HAMK
Hämeen ammatti-
korkeakoulu

SOVELLUKSEN KÄYTTÖLIITTYMÄN KEHITTÄMINEN ÄLYPUHELIMILLE

Ammattikorkeakoulututkinnon opinnäytetyö

Tieto- ja viestintätekniikka, insinööri (AMK)

Syksy, 2024

Akin Lawrence

HAMK
Hämeen ammatti-
korkeakoulu

Tieto- ja viestintätekniikka Tiivistelmä
Tekijä Akin Lawrence Vuosi 2024
Työn nimi Sovelluksen käyttöliittymän kehittäminen älypuhelimille
Ohjaaja Toni Laitinen

Työn tavoitteena oli kehittää käyttöliittymää mobiilisovellukselle, joka on suunnattu verkostoitumiselle. Tässä opinnäytetyössä käytiin tärkeimmät asiat ja vaiheet, jotka liittyvät yleisesti React Native -sovelluskehitykseen. Itse projektissa haluttiin helpottaa sosialisoitumista ja tapahtumien etsintää, jotta siitä tulisi vaivatonta. Ohjelmiston tehtävänä on yksinkertaistaa tätä prosessia asettamalla asiat sellaiseen järjestykseen, että jokaisessa kaupungissa jokainen tietäisi välittömästi meneillään olevista tapahtumista ympäri Suomea.

Opinnäytetyötä toteutettiin toiminnallisena tutkimuksena, jossa hyödynnettiin erilaisia ohjelmointityökaluja. Työnteon yhteydessä selvisi paljon uutta tietoa tiedon käsittelystä ja hallitsemisesta. Työn aikana käytiin asioita läpi, jotka tekevät projektista pitkäkestoisen ja myöhemmin helposti päivitettävän.

Johtopäätöksenä selvisi, että jokaista JavaScript käyttöliittymä koodia kannattaa päivittää tietyin aikavälein. Päivitysten tekeminen ratkaisee ajan myötä tulevia haavoittuvuuksia ja se helpottaa ominaisuuksien lisäämistä sitä mukaan, kun niitä pitää integroida projektiin.

Avainsanat JavaScript, Käyttöliittymä, Mobiilisovellus, React Native
Sivut 46 sivua

HAMK
Hämeen ammatti-
korkeakoulu

Information and Communication Technology
Author Akin Lawrence
Subject User Interface Development for a Mobile Application
Supervisor Toni Laitinen

Abstract
Year 2024

The goal of the work is to create a user interface for a mobile application that facilitates connections with others. This thesis covers the most important aspects of user interface development when using React Native and steps that are generally related to the development environment. The idea itself of the application is to make it quick to search for all events, so that it becomes easy and effortless. The software accomplishes this by simplifying the process of meeting others by organizing events in an order which allows users to be instantly aware about the ongoing events in cities in Finland.

During this thesis various programming tools were utilized to get more experience. A wealth of new information about data processing and management was learned. Some steps were also taken to make sure the projects user interface is as long-lasting as possible and easily updatable.

The conclusion was that the JavaScript code should be updated at certain intervals. Making updates resolves vulnerabilities and makes it easier to add features as they come.

Keywords JavaScript, mobile application, React Native, User interface
Pages 46 pages

Sisällys

Kuvat, taulukot ja kaavat

1 Johdanto

Tavoitteena on tehdä käyttöliittymää sovellukselle, jolla on suuri potentiaali kasvaa sisällöllisesti ja tavoittaa mahdollisimman paljon ihmisiä. Tästä syystä sovelluksen käyttöliittymän kehityksessä huomioidaan pääosin älypuhelimia, sillä nykyään lähes kaikilta löytyy Android tai IOS käyttöjärjestelmää käyttävää puhelinta. Tarkoituksena on myös näyttää mobiilisovelluksen luontivaiheet ja analysoida sitä kokonaisuutena. Työtä on kehitetty melko toiminnalliseksi, jotta sitä voidaan kokeilla.

Itse työn ideana on toteuttaa uusi sosiaalimedia, joka näyttää kaikkia tapahtumia vaivattomasti vain yhdessä sovelluksessa. Sovelluksessa voidaan selata kaupungeittain eri tapahtumia, jotka perustuvat omiin kiinnostuksiin ja harrastuksiin. Tällä tavoin henkilö voi helposti tutustua muihin ihmisiin, vaikka menisi kokonaan vieraaseen kaupunkiin. Tämä helpottaa verkostoitumista henkilöille, sillä usein ihmiset pystyvät samaistumaan helpommin kiinnostuksien parissa. Sovellus lisäksi antaa jokaiselle oikeuden luoda omia tapahtumia, joihin kuka tahansa voi osallistua. Ilmoittautuminen tapahtuu tykkäyksellä, jota tilastoidaan ja sen avulla pystytään tunnistamaan suosittuja tapahtumia. Samaan aikaan ilmoittautuneiden ihmisten kiinnostusten avulla tehdään julkaisu suosituksia.

Työssä käydään teoriaosuudessa pääosin React ja React Nativeen liittyviä tärkeitä aiheita ja esimerkkejä sekä projekti kehitykseen käytettyjä asioita. Tarkoituksena ei ole käydä kaikkia asioita kokonaan läpi, vaan tätä työtä on rajattu vain käyttöliittymän olennaisiin asioihin, jotta projektin kehitystä ja sen taustaa pystyttäisiin ymmärtämään. Pääosin projektissa käytetään JavaScriptiä. Työssä ei käydä syvällisesti palvelimenkehitystä. Lopputuloksen laadusta, vaikeuksista ja mahdollisista jatkokehityksestä käydään läpi. Käyttöliittymän ominaisuuksista ja toiminnoista näytetään esimerkkejä.

2 Sovelluskehityksen tavoitteet

2.1 Toiminnallinen vaatimus

Sovelluksen toiminnallisuuteen kuuluu REST APIn eli Representational State Transfer Application Programming Interfacen käyttö. Tällä tekniikalla otetaan palvelimeen yhteyttä HTTP eli Hypertext Transfer Protocolin avulla ja sovellusta ei ole liitetty palvelimeen MVC (Model View Controller) tavoin. MVC tekniikassa käyttöliittymä sekä palvelinkoodi ovat yleensä samassa projektissa ja ne eivät toimi itsenäisesti. Joka tapauksessa RESTia hyödyntäessä saadaan sovelluksesta helposti skaalattavan, kun halutaan jatkaa projektia muihin alustoihin kuten web- tai vaikka konsolisovellukseen.

REST API standardi on kehitetty 2000-luvun alkuvaiheessa ja sen ideana on ottaa käyttöön CRUD eli Create, Read, Update ja Delete menetelmää, jolla tietoa käsitellään. Yhdellä URIlla voi olla monta Endpointtia, jotka ovat kuten linkkipolkuja, jossa käsitellään yleensä vain yhden resurssin tietoja. Palvelimen Endpointtien luomisen jälkeen pystytään lähettämään niihin esimerkiksi HTTP-metodit POST, GET, PUT ja DELETE. Halutessa resurssien käyttöä voidaan rajata vaatimalla käyttäjältä tietoa, joita pitää sisällyttää pyyntöjen kanssa. Asiakaspuoli saa aina vastauksen pyynnön jälkeen, joka saattaa sisältää yleisesti JSON tai XML muotoista tietoa. (IBM, n.d.) Sen lisäksi projektin REST APIn takana hyödynnetään SQL eli Structured Query Language, jonka avulla tehdään hakuja, tietojen tallennuksia sekä kaikki tietojenkäsittelyyn ja etsintään liittyviä asioita. Lähes kaikki sovelluksen sisällöt pystytään muokkaamaan palvelimen kautta, jolloin esimerkiksi jopa etusivun sisällöt päivittyvät uusimpaan tietoon.

