Text>

}
```

Esimerkkikoodi 4. React Queryllä toteutettava datan haku [React Native].

React Queryä käytettäessä datan hakemiseen (esimerkkikoodi 4) tulokset tallennetaan paikalliseen välimuistiin, jolloin samaa dataa tietyn ajan sisällä haettaessa palautuu vanha data, jolloin vältetään tekemästä uutta ohjelmointirajapintakutsua, mikä tarkoittaa mahdollisia säästöjä maksullisia ohjelmointirajapintoja käytettäessä. Välimuisti mitätöityy automaattisesti, kun data muuttuu, joten viimeisin data on aina saatavilla. Tämän lisäksi React Query mahdollistaa datan uudelleenhaun myös sovelluksen ollessa taustalla, jos välimuistissa oleva data

on vanhaa. Siihen, milloin dataa pidetään vanhana, on mahdollista vaikuttaa, mutta käytössä oleva oletusaika on viisi minuuttia. Näiden kahden ominaisuuden yhdistelmä tekee React Querystä hyvän työkalun datan haun hallintaan tehostamalla suorituskykyä, vähentämällä ohjelmointirajapintakutsuja sekä pitäen datan ajan tasalla. [Mohan 2023.]

React Queryä käytettäessä hyvinä käytäntöinä voidaan pitää esimerkiksi kutsujen eriyttämistä omaksi Hook-erikoistoiminnokseen, jolloin samaa kutsua käytettäessä voidaan kutsua luotua Hook-erikoistoimintoa uuden kutsun kirjoittamisen sijaan. Näin koodi pysyy siistinä ja React Query kutsut helpommin hallittavina. Toisena hyvänä käytäntönä tiedonhakulogiikka on hyvä pitää erillään esityskomponenteista, jolloin sovelluksen eri osia on helpompi testata ja myös osaltaan helpottaa koodipohjan ylläpitoa. Viimeisenä käytäntönä tuodaan esille virheiden käsittely, sillä käsittelemällä virheet Hook-erikoistoimintojen sisällä on mahdollista tarjota johdonmukainen ja keskitetty tapa käsitellä virheitä sovelluksessa. Tämä voi myös parantaa käyttökokemusta esittämällä johdonmukaisia ja selkeitä virheilmoituksia käyttäjälle. [Lotfinia 2023.]

4.5 Zustand

Zustand on Jotain ja React-springin luoma Reactin avoimen lähdekoodin tilanhallintakirjasto, jonka avulla on mahdollista hallita tilaa intuitiivisesti ja yksinkertaisesti. Se on rakennettu Context-ohjelmointirajapinnan päälle ja käyttää tilanhallintaan storeja (esimerkkikoodi 5). Store on säiliö tilalle ja toiminnoille, joilla kyseistä tilaa hallitaan. Storeja on mahdollista luoda niin monta kuin on tarpeen ja niitä on mahdollista käyttää ympäri sovellusta. [L 2023.]

```
import create from 'zustand'

const useStore = create(set => ({

          count: 1,

          inc: () => set(state => ({ count: state.count + 1})})

          function Controls() {

                    const inc = useStore(state => state.inc)

                    return <button onClick={inc}>one up</button>}

          function Counter() {

                    const count = useStore(state => state.count)