JSON on JavaScript Object Notation, jonka tehtävänä on muotoilla objekti dataa tekstiksi, jotta sitä voidaan lähettää esimerkiksi API-pyynnön kautta kommunikointia varten toiselle laitteelle. Muunnoksen jälkeen JSON-teksti on järjestelmällisessä muodossa, jotta sitä voidaan muuntaa takaisin objektiksi ja kohdella kuten normaali objekti. JSON tukee vain nämä tietotyypit taulukko, objekti, numero, teksti, Boolean ja Null. Vielä lisäksi useilla ohjelmointikielillä on jo valmiiksi tuettuna tai helposti saatavilla valmis JSON-muuntaja, jolla voidaan hoitaa muunnokset automaattisesti valmiina olevilla funktioilla. Alla on esimerkki JSON-muotoisesta datasta, jota palvelimeen voidaan lähettää kirjautumista varten Endpointtiin ”/api/auth/local/”.

```
{"identifier": "oppari", "password": "salaisuus2024"}
```

Projektin palvelimeen saadaan yhteyttä taulukoissa 1,2 ja 3 esiintyvien polkujen avulla. JSON-muotoista dataa palvelin vastaanottaa sekä palauttaa takaisin asiakaspuoleiselle sovellukselle, jos tietoa on palautettavana. Lähes jokainen polku vaatii Token-avainta, sillä sen avulla rajoitetaan resurssin käyttöä tietyille käyttäjille. Kirjautumispyynnön avulla käyttäjä saa voimassa olevan Token-avaimen, jota kuuluu säilyttää muita pyyntöjä varten. Token-avain sisältää käyttäjä ID-tunnuksen, jonka avulla tunnistetaan avaimen omistajaa ja pystytään esimerkiksi ”/api/users/me” -polusta saamaan kirjautuneen käyttäjän tietoa vastaan. Token-avainta sisällytetään HTTP-Headerin kanssa, joten se tulee aina mukana automattisesti, kun tehdään pyyntöjä.

Taulukko 1 Projektin palvelimessa olevat Event-tapahtuman Endpointin-polut ja siihen liittyviä polkuja.

Method-tyyppi	Endpoint-polut	Tehtävä (Järjestyksessä Endpointin kanssa)
GET	/api/events/:id /api/event/suggest /api/events /categories/:id?populate=events	- Hakee kaikista tapahtumista julkaisun tietyn id-parametrin mukaan. - Hakee ehdotettua julkaisua ja suodattaa kaikki ei tykätyt pois sekä laittaa suosituin ensin käyttäjä kohtaisesti. - Listaa kaikki tapahtumat. - Listaa kaikki tapahtumat ja voidaan hakea tietyillä rajauksilla tai kategorioilta tietoa.
POST	/api/event/upload/:id /api/event/create	- Julkaisee kuvat palvelimelle FormData ja Endpointin ID-parametrissa pitää määrittää oman julkaisuntunnusta. - Luoo tapahtuman.
DELETE	/api/event/my/:id	- Pystytään poistaa tiettyä omaa julkaisua.
PATCH	/api/event/my/:id	- Pystytään muokkaamaan tiettyjä kohtia omista julkaisuista ID:n perusteella.

Taulukko 2 Käyttäjätilin Endpointit.

Method-tyyppi	Endpoint-polut	Tehtävä (Järjestyksessä Endpointin kanssa)
GET	/api/users/me	- Saadaan vastauksena omat tiedot.
POST	/api/auth/change-password /api/auth/local/register /api/auth/local/	- Salasanaa muutetaan ja tarkastetaan ensin, jos se on sopiva. - Rekisteröidään laittamalla omat tiedot nimi, ikä, salasana, sähköposti ja niin edelleen. -Kirjautumistietoja palautetaan ja takaisin saadaan Token-avain.
PATCH	/api/users/my	- Pystytään muuttamaan omia käyttäjäkohtaisia tietoja.

Taulukko 3 Tykkäykseen liittyvät Endpointit.

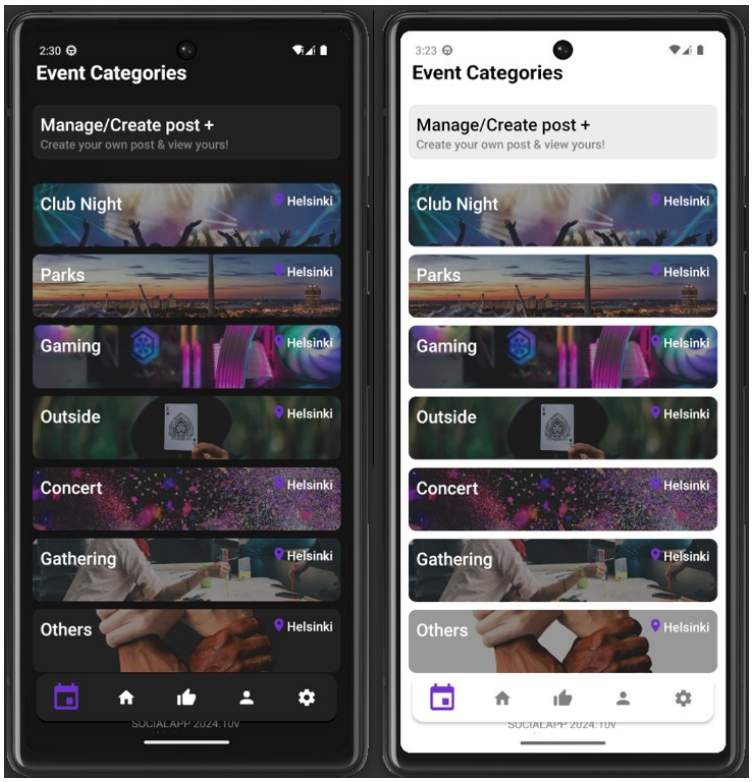
Method-tyyppi	Endpoint-polut	Tehtävä (Järjestyksessä Endpointin kanssa)
GET	/api/dislike/:id /api/like/:id /api/likes/my	- Id mukaan ”ei tykkäys” lisätään, jolloin ehdotusta ei enää tule. - Id mukaan tykkäys lisätään, jolloin julkaisu menee omiin julkaisuihin - Haetaan tällä kaikki omia tykkäyksiä
DELETE	/api/like/my/:id	- Otetaan tykkäys pois ja ehdotus julkaisusta saattaa tulla kyseistä tapahtuman ID:stä takaisin käyttäjälle.

2.2 Sovelluksen käyttöliittymä esittely

Käyttöliittymän vaatimuksiin kuuluu pääosin helppokäyttöisyys, nykyaikainen näkymä sekä mahdollisimman yksinkertainen navigointi ympäri sovellusta. Tarkoituksena on tehdä mahdollisimmalle monille mukavan tuntoisen kokemuksen. Tätä on pääsääntöisesti yritetty animaatioiden avulla, jotta kokemus olisi sulava ja nopean tuntoista. Sovelluksen pää näkymät ovat seuraavat listassa:

- Tapahtuma hakemisto näkymä (Kuva 1):
    - Voidaan hakea eri kategorioista tapahtumia
    - Hakemiston juuressa on nappi, joka johtaa omien tapahtumien luomiseen ja hallintaan. Omiin tapahtumiin voidaan lisätä kuvia korkeimmillaan viisi, otsikoita, kuvauksia ja päivämääriä kyseisistä julkaisuista. Tässä näkymässä on pakollisia kenttiä, joita pitää täyttää. Kategoriat ovat määritelty jo ennestään ja niistä voi valita vain olemassa olevia.

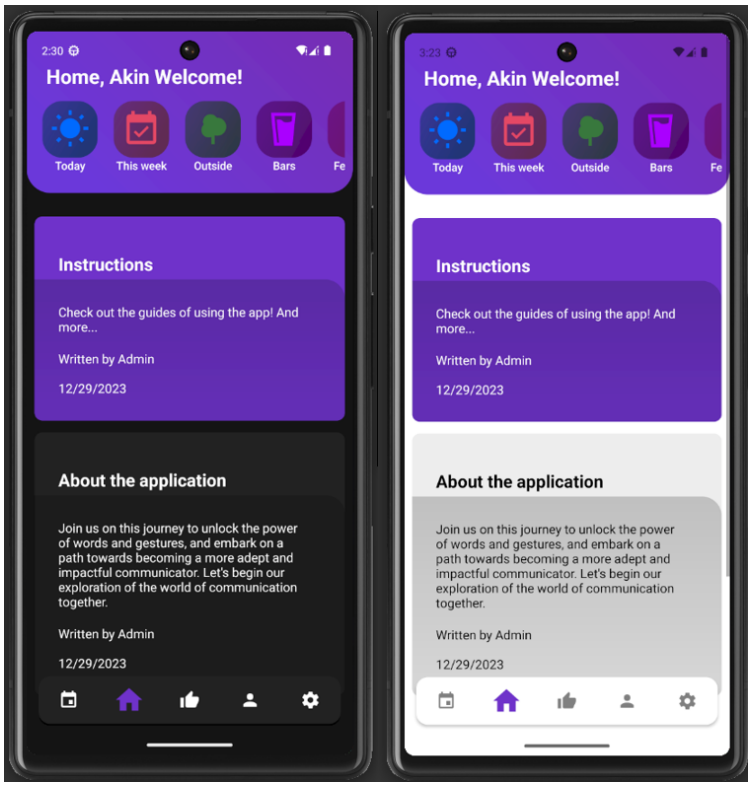
Kuva 1 Tapahtuma näkymä, jossa on eri kategorioita.



- Koti näkymä (Kuva 2):
    - Ylhäältä löytyy vieritettäviä pikakuvakkeita, jotka lyhentävät navigointia sovelluksessa. Eri kuvakkeet antavat eri vaihtoehtoja ja osa johtaa suoraan vain kategorioihin, josta saa listan tapahtumista valittuun kategoriaan.

- Tämän jälkeen alhaalta löytyy uutisia, joita järjestelmävalvoja voi muuttaa tietokannan kautta. Etusivu automaattisesti hakee uusia uutisia ja päivittää niitä näytölle.

Kuva 2 Koti näkymä, joka on oletuksena, kun sovellusta avataan.

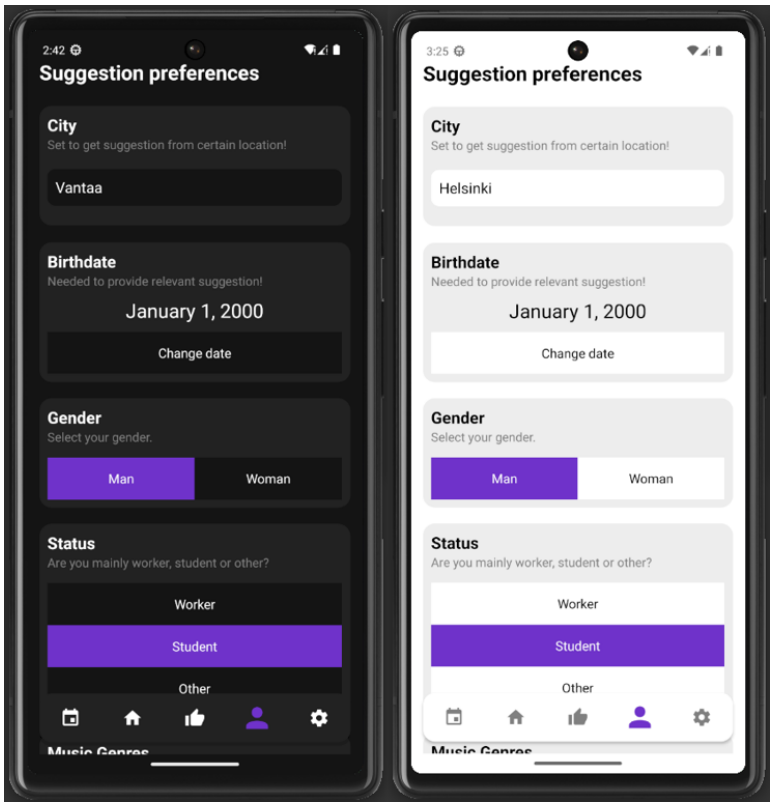


- Tapahtuma suositus näkymä (Kuva 3):
    - Ensimmäisenä näkyy ehdotettu tapahtuma, joka sisältää otsikon, lisätiedon, ajankohdan sekä julkaisija tiedon. Lisäksi kolme nappia ovat suunnilleen näytön keskellä. Nämä napit antavat mahdollisuuden katsoa tilastoa, kommentoida ja jakaa eteenpäin. Käyttäjä voi painaa tykkäystä tai ei tykätä julkaisusta, jolloin uudet ei nähdyt julkaisut ehdotetaan käyttäjän henkilökohtaisilla asetusmääritelmillä.
    - Tämän näkymän yläpalkin oikealta löytyy tykkäys historia välilehti, joihin tulevat kaikki aiemmin tykätyt julkaisut. Käyttäjä samalla voi poistaa tykätyn julkaisun omasta historiastaan.

Kuva 3 Tapahtuma suositus näkymä.

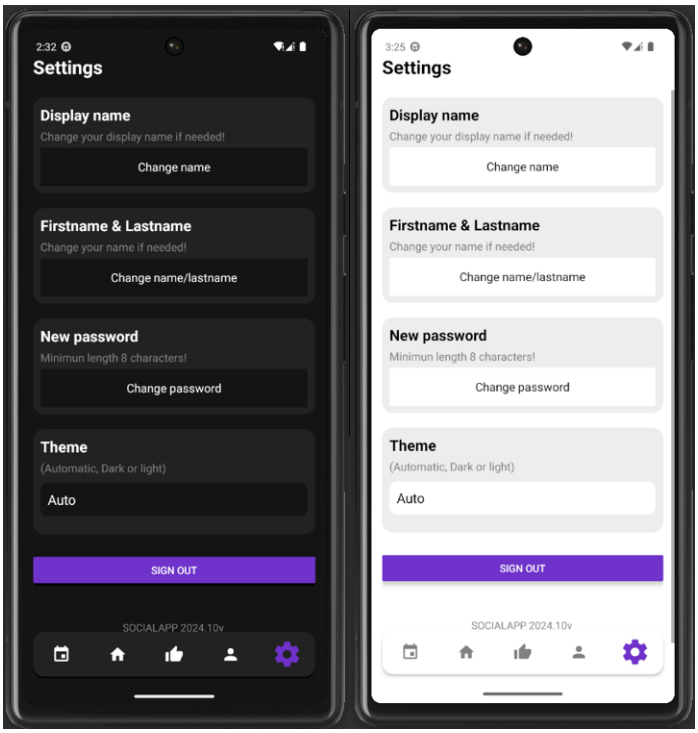
- Henkilökohtaiset asetukset näkymä (Kuva 4):
    - Käyttäjä voi asettaa haluamansa kaupungin, joista julkaisut tulevat. kun vierittää näkymää alas niin seuraavina tulevat syntymäpäivä, sukupuoli tämänhetkinen elämäntilanne, lempimusiikki ja vapaa-ajan aktiviteetit. Neljä viimeistä listattuna ovat valmiiksi määritelty monivalinnat, joista voi valita joko monta tai yhden vaihtoehdon.

Kuva 4 Henkilökohtaiset asetukset näkymä.



- Yleiset asetukset (Kuva 5):
    - Käyttäjä nimi, etu- ja sukunimi, salasana ja teema asetuksia voidaan säätää täältä. Uloskirjautumisnappi löytyy vierittämällä näkymää pohjaan asti.
    - Salasanan vaihto näkymässä validoidaan kenttiä ennen lähettämistä, jos jostain syystä palvelulla tapahtuu virhe, niin tilanne näytetään viestillä käyttäjälle. Onnistuessa myös viestiä palautetaan ja automaattisesti näkymää suljetaan. Samat asiat pätevät muihin tallennettaviin kenttiin.

Kuva 5 Asetukset näkymä.