                    return <h1>{count}</h1>}})
```

Esimerkkikoodi 5. Yksinkertaistettu yhden tiedoston sisällä tapahtuva esimerkki Zustand storen luomisesta ja sen käytöstä [How to use Zustand].

5 Listat

5.1 Virtuaaliset listat

Mobiilisovelluksia kehitettäessä suorituskyky ja tehokas hahmontaminen on tärkeässä osassa hyvässä käyttäjäkokemuksessa. Virtuaalisilla listoilla suurien datalistojen käsittely on mahdollista, sillä ne tarjoavat apua tehokkaammin toimivaan hahmontamiseen ja parempaan suorituskykyyn. Virtualisointi parantaa suurien listojen suorituskykyä ja vähentää muistin kulutusta ylläpitämällä aktii-

visten kohteiden rajallista hahmontamisikkunaa sekä korvaamalla tämän ikkunan ulkopuolella olevat kohteet sopivan kokoisella tyhjällä tilalla. [Rahman 2023.]

Kaikki React Nativen virtualisoidut listakomponentit pohjautuvat samaan VirtualizedList-komponenttiin ja sisältävät joko kaikki tai suurimman osan samoista ominaisuuksista. Tärkeänä ominaisuutena mainitaan initialNumToRender, jolla määritetään, kuinka monta kohdetta hahmonnetaan alkuperäisessä erässä. Valitun initialNumToRender-arvolla määritellyn määrän tulisi täyttää näyttö, mutta ei mennä paljon sen yli. Toinen ominaisuus on windowSize, jonka arvoa muuttamalla on mahdollista määrittää, kuinka monta näytöllistä kohteita hahmonnetaan. Esimerkiksi arvolla 3 hahmonnettuina on esillä olevan näytön lisäksi yksi näytöllinen näytön ylä- sekä alapuolella. Virtuaaliset listat mittaavat automaattisesti kohteiden kokoa. Kohteiden turha mittaus on mahdollista ohittaa käyttämällä samankokoisia kohteita ja syöttämällä listalle ominaisuus getItemLayout, joka voi tarjota suurta suorituskyvyn parannusta. Ominaisuuksista tuodaan esille vielä keyExtractor, jonka avulla jokaiselle listan kohteelle asetetaan ainutkertainen tekstimuotoinen muuttumaton avain, jolloin Reactin on helpompi tunnistaa muuttuneet listan kohteet. Viimeisenä tärkeänä ominaisuutena mainitaan renderItem, jonka avulla hahmonnetaan kohde listalta koodissa määritellyllä tavalla. [Rahman 2023.]

5.2 Flatlist

Flatlist on monipuolinen komponentti, joka hahmontaa tehokkaasti vieritettävän listan sille syötetystä rakenteeltaan samanlaisesta datasta, joten Flatlistin toiminnan optimoimiseksi hahmonnettavien komponenttien on tärkeää olla rakenteeltaan samankaltaisia. Flatlist tukee sekä pysty- että vaakasuuntaista vieritystä ja pitää hahmonnettuna vain näytöllä näkyvät kohteet tehden Flatlististä hyvän vaihtoehdon suurten tietojoukkojen esittämiseen suorituskykyä edelleen ylläpitäen [Exploring React Native’s Powerful List Components: FlatList, SectionList, and VirtualizedList 2023]. Flatlist tarvitsee toimiakseen kolme virtuaalisen listan ominaisuutta, jotka ovat esitettävä data, renderItem ja keyExtractor.

Myös muita virtuaalisen listan esittelyssä mainittuja ominaisuuksia on mahdollista käyttää, mikäli kokee tarvetta listan toiminnan optimoinnille.

Flatlistista on myös lisäosa nimeltään SectionList, joka mahdollistaa listan hahmontamisen kategorioituina osioina. Jokaisella osiolla on mahdollista olla oma otsake sekä alatunniste, joka on hyvä ryhmitetyn datan esityksessä. SectionList on hyödyllinen työskenneltäessä erottelua edellyttävien tietojen, kuten luokkiin järjestettyjen tuotteiden tai yhteystietojen kanssa. [Exploring React Native's Powerful List Components: FlatList, SectionList, and VirtualizedList 2023.]

Flatlistia käytettäessä on hyvä muistaa kaksi käytäntöä, jotka edesauttavat listan toimintaa. Listaa käyttäessä on hyvä välttää anonyymeja toimintoja, jotta toiminnot eivät luo itseään uudelleen jokaisella näyttökerralla. Anonyymeja toimintoja välttämällä saadaan säästettyä laitteen muistia ja prosessorin resursseja (esimerkkikoodi 6). [Arif 2023].

```
//correct way: no anonymous functions

const renderItem = ({ item, index }) => (

<Text>{index}. {item.title}</Text> );

// the wrong way. Steer clear from this:

{ data && (

<Flatlist data={data} keyExtractor={(item) => item.id} renderItem={({
item, index }) => (