- Kirjautumis- näkymä (Kuva 6):
    - Käyttäjä voi kirjautua kenttien täyttämisen jälkeen, jolloin saattaa tulla virhe ilmoitusta, jos käyttäjää ei ole olemassa tai salasana on väärin.
    - ”Register” napin avulla pystytään rekisteröitymään, jolloin ponnahtaa uusi ikkuna, jossa pyydetään täyttämään omia tietoja.

Kuva 6 Kirjautumis- näkymä.

3 JavaScriptin tärkeämmät aiheet Reactin kannalta

3.1 ES ja JSX

JSX on Metan kehittämä ja sen nimi tulee lyhenteestä JavaScript XML, joka laajentaa JavaScriptiä. Tämä mahdollistaa HTML kaltaisten koodien kirjoittamista esimerkiksi muuttujiin, joka näkyy esimerkissä 1. Tyypillisesti tätä laajennusta ei voida suoraan käyttää, sillä selain ei ymmärrä, jos sitä ei käännetä luettavaan muotoon. Kuvassa näkyy mitä kääntäjä Babel tekee käännöksen jälkeen React koodille (Esimerkki 2), jotta se voisi toimia suoraan selaimessa. JavaScript XML on tärkeä osa Reactissa, koska se mahdollistaa merkintäkielen, tyylin ja logiikan kirjoittamista samalle tiedostolle. Ilman sitä perinteisesti pitäisi kirjoittaa niitä erillisiin osioihin tai tiedostoihin. (Kinsta, 2023)

Esimerkki 1 JSX:n laittaminen JavaScriptin muuttujaan.

```
const heading = <h1>Hello, JSX</h1>;
```

(Kinsta, 2023)

Esimerkki 2 Käännetty JSX koodin palauttama funktio

```
const element = React.createElement("h1", null "Hello, world!")
```

(Kinsta, 2023)

JavaScript XML optimoituu automaattisesti koodin kääntyessä ja se helpottaa modulaaristen koodien kirjoittamista, jota React hyödyntää komponenttien luomisessa. Komponentit ovat koodin osia, joissa voi olla eri parametreja ja ne pystyvät toimimaan itsenäisesti. Lähes kaikista käyttöliittymän osista tehdään oma komponentti, joka mahdollistaa uudelleen käyttämistä ympäri projektia. Tämä kirjasto helpottaa myös virheiden ja varoitusten etsinnässä, koska sen syntaksi on helposti ymmärrettävää. (Mohan, 2023)

ES eli ECMAScript on JavaScriptin käyttämä standardi, jonka ECMA International on luonut. Tämä standardi määrittää yksityiskohtia, sääntöjä sekä ohjeita skriptin luomiseen. JavaScript noudattaa suurimmaksi osaksi näitä ohjeita ja siksi sitä voidaan luokitella osaksi ECMAscriptin käyttäjäksi. Usein JavaScriptin määritelmä menee sekaisin ECMAScriptin kanssa, sillä yksi on määritelmä, kun taas toinen seuraa jo olemassa olevia määritelmiä. (Aranda, 2017)

Tässä on luettelo ECMAScript versioista 1–6 julkaisuista

- ECMAScript 1 (1997)
    - Pohjautuu JavaScriptiin ja sen tarkoituksena oli kehittää sitä.
- ECMAScript 2 (1998)
    - Pieniä muutoksia tehtiin.
- ECMAScript 3 (1999)
    - Lisättiin Try-Catch virhe hallintaa, Regular Expression ja String toimintoja kehitettiin.
- ECMAScript 4 (Ei tullut)
    - Tämä vastustettiin, joka johti sen peruuttamiseen. Sen tarkoituksena oli tehdä suuria muutoksia standardiin.
- ECMAScript 5 (2009)
    - Uutena tuli JSON-tuki, ”strict-mode” sekä taulukonkäsittelyyn liittyviä parannuksia.
- ECMAScript 5.1 (2011)
    - Tässä versiossa tehtiin pieniä parannuksia ja virheiden korjaamista.
- ECMAScript 6 (2015)
    - Nykypäiväiset ominaisuudet tulivat pääosin tästä versiosta. Uutena tuli let, const, Class-luokat, template literals ja Arrow-funktio.

(Web Reference, n.d.) Näiden päivitysten jälkeen on tullut säännöllisesti vuosittaisten päivitysten mukana aina jotakin uutta.

3.2 TypeScript

TypeScript on Microsoftin kehittämä lisäosa JavaScriptiin, joka tuo tyyppi määritelmiä koodiin. JavaScriptin alkuperäinen käyttötarkoitus oli tarkoitettu selaimelle ja siksi siinä esiintyy puutteita esimerkiksi palvelinkehityksessä, kun käytetään Node.js kehitysympäristöä. Tyyppien puuttuminen tekee koodin ennakoimisesta haastavan ja se hankaloittaa uusien koodien kirjoittamista sekä yleisesti ylläpitämistä. Se on aikaa säästävää, koska se vähentää esimerkiksi funktioiden palautus tyyppien arvailua ja vähentää dokumentaation katselua. TypeScript koodi tiedosto nimetään yleensä ”.ts”, joka eroaa JavaScriptin tyypillisestä ”.js” tiedostosta. TypeScript koodia pitää ensin muuntaa ennen kuin sitä voidaan käyttää ja tämä tapahtuu TSC eli The TypeScript Compilerilla. Tämä kieli antaa mahdollisuuden myös kirjoittamaan tavallista JavaScriptiä, sillä se ei korvaa kokonaan tuttua perinteistä tapaa, vaikka on suositeltavaa käyttämään sen tuomia lisäominaisuuksia. (Shubel, 2022)

Tässä on osa yleisimmistä TypeScriptin mahdollistamia tyyppi määritelmiä listattuna

- TypeScript mahdollistaa data tyyppien määrittelyä, jossa voidaan määrittää muuttujia esimerkiksi numeroiksi, teksteiksi tai taulukoksi. Alla olevassa koodi esimerkissä nähdään kuinka TypeScript eroaa JavaScriptistä. Funktion logMessage päädyssä nähdään void, joka kertoo sitä, että funktiota kutsuessa se palauttaa tyhjää. Voidaan myös havaita, että funktion parametreillekin pystytään määrittämään tyyppiä. Tyypit any, never ja void ovat aika samankaltaisia, mutta niillä on kuitenkin eroa. Void tarkoittaa sitä, että palautetaan tyhjää ja never taas sitä, että ei ikinä palauteta mitään koskaan. Any palautustyyppinä antaa mahdollisuuden palauttamaan mitä tahansa tietoa funktion. Any muuttujien tapauksissa tarkoitetaan nimensä mukaan, että muuttuja voi olla mikä tahansa. (Soeiro, 2023)

```
//TypeScript
let numerot: number[] = [56, 23, 35, 54]
let kirjanNimi: string = "Oppari"
let luku: any = 54
function sano(viesti: string): void {
    console.log(viesti)
}

function throwError(viesti: string): never {
    throw new Error(viesti)
}

//JavaScript
let numerot = [56, 23, 35, 54]
let kirjanNimi = "Oppari"
let luku = 54
function sano(viesti) {
    console.log(viesti)
}

function throwError(viesti) {
    throw new Error(viesti)
}
```

- Esimerkissä on Arrow-funktion tyypin määrittelyä, jossa ensin muodostetaan tyyppiä ja kerrotaan, että a ja b parametrin sekä palautuksen tyyppi on numero. Lisäksi Interfacen avulla voidaan antaa objekteille tyyppi määritelmän. Interface on siitä hyvä, että sitä voi laajentaa extend-toiminnolla toiseen interfaceen, jolloin se perii toisen interfacen ominaisuuksia eli sen märittämät kentät. (Soeiro, 2023)

```
// function type
```

```
type AddFunction = (a: number, b:number) => number;
const add: AddFunction = (a,b) => a + b;


// type for object
interface Person {
    name: string;
    age: number;
}

const person: Person = { name: "Alice", age: 30 };
```

(Soeiro, 2023)

- Luokan määrittämiseen voidaan lisätä <T> nimen jälkeen, joka kertoo, että kyseessä on geneerinen luokka ja objektia luodessa T tilalle pitää laittaa haluttu tyyppi kuten esimerkissä. Tämä antaa mahdollisuuden tehdä uudelleen käytettäviä luokkia, jotka toimivat monen muuttuja tyyppien kanssa. Nyt kun T on määritettynä niin automaattisesti luokan metodien palautus, ja parametrin arvot ovat annetun tyypin mukaisia. (Soeiro, 2023)

```
class MyArray<T> {
    private elements: T[] = [];

    push(item:T) {
      return this.elements.push(item);
    }

    pop(): T | undefined {
      return this.elements.pop();
    }

    get(index:number): T | undefined {
      return this.elements[index];
    }
}

const numberArray = new MyArray<number>();
numberArray.push(1);

const stringArray = new MyArray<string>();
stringArray.push('Hello');
stringArray.push('World');
```

(Soeiro, 2023)

4 React ja React Native

4.1 React

React on yksinkertaisesti JavaScript kirjasto, jonka tarkoituksena on avustaa käyttöliittymien tekemisessä. Facebook nykyisin Meta -yhtiö kehitti tämän työkalun (Meta, 2021). Tämä kirjasto nopeuttaa kehitystä ja se on rakenteeltaan joustava, koska siinä pystyy hyödyntämään komponentteja, joiden tarkoituksena on tehdä koodeista uudelleen käytettäviä. (Shah, 2024) Vielä tämän mukana tulee monia React hook, kuten usein käyttöön otettu useState, jota käytetään funktiokomponenteissa hallitsemaan tietojen sijoittamista käyttöliittymään (Esa, 2023). Tämä Hook helpottaa tietojen päivittämisessä ja vastaanottamisessa huomattavasti, kun verrataan esimerkiksi puhtaaseen HTML:n ja JavaScript koodiin, jossa pitää hallita tiedon kulkua manuaalisesti.

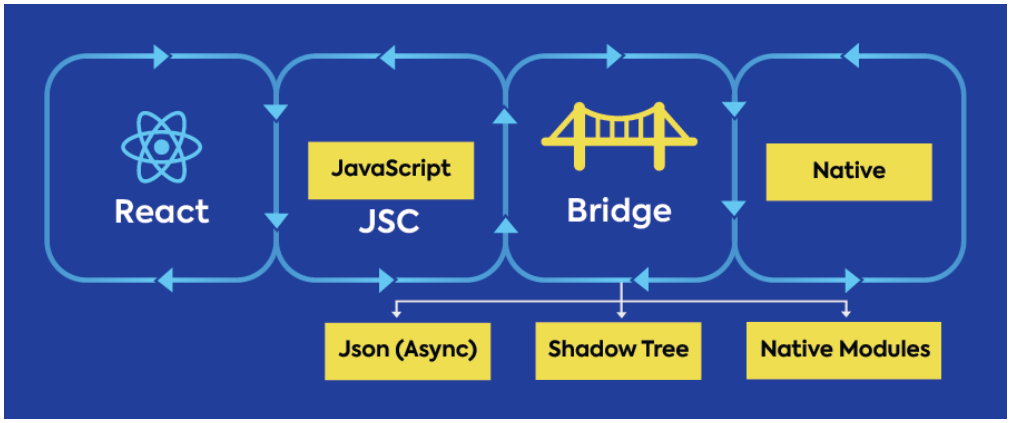
Yleisin käyttö Reactille on verkkosivujen kehittäminen, vaikka sitä voidaan käyttää moniin muihin JavaScriptin ympäristöihin. Monet suuret yritykset, kuten Netflix, Codeacademy ja Facebook käyttävät tätä kirjastoa omissa sovelluksissaan. Suosittu Android sovellus WhatsApp käyttää molemmissa verkkosivuilla sekä Android ja IOS sovelluksessaan Reactia. Suurimmat hyödyt tästä kirjastosta on ollut sovelluksen toiminnan nopeus ja sen mukana tuleva modulaarisuus. (Shah, 2024)

4.2 React Nativen tarkoitus ja hyöty

React Native on JavaScriptiin pohjautuva mobiilisovellus kehitysalusta, joka antaa käyttäjän luoda yhdellä projektilla molemmille Android ja IOS käyttöjärjestelmälle natiivisovelluksen. Sitä luotiin 2015 avoimena lähdekoodina Githubiin, koska aluksi yritettiin kirjoittaa selain tekniikalla HTML5:n kanssa sovellusta, joka lopulta kuitenkaan ei toiminut hyvin. Nimestä voidaan myös päätellä, että se hyödyntää React-kirjastoa JavaScriptin kääntäjän avulla. Natiivisovelluksella tarkoitetaan sellaista sovellusta, jota voidaan asentaa käyttöjärjestelmään hyödyntäen kohdealustan valmiina olevia ohjelmointirajapintoja. Siksi natiivisovellukset ovat aina paljon optimoidumpia kuin muut ratkaisut. React Nativea kutsutaan natiiviksi, koska se hyödyntää JavaScriptin kautta suoraan kohdealustojen rajapintoja. Osa paketeista hyödyntää JavaScriptin ja Nativen välillä siltaa, joka yhdistää niitä kommunikointia varten (kuva 7). Silta ei ole kuitenkaan kokonaan täydellinen, vaan se lisää ylimääräisen vaiheen, joka tekee sovelluksesta hitaamman kuin tavallinen natiivisovellus. (Budziński, 2023) Silta pääosin lähettää JSON-muotoista dataa ja muuntaa sitä objekti muotoon edestakaisin ja tämä syö suoritin resursseja. Nykyään kuitenkin on luotu JSI eli JavaScript Interface, jolla

voidaan suoraan kommunikoida C++ ja JavaScriptin kääntäjän välillä ilman JSONin käyttö. JSI on uusi arkkitehtuuri, jonka tarkoitus on korvata kokonaan sillan käyttöä. JavaScript kääntäjät kuten Hermes ja JSC käytetään yleensä React Nativen kanssa. Niillä on pienet erot toisiinsa verrattuna suorituksen kannalta, mutta pääsijaisesti JSI eli JavaScript Interface tuo niihin Fabric ja Turbo moduulin. Nämä ovat tärkeitä, sillä ne tuovat uusia ominaisuuksia ja parantavat huomattavasti suorituskykyä. (Abdallah, 2023)

Kuva 7 React ja natiivin välillä olevan sillan paikka. (Saini, 2022-b)



React Nativen suurin tarjoama mahdollisuus on ohjelmointi taitojen uudelleenkäyttöä. Toisten ohjelmointi kielien kuten Objective-C ja Javan opiskelu ovat aikaa vieviä monimutkaisia kieliä. Tästä hyötyy eniten käyttöliittymän kehittäjät, koska he voivat myös hyödyntää ennestään tuttuja NPM eli Node Package Managerin paketteja, joita yleisesti käytetään web kehityksessä. React Native ei ole nopein vaihtoehto, mutta siitä huolimatta sen mukana tulee myös uudelleen käytettävät koodit, natiivi käyttöliittymäkomponentit, Live Reload ja suuri yhteisö. Se vähentää huomattavasti tarvittavien kehittäjien määrää ja pienellä ryhmällä voidaan saavuttaa enemmän molempiin alustoihin. Suosituimmat sovellukset kuten Facebook, Discord, Instagram, Pinterest, Ubert eats ja Skype hyödynsivät tätä tekniikkaa. (Saini, 2022-a)