<Text>{index}.{item.title}</Text>

)}/>

);}
```

Esimerkkikoodi 6. Flastlistin toimintojen nimeämiskäytännöt. [Arif 2023].

Toisena hyvänä käytäntönä on komponenttien kevyenä pitäminen, sillä mitä raskaampia komponentit ovat, sitä enemmän muistia käytetään. Listalla esitettävä tiedon määrä on hyvä pitää pienenä. Tämän avulla myös käyttöliittymä näyttää siistimmältä. Paljon tietoa sisältävien kohteiden kohdalla on mahdollista käyttää esimerkiksi sovelluksessa navigointiin tarkoitettua React Navigationia, jonka avulla käyttäjä voidaan kohdetta painamalla siirtää erilliselle sivulle, jossa lisätiedot voidaan esittää. [Arif 2023].

5.3 Flashlist

Flashlist on React Native komponentti, joka on suunniteltu korvaamaan edellä mainittu FlatList, ja siksi sen toiminnallisuudet ja ominaisuudet ovat suurelta osin samanlaisia. Ero näiden listojen välillä on, että FlatList käyttää vanhempaa virtualisointikonseptia, FlashListilla on käytössään uudempi toimintatapa, jota kutsutaan nimellä Cell Recycling. Cell Recyclingin avulla vältytään turhalta komponenttien irrottamiselta ja kiinnittämiseltä, mikä tekee hahmontamisesta tehokkaampaa kuin virtualisoinnilla toimivissa listoissa. Flashlist on rakennettu RecyclerListView-komponentin päälle, mikä tarkoittaa, että se sisältää kaikki kyseisen komponentin ominaisuudet ja hyödyt. Flashlist sisältää RecyclerListView-komponentin ominaisuuksien lisäksi uusia toiminnallisuuksia. Flashlist käyttää kierrätysnäkymien käsitettä, eli se hahmontaa ja luo vain rajoitetun määrän näkymiä, jotka näkyvät näytöllä, ja uudelleen käyttää niitä, kun käyttäjä selaa listaa. Tämä parantaa sovelluksen suorituskykyä ja vähentää muistin käyttöä. [Rahman 2023.]

RecyclerListViewin mukana FlashListiin on saatu monia ominaisuuksia, joita ovat muun muassa porrastettujen ruudukkoasettelujen tuki, kohteiden vaihtelevien korkeuksien tukeminen, horisontaali listan tuki, näkyvyystapahtumat ja alkuperäisen hahmonnuspoikkeaman sekä indeksin tuki.

Ongelmaton FlashList ei silti vielä ole, sillä kyseessä on kehitysvaiheessa oleva komponentti. Tällä hetkellä tiedettyjä ongelmia ovat varoitus käyttökelvotto-

masta hahmonnnetusta koosta, joka liittyy FlashListin käyttämän recycler-listview-ominaisuuden oletusasettelualgoritmiin, joka ei voi toimia ilman kelvollista kokoa. Tämän ominaisuuden on ensin mitattava itsensä, jonka jälkeen se voi päättää, kuinka paljon täytyy hahmontaa uutta ja mitä voi uudelleen käyttää. Tämä ongelma on ratkaistavissa lisäämällä FlashListin ympärille näkymä, jonka koko on kelvollinen, eli yli kaksi pikseliä. Edellä mainitun korjauksen lisäksi FlashListin koon on vastattava ympärillä olevan näkymän kokoa. [FlashList Known Issues 2024.]

6 Yhteenveto

React Native on vuonna 2015 julkaistu avoimen lähdekoodin käyttöliittymäohjelmointikehys, joka on työkalu natiivien mobiilisovellusten kehittämiseen ja sopii hyvin myös vasta-alkajalle. Koska React Native on hyvin suosittu työkalu, sen käyttöön on erittäin paljon hyviä oppaita internetissä. Vaikka React Native on kehitetty JavaScriptille, sen käyttö TypeScriptillä tekee tyypityksen kautta sovelluskehitysprosessista selkeämmän ja luotettavamman, niin yksin kuin joukolla työskenneltäessä. Selkeyden avulla ulkopuolisten on helpompi ymmärtää koodin toimintaa ja kehittää tarvittaessa kehittää projektia eteenpäin.

React Nativen parhaita käytäntöjä tutkiessa TypeScriptin käyttö oli yksi eniten esiintyneistä aiheista. Tämän lisäksi koin tärkeäksi käsitellä kahta muuta oleellista ilmi tullutta asiaa, joista ensimmäinen oli tyylien käyttö React Nativessa. Tyylitiedostojen sijoittamisella ja tyylien käyttötavalla voidaan selkeyttää tiedostoja paljon, mikä myös selkeyttää koodia ja helpottaa sen ymmärtämistä. Työssä parhaaksi tavaksi osoittautui tapa eritellä tyylit omaan tiedostoonsa, jolloin tiedostot säilyvät lyhyinä ja kaikki komponenttikohtaiset tyylit löytyvät aina samasta paikasta.