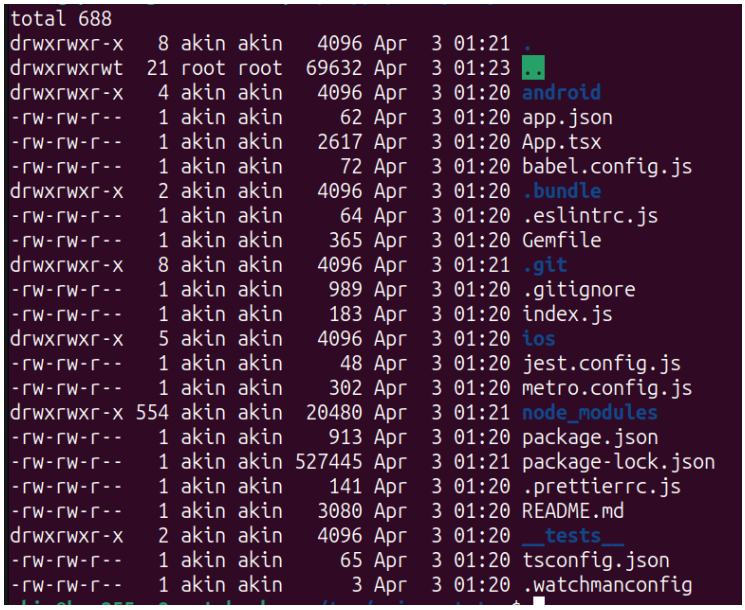
4.3 Reactin ja React Nativen ero

Nyt tiedetään Reactista ja React Nativesta, mutta lyhyesti sanottuna yksi niistä on pelkkä kirjasto ja toinen on kirjastoon perustuva kehitysalusta. Niitä käytetään aika lailla samalla tavalla, mutta seuraavina käydään niiden asennusta, projektin muotoa ja koodissa esiintyviä eroja.

Linux Pohjasilla käyttöjärjestelmillä React Native on riippuvainen näihin seuraaviin Node, React Native Command Line Interface, Watchman, JDK eli Java Development Kit ja Android

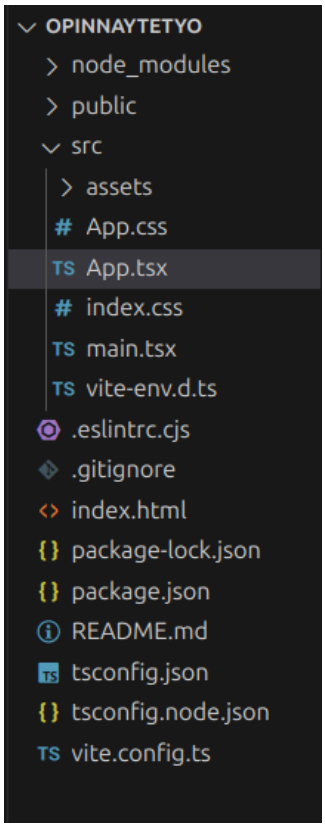
Studio. Android studio on koodieditori, jonka mukana tulee paljon muuta ominaisuuksia kuten Android kehitykseen tarkoitettujen työkalujen päivitysikkunat ja itse Android emulaattori. Näiden jälkeen voidaan asentaa oman maun mukaan mikä tahansa koodieditoria. Seuraavaksi voi edetä riippuvuuksien asentamisen jälkeen asentamaan ”npx react-native@latest init opinnaytetyo ” komentoa käyttäen NPX eli Node Package Execute. Vaihtoehtoisesti voidaan myös asentaa paikallisesti react-native-cli, joka tekee täysin saman kuin NPX, mutta sitä ei suositella, koska se ei päivitä automaattisesti vanhaa react-nativen projektin luojaa. Uuden projektin pohjassa on aika monta tiedostoa (Kuva 8), mutta ”App.tsx” on lähtökohta, josta voidaan saada projektin alkuun. (React Native, n.d.-b)

Kuva 8 Puhtaan React Nativen luoma projektikansio.



Puhdas React projekti tehdään muilla työkaluilla kuten Vite. Perinteinen tapa Create React App vanheni vuonna 2023 ja sitä ei enää ylläpidetä. Vite ei käytä Webpack ja Babelia käännökseen, joten niiden osalta ei tarvitse tehdä enää konfiguraatioita. Se on monta kertaan nopeampi ja tiiviimpi kuin perinteinen vanhentunut tapa. Tämän mahdollisti Webpack kääntäjän korvaamista ESBuildilla. ESBuild on kirjoitettu käyttäen GO-kieltä, joka hyödyntää moniytimistä prosessoria paremmin kuin alkuperäinen JavaScriptillä rakennettu kääntäjä. Reaaliaikaisesti ja rakentaessa saadaan projektin muutokset näkymään näytössä paljon nopeammin tällä uudella tekniikalla. Vite:llä muutokset tapahtuvat lähes odottamatta, kun taas alkuperäisellä React komennolla kestää ainakin viisi sekuntia latautumiseen riippuen koneen nopeudesta. (Madushan, 2024) Vite toimii monille alustoille, mutta Reactin kannalta projektia luodaan käyttämällä komentoa ”npm create vite@latest my-react-app opinnaytetyo --template react-ts” ja tähän tarvitaan Node.js asennettuna koneeseen (Vite, n.d.). Tämän jälkeen voidaan aloittaa lähtö tiedostosta ”src/App.tsx” ja projektin juuri näkyy kuvassa 9.

Kuva 9 Viten luoma projektikansio.



Syntaksista voidaan havaita, että React Nativessa käytetään lisänä ”react-native” niminen import kirjasto riviltä 2 (Esimerkki 1). Tämä tuo natiivi komponentit Text ja View, joita käytetään HTML5 tavoin JSX syntaksissa. Molemmat näistä esimerkki koodeista näyttää näytölle ”Moi opinnäytetyö!”. Esimerkissä 2 voidaan, vaan suoraan käyttää HTML-elementti h1, koska selain ymmärtää suoraan tätä syntaksia. Lisäksi yleisesti navigointi järjestelmässä on eroa näiden välillä, koska pelkässä Reactissa käytetään react-router ja React Nativessa on paljon muita vastaavia, jotka tekevät samanlaista työtä.

Esimerkki 1. React Nativessa tehty yksinkertainen koodi.

```
import React from 'React';
import {Text, View} from 'react-native'

const App = () => {
    return (
        <View>
            <Text>Moi opinnäytetyö!</Text>
        </View>
    );
};
export default App;
```

Esimerkki 2. Reactissa tehty yksinkertainen koodi.

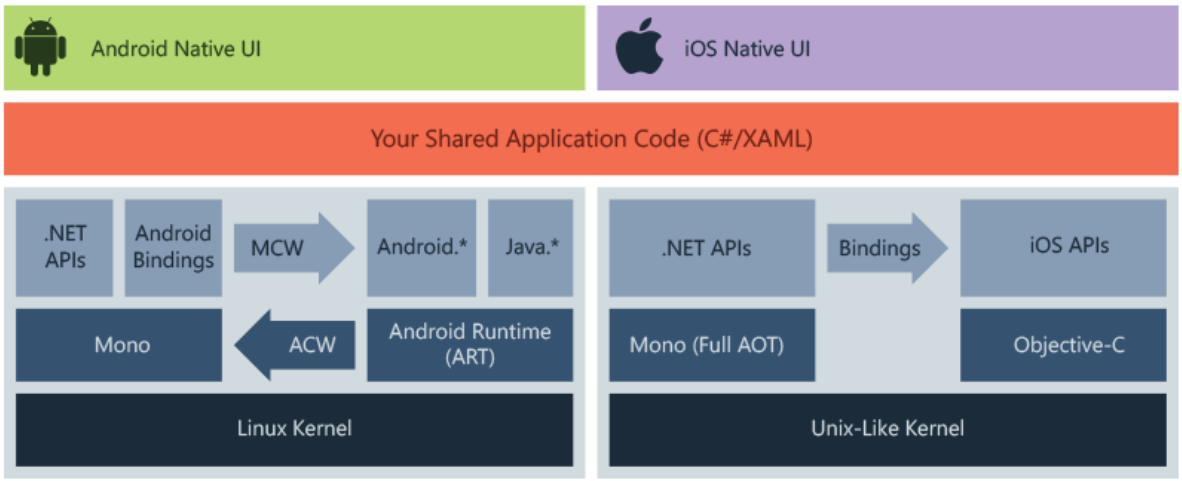
```
import React from 'React';

const App = () => {
    return (
        <h1>Moi opinnäytetyö!</h1>
    );
};
export default App;
```