Luvussa 3 käsitelty tärkeitä käytännön asioita olivat tiedostorakenne ja nimeäminen React Nativessa, sillä nämä asiat vaikuttavat myös projektin yleisilmeeseen, tiedostojen löydettävyyteen ja selkeyteen. Tiedostorakenteessa ja nimeämisessä tärkeimmiksi muodostuivat kaiken sisällyttäminen app-kansion sisälle,

jossa jokainen aihealue saa omat kansionsa, esimerkiksi komponenteille kansio “komponentit”, jonka sisään jokainen komponentti saa oman alikansionsa. Nimeämistyylejä on olemassa muutamia, mutta tutkielmassa käsiteltävinä vaihtoehtoina esiin nousivat kansioiden nimeäminen pienellä alkukirjaimella ja niiden sisältämien tiedostojen nimeäminen PascalCase-tyylillä, jolloin nimen ensimmäinen kirjain ja jokaisen nimessä esiintyvän sanan ensimmäinen kirjain kirjoitetaan isolla kirjaimella.

Alaluvussa 3.3 käsitelty Expo vaikuttaa varteen otettavalta vaihtoehdolta Expon ollessa nopea ja yksinkertainen tapa aloittaa React Native -projekti. Expon parhaalta ominaisuudelta vaikuttaa mahdollisuus julkaista käyttäjille sovelluspäivityksiä tekemättä versiopäivitystä sovelluskauppoihin, mikä mahdollistaa kiireellisiä virhekorjauksia.

Luvussa 4 käsiteltiin tilanhallintaa, mikä liittyy sovelluksen sisällä komponenttien välillä jaettavaan dataan ja sitä kautta käyttäjälle sovelluksessa näkyvään reaktiivisuuteen, mitä sovelluksessa halutaankaan esimerkiksi napin painalluksesta tapahtuvan. Tilanhallinnallisia työkaluja on olemassa lukuisia, joista käsiteltäviksi otettiin monissa oppaissa toistuvasti esiin tulleet hyvät sekä kevyehköt vaihtoehdot. Yksi vaihtoehdoista on Zustand, josta on vielä vähän tietoa. Zustand vaikuttaa kuitenkin hyvältä vaihtoehdolta ja se luultavasti yleistyy myös yritysmaailmassa.

Tutkittavana tilanhallinnallisista työkaluista olivat paikallinen tilanhallinta, eli React Nativen oma useState Hook -erikoistoiminto, joka toimii parhaiten hallitessaan tilaa paikallisesti, eli esimerkiksi omassa komponentissaan. useState on kevyt ja hyödyllinen ja omaan tehtäväänsä sopiva työkalu. Alaluvussa 4.2 tutkittiin Context API -tilanhallintatyökalua, joka on kevyt tapa luoda globaaleja muuttujia, joita on mahdollista käyttää ja välittää komponenteissa ympäri sovellusta. Context API -tilanhallintatyökalu toimii hyvin yleisen tiedon välittämiseen, kuten teeman valintaan tai kieliasetuksiin.

Alaluvussa 4.3 käsiteltiin MobX:ää, joka store-varastojen muodossa tuo tilanhallintaan enemmän rakennetta ja mahdollistaa esimerkiksi omille observable-muuttujilleen luotavia globaaleja funktioita. MobX on ollut käytössä useassa eri projektissa ja on kokemuksien perusteella hyvä ja riittävän kevyt vaihtoehto paikallisen tilanhallinnan ja soveltuu Context API:n kanssa sovelluksen tilanhallintaan. Tutkielmassa nostettiin esille myös React Query, joka on olennainen vaihtoehto datan hakemiselle rajapinnasta, sillä se osaa hoitaa tiedon tallennuksen välimuistiin, jolla vältetään ylimääräisiä pyyntöjä ja parhaassa tapauksessa jopa rahaa.

Viimeisenä luvussa 5 esiin nostettiin listat, joita olivat virtuaalisiin listoihin pohjautuva FlatList ja uutena vaihtoehtona esiin noussut FlashList. FlatList on tutumpi vaihtoehto ja toiminut käytännön työssä hyvin käsiteltäessä isoja määriä kuvia, tekstiä tai lyhytvideoita. FlashList uutena komponenttina on mielenkiintoinen. Flashlist on pohjimmiltaan FlatListia paremmin suoriutuva ja nopeammin toimiva vaihtoehto. FlashList kannattaa siis pitää mielessä, kun vertaillaan sovelluksessa käyttöönotettavia listavaihtoehtoja.

Lähteet

Adedotun, Adebiyi. 2023. React Context API: A deep dive with examples. Verkkoaineisto. Logrocket. <https://blog.logrocket.com/react-context-api-deep-dive-examples/>. 24.3.2023. Luettu 20.2.2024.

Arif, Hussain. 2023. A deep dive into React Native FlatList. Verkkoaineisto. Logrocket. <https://blog.logrocket.com/deep-dive-react-native-flatlist/>. Päivitetty 23.5.2023. Luettu 29.3.2024.

Boateng, Dickson. 2023. How to Use the React Context API in Your Projects. Verkkoaineisto. Freecodecamp. <https://www.freecodecamp.org/news/context-api-in-react/>. 29.3.2023. Luettu 24.2.2024.

Eisenmann, Bonnie. 2017. Learning React Native. Verkkoaineisto. O'Reilly Media. <https://www.oreilly.com/library/view/learning-react-native/9781491929049/ch01.html>. 1.11.2017. Luettu 24.9.2022.

Exploring React Native's Powerful List Components: FlatList, SectionList, and VirtualizedList. 2023. Verkkoaineisto. Medium. <https://medium.com/@mujaffarhssn/exploring-react-natives-powerful-list-components-flatlist-sectionlist-and-virtualizedlist-1c47618be904>. 10.7.2023. Luettu 29.3.2024.

Expo FAQ. Verkkoaineisto. Expo Documentation. <https://docs.expo.dev/faq/>. Luettu 20.3.2024.

Expo Reference. Verkkoaineisto. Expo Documentation. <https://docs.expo.dev/versions/latest/>. Luettu 20.3.2024.

Expo vs React Native CLI. 2023. Verkkoaineisto. Medium. <https://medium.com/@softworthsolutionspvtltd/expo-vs-react-native-cli-7e47c7630039>. 27.6.2023. Luettu 20.3.2024.

FlashList Known Issues. Verkkoaineisto. FlashList Documentation. <https://shopify.github.io/flash-list/docs/known-issues/>. Luettu 30.3.2024.

History of JavaScript. 2023. Verkkoaineisto. GeeksforGeeks. <https://www.geeksforgeeks.org/history-of-javascript/>. Päivitetty 15.5.2023. Luettu 17.5.2024.

How to use Zustand. Verkkoaineisto. Zustand Documentation. <https://docs.pmnd.rs/zustand/getting-started/introduction>. Luettu 25.3.2024.

Ikechi, Fortune. 2020. Using Mobx As A State Manager In React Native Applications. Verkkoaineisto. Smashing Magazine. <https://www.smashingmagazine.com/2020/08/mobx-state-manager-react-native-applications>. 26.8.2020. Luettu 20.3.2024.

Ifeoluwa, Covenant. 2023. “Simplifying State Management in React Native with MobX: A Developer’s Guide to Seamless App Control”. Verkkoaineisto. Medium. <https://medium.com/@covenantcodes/simplifying-state-management-in-react-native-with-mobx-a-developers-guide-to-seamless-app-b750ce64d32b>. 28.8.2023. Luettu 1.3.2024.

Jabbar, Gilshaan. 2021. React Native Coding Standards and Best Practices. Verkkoaineisto. Medium. <https://gilshaan.medium.com/react-native-codingstandards-and-best-practices-5b4b5c9f4076>. 10.3.2021. Luettu 9.10.2022.

Jorge, Ivan. 2023. Simplifying State Management in React Native: Harnessing Efficiency with CronJ. Medium. <https://medium.com/@livajorge7/simplifying-state-management-in-react-native-harnessing-efficiency-with-cronj-23834c55f109>. 28.7.2023. Luettu 18.3.2024.

Joshi, Darshan. 2023. React Native State Management: Importance, Libraries, and Examples. Verkkoaineisto. Bacancy. <https://www.bacancytechnology.com/blog/react-native-state-management>. Päivitetty 29.12.2023. Luettu 21.1.2024.

Kundariya, Harikrishna. 2020. React Native Best Practices Every Developer Should Know. Verkkoaineisto. eSparkBiz. <https://www.esparkinfo.com/blog/react-native-practices-developer-should-follow.html>. Päivitetty 30.12.2022. Luettu 28.9.2022.

Kylmäniemi, Ilkka. 2023. Toolchain – JavaScript. Opintomateriaali. Metropolia Ammattikorkeakoulu. 11.9.2023. Luettu 14.4.2024.

L, Joris. 2023. [Tutorial] Zustand: A Simple and Powerful State Management Solution. Verkkoaineisto. Medium. <https://medium.com/@joris.l/tutorial-zustand-a-simple-and-powerful-state-management-solution-9ad4d06d5334>. Luettu 3.3.2024.