4.4 React Native verrattuna muut vastaavat vaihtoehdot natiivin lisäksi

React Nativen ja natiivi ratkaisujen lisäksi on muita suosittuja työkaluja kuten Flutter, Xamarin/MAUI ja NativeScript. Flutter on React Nativen pääkilpailija, sillä sitä on kehittänyt suuri yritys Google. Flutter pohjautuu Dart kieleen ja se on vakaa sekä se hyödyntää tehokkaan renderöinti moottorin Impellerin (Flutter, n.d.). Flutter ei käytä lainkaan siltaa kommunikointiin kuten jotkut vanhat React Nativessa tehdyt paketit ja siksi se on prosessorille resurssi ystävällisempi ja nopeampi joissain tapauksissa. Lisäksi se on saanut suosionsa, koska siitä löytyy valmiiksi suuri valikoima natiivi käyttöliittymäkomponentteja (Walburg, 2022). Tällä tavoin ohjelmistokehittäjät voivat välttää pelkän React Nativen tavoin erillisten pakettien asentamista ja saada enemmän vakautta koodien ylläpitämiseen. Joka tapauksessa nykyään React Nativen mukana käytetään Expoa, joka antaa myös valmiiksi paljon komponentteja, kuten Flutterissa on jo oletuksena.

Xamarin on avoin lähdekoodi, jonka avulla voidaan kehittää sovelluksia Android, IOS, Linux sekä Windows käyttöjärjestelmille käyttäen .NET C#. Yhdeksänkymmentä prosenttia koodeista voidaan jakaa ympäri alustoja ja kehittäjille tämä on hyvä. (Microsoft, 2020-a) C# on aika suosittu ja sitäkin käytetään muun muassa palvelin kehitykseen, peliohjelmointiin ja tietokone ohjelmien luomiseen (Gupta, 2024). Kuvassa 10 näkyy sen toiminnan malli ja mitä se hyödyntää eri käyttöjärjestelmissä. Nykyisin kuitenkin .NET MAUI korvaa kokonaan Xamarinia, sillä se on vanhentunut ja sitä ei enää ylläpidetä. MAUI on yksinkertaisesti parannettu ja jatko versio Xamarinista. Tämä uusi versio helpottaa sitä, että voidaan käyttää yhtä projekti pohjaa monen käyttöjärjestelmän alustojen ohjelmien luontiin, kun taas vanhassa versiossa piti tehdä erikseen eri projekti kansioita. Lisäksi sen ympäristö on aika tuttua ja helposti ymmärrettävää niille, jotka ovat tulleet Xamarinista. (Thirunaukkarasu, 2023)

Kuva 10 Xamarinin toiminta systeemi. (Microsoft, 2020-b)



NativeScript tukee erilaisia ympäristöjä kuten Angular, Vue.js ja myös React. NativeScript tekee samaa kuin React Native, mutta siinä on pienempi yhteisö ja sen tuki ei ole yhtä hyvä. Oppimisessa menee huomattavasti enemmän aikaa ja se on haasteellista uusille JavaScript kehittäjille. Hyödyt tästä työkalusta ovat nopea käynnistyminen, suora Native API yhteys ja sen joustavuus. Yksinkertaisesti on kannattavampaa käyttää muita vaihtoehtoja, sillä projektin lopullinen koko on suuri, jos käyttää tätä työkalua. (Wadhwani & Verma, 2024)

4.5 Expo React-Nativen kanssa

Expo on React Nativelle suunnattu kehitysalusta, jolla voidaan luoda Android, IOS ja Web sovelluksia. Vaatimuksena on XCode yleensä, kun kehittäjä haluaa kehittää sovelluksen IOS käyttöjärjestelmälle. Haasteena on, että XCode vaatii Applen tietokonetta ja se ei toimi muissa käyttöjärjestelmissä. Expo ratkaisee tämän ongelman täydellisesti, sillä se mahdollistaa reaaliaikaista kehitystä missä vaan. Tämä toimii Expon ympäristöön kuuluvien työkalujen avulla kuten Expo Go ja Expo SDK ansiosta, joka standardisoi natiivi tarpeet ja tekee niistä helposti käytettävän. Puhtaassa React Nativessa taas pitää asentaa erillisesti näitä NPM natiivi paketteja, jotka tekevät ylläpitämisestä vaikean, koska ne vanhenevat nopeasti ja eri henkilöt ylläpitävät niitä. Expo Go niminen kehityssovellus löytyy suoraan Play kaupalta tai App Storesta, josta sitä voidaan ladata puhelimelle. Tämä sovellus toimii siten, että QR-koodia tietokoneelta skannataan puhelimella ja automaattisesti tietokoneella tehdyt muutokset tulevat näkyviin puhelimen näytölle. Expo SDKn kirjastot ovat pakattu valmiiksi siihen valmiiseen kehitys sovellukseen, jotta kehityksen aikana kehittäjä voisi hyödyntää suoraan näitä natiivi komponentteja ilman odotusaikaa. Kaikki ei ole kumminkaan niin mahtavaa, vaan Expo SDK:lta puuttuu osa natiivi komponenteista, joita ohjelmoija saattaisi tarvita. Pitkään tämä on ollut ongelmana, joten Expo on ratkaissut tätä ”expo eject”

komennolla, joka muuttaa projektin takaisin perinteiseen React Native muotoon, jossa nyt Expon moduuleita sekä omia natiivi komponentteja pystytään käyttämään yhdessä. Tämä oli pitkään ollut ratkaisuna, kunnes Expo julkaisi Expo Dev Clients. Tällä uudella paketilla voidaan suoraan laittaa omia natiivi paketteja Expon pakettien lisäksi ilman, että projektia muunnetaan takaisin React Nativeksi ja menettämättä Expon tuomia ominaisuuksia. Se toimii siten, että se laajentaa avoimen lähdekoodin Expo GO sovellusta ja lisää nämä uudet natiivi paketit siihen mukaan, jotta ne olisivat käytettävissä. Expo ei ole aina ollut näin hyvä ja siksi siihen on jäänyt sellaista mainetta, että se on puutteellinen ja rajoittava sekä tuotantoon sopimaton ratkaisu. Lähiaikoina se on kehittynyt ja jopa React Nativen dokumenteissa suositellaan nykyään vahvasti Expon käyttöä. (Pregasen, 2023)

EAS eli Expo Application Service on pilvipalvelu, joka antaa käyttäjälle mahdollisuuden rakentamaan Expo sovellusta Expon julkisella palvelimella tuotanto muotoon käyttäen EAS Build. EAS Submit taas antaa käyttäjän julkaista sitä suoraan Play kauppaan tai App Storeen rakennuksen jälkeen. Mukana tulee myös mahdollisuus julkaista päivityksiä suoraan pilven kautta sovellukseen, jonka avulla pystyy päivittämään joitakin osia kuten kuvat, tyylit ja JavaScriptiin tehtyjä muutoksia suoraan julkaistuun sovellukseen. (Pregasen, 2023)

5 Käyttöliittymän tiedonkulku

5.1 Tietojen tallentaminen

Tietojen tallentamiseen löytyy useampia tapoja, mutta käyttökohteen mukaan kannattaa valita sopivin vaihtoehto. Projektissa käytetään React Nativen asyncStoragea, joka mahdollistaa tallentamista ympärisovellusta. Se ei ole salattua tietoa, mutta se toimii sovelluksessa Async toiminnon avulla, jonka ansiosta se toimii esteettömästi. IOS käyttöjärjestelmässä tämä kirjasto tallentaa tietoa natiivilla ominaisuudella, kun taas Androidissa hyödynnetään SQLite tai RockDB tietokantaa riippuen saatavuudesta. (React Native, n.d-a) Tällä kirjastolla kuitenkin on omat heikkoudet, koska tällä kirjastolla ei voida pitää kirjautumistietoja salattuna, joita käytetään uudelleen kirjautumisissa. Kuitenkin ratkaisu tälle on react-native-keychain, jolla voidaan salata henkilökohtaista tietoa hyödyntämällä IOS ja Androidin natiivi menetelmiä arkaluonteisten tietojen säilyttämiseen. Vanhoissa Android laitteissa tämä ei ole täysin turvallinen, koska dekryptointi avaimet säilytettyyn dataan löytyvät tiedostona, johon pääsee käsiksi Root vapautetulla laitteella. Uudemmilla laitteilla taas avaimet ovat suoraan säilytetty suoraan laitteistoon, joka tekee siitä turvallisemman. (Khan, 2021)