Lende, Saurabh. 2023. Improving Code Quality in React with ESLint, Prettier, and TypeScript. Medium. <https://medium.com/globant/improving-code-quality-in-react-with-eslint-prettier-and-typescript-86635033d803>. 26.6.2023. Luettu 8.4.2024.

Li, Xiaoru. 2019. React + MobX Crash Course (in 5 pics!). Verkkoaineisto. Dev.to. <https://dev.to/methodcoder/react-mobx-crash-course-in-5-pics-2m3b>. Päivitetty 7.1.2020. Luettu 20.2.2024.

Lotfinia, Majid. 2023. React-Query Best Practices: Separating Concerns with Custom Hooks. Verkkoaineisto. Medium. <https://majidlotfinia.medium.com/react-query-best-practices-separating-concerns-with-custom-hooks-3f1bc9051fa2>. 19.3.2023. Luettu 4.4.2024.

Mohan, Kiran. 2023. React Query: A Comprehensive Guide. Verkkoaineisto. Medium. <https://medium.com/@KiranMohan27/react-query-a-comprehensive-guide-1e89ddc3bcef>. 9.7.2023. Luettu 1.3.2024.

Rahman, Anisur. 2023. React Native — Ultimate Guide on Virtualization Performance Optimization. Verkkoaineisto. Medium. <https://medium.com/@anisurrahmanbup/react-native-virtualization-performance-optimization-flatlist-section-list-virtualizedlist-8430da4c68b3>. 21.9.2023. Luettu 28.3.2024.

Rahman, Anisur. 2023. React Native — Ultimate Guide on FlashList: Performant List view (Implementation + Analysis). Verkkoaineisto. Medium. <https://medium.com/@anisurrahmanbup/react-native-flashlist-performant-list-view-implementation-analysis-8b29df8f2560>. 27.9.2023. Luettu 30.3.2024.

React Native. Verkkoaineisto. TanStack Query. <https://tanstack.com/query/latest/docs/framework/react/react-native>. Luettu 1.3.2024.

Ronen, Eylon. 2023. React Context API: What is it and How it works?. Verkkoaineisto. Loginradius. <https://www.loginradius.com/blog/engineering/react-context-api/>. 9.8.2023. Luettu 20.2.2024.

Samman, K. 2023. Simplify Data Management in React Native. Verkkoaineisto. Medium. <https://medium.com/@mr.kashif.samman/simplify-data-management-in-react-native-with-react-query-36f264d253a7>. 28.2.2023. Luettu 1.3.2024.

State: A Component's Memory. Verkkoaineisto. React Documentation. <https://react.dev/learn/state-a-components-memory>. Luettu 25.3.2024.

The history of React Native: Facebook's Open Source App Development Framework. 2020. Verkkoaineisto. TechAhead. <https://www.techaheadcorp.com/blog/history-of-react-native/>. Päivitetty 16.11.2023. Luettu 24.9.2022.

useState. Verkkoaineisto. React Documentation. <https://react.dev/reference/react/useState>. Luettu 21.1.2024.

Weatherston, Grant. 2023. How to Manage State in React and React Native with the PullState Library. Verkkoaineisto. Freecodecamp. <https://www.freecodecamp.org/news/state-management-in-react-and-react-native-with-pullstate>. 3.4.2023. Luettu 21.1.2024.

What is JavaScript?. 2024. Verkkoaineisto. GeeksforGeeks. <https://www.geeksforgeeks.org/what-is-javascript/?ref=ml_lbp>. Päivitetty 20.2.2024. Luettu 17.5.2024.

What is JavaScript used for?. 2021. Verkkoaineisto. Hack Reactor. <https://www.hackreactor.com/blog/what-is-javascript-used-for>. 26.8.2021. Luettu 17.5.2024.

What is TypeScript?. Verkkoaineisto. TypeScript Tutorial. <https://www.typescripttutorial.net/typescript-tutorial/what-is-typescript/>. Luettu 28.9.2022.

Yordanov, Boris. Working With the React Context API. Verkkoaineisto. Toptal. <https://www.toptal.com/react/react-context-api>. Luettu 4.5.2024.

7 Best Practices for React Native Applications. 2021. Verkkoaineisto. In Plain English. <https://javascript.plainenglish.io/7-best-practices-for-react-native-applications-be1dd907e657>. 28.2.2021. Luettu 28.9.2022.