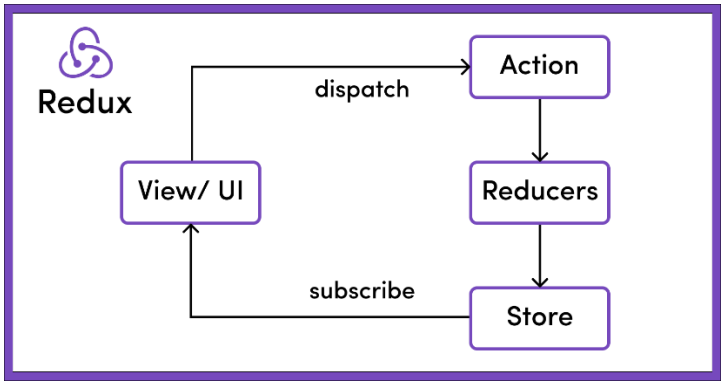
Redux Persist on suunnattu Reduxin tallentamista varten ja se hyödyntää React Nativessa AsyncStorage, johon se tallentaa sen Staten eli tilan. Se on laajennusosa, jota voidaan suoraan asentaa käyttäen NPM eli Node Package Managerin. Se tekee Staten tallentamisesta helpon ilman, että montaa riviä koodia tarvitaan. Se tarjoaa jonkinlaisen varmuuden ohjelman kaatumia, verkkoyhteysongelmia sekä tiedon menetystä vastaan. Varsinkin silloin kun sovellus on Offline-tilassa niin Redux Staten tietoa, voidaan silti pitää esillä, kunnes uutta tietoa tulee paikalle. (Briggs, 2023) Vaihtoehtona on myös käyttää SQLite suoraan, koska sillä voi tehdä manuaalisempia toimintoja. Hyötynä tästä on silloin, kun halutaan säilyttää suuri määrä tietoa tietyssä järjestyksessä, jolloin tiedon etsintä helpottuu. Tämän käyttöön vaaditaan SQL-tietokanta osaamista.

5.2 Sovelluksen tilanhallinta

5.2.1 Redux

Redux on JavaScript-kirjasto, joka on tarkoitettu suuren tai monimutkaisempien tietojen hallitsemiseen. Tämä kirjasto helpottaa tietojen välittämistä ympäri sovellusta ja lisäksi se tekee koodista helppo lukuisemman ja ennustettavan varsinkin, kun projektissa tekijöitä on paljon. Redux perustuu tapahtumien lähettämiseen, joka tapahtuu Action-tapahtumalla ja seuraavaksi Reducer eli funktio, joka tekee varastosta uuden kopion uusien tietojen avulla Redux Storeen. Storeen tulleet muutokset voidaan saada välittömästi mistä tahansa projektin komponenteista (Kuva 11). Reduxin päävarasto on tyyppiä Immutable eli sitä ei voida ikinä muuttaa suoraan ja vain lukuoikeus löytyy suoraan käsiteltäessä. (De Roy, 2022-a)

Kuva 11 Reduxin elämänkaari. (De Roy, 2022-b)



Alhaalla olevassa koodissa määritellään alkutilanne Redux varastolle. cartReducer funktiossa luodaan omia Actionia, ja samalla niille annetaan oma nimi ja toiminto Switchin sisällä. Switch palauttaa arvon kohdassa case ”action.type” sisällön mukaan. CarReducerin

funktion parametrista huomaa kohdassa “state = initialCartState”, että tämä asettaa Reducerille oletustilan, sillä kun koodia kutsutaan ensimmäistä kertaan, niin se on tyhjä. Switch määrittää myös Default, joka palauttaa oletuksena nykyisen tilan, jos actionia ei ole määritelty kutsuessa.

```
const initialCartState = {
    noOfItemInCart: 0,
    cart: []
}

const cartReducer = (state = initialCartState, action) => {
    switch (action.type) {
        case "ADD_ITEM_TO_CART":
            return {
                ...state,
                noOfItemInCart: state.noOfItemInCart + 1,
                cart : [
                    ...state.cart,
                    action.payload
                ]
            }
        case "DELETE_ITEM_FROM_CART":
            return {
                // Remaining logic
            }
        default:
            return state
    }
}
```

(De Roy, 2022-a)

Koodi esimerkissä otetaan käyttöön useDispatch() hook, joka tulee suoraan react-redux kirjastosta. Seuraavaksi määritellään addItemToCart funktio, joka palauttaa Actionin objekti muodossa, jotta Redux tietää oikean funktion, johon se laittaa nämä ”payload” tiedot. Payload-tietoihin kuuluu kirjan nimi ja lukumäärä. Button on käyttöliittymä komponentti, joka käynnistää koko prosessin kuin näppäintä painetaan.

```
// Rest of the code

const dispatch = useDispatch()

const addItemToCart = () => {
return {
    type: "ADD_ITEM_TO_CART"
    payload: {
```

```
        bookName: "Harry Potter and the Goblet of Fire",
        noOfItem: 1,
        }
    }
}
<button onClick = {() => dispatch(addItemToCart())}>Add to cart</button>
```

(De Roy, 2022-a)

5.2.2 Redux Thunk ja Saga

Redux thunk on yksi suosituimmista lisäosa kirjastoista, joita käytetään Reduxin kanssa. Sen käyttötarkoitus on tehdä asioita viiveellä sekä erotella logiikat siistiin järjestykseen ja sallia Async toiminnon käyttämistä. Async tulee sanasta Asyncronous eli yhtä aikaan tapahtuva. Esimerkkinä tähän olisi, vaikka HTTP eli Hypertext Transport Protocolin -pyynnöt, joissa odotetaan vastausta palvelimesta, sillä Reduceriin ei saa laittaa näitä side -efekteja eli tällaisia toimintoja. Redux kirjastosta tuleva ”store.dispacth” antaa Thunk funktiolle dispatch ja getState funktiot parametreina, kun sitä syötetään Callback -funktiona. Näillä voidaan kutsua muita Actioneita Thunkin sisällä, joilla pystytään tekemään muokkauksia sekä lukea nykyistä Reduxin Storen tilaa. Tästä löytyy alhaalta koodi esimerkki, joka kuvaa tätä prosessia. (Redux, n.d.-b)

```
const thunkFunction = (dispatch, getState) => {
   // logic here that can dispatch actions or read state
 }

 store.dispatch(thunkFunction)
```

(Redux, n.d.-b)

Redux Thunk saadaan asennettua komennolla ”npm install redux-thunk”. Asennuksen jälkeen sen käyttö vaatii esimerkki koodista alhaalla seuraavia määrityksiä, jotta ylhäällä oleva koodi toimisi. Tarkennettuna alhaalla olevassa koodissa luodaan Redux varastoa eli Store ja sen tekemiseen käytetään rootReducer sekä thunk middlewarea lisätään, sillä se ei sisälly Reduxiin oletuksena. Tämä rootReducer on sitä varten, että kun Reducereita on enemmän kuin yksi ja halutaan käyttää monta samaan aikaan. Storeen voidaan laittaa vain yksi Reducer ja siksi niitä pitää yhdistää tällä tavoin, koska parametrina pystyy syöttämään vain ainoastaan yhden pää Reducerin. (reduxjs/redux-thunk, 2024)

```
import { createStore, applyMiddleware } from 'redux'
import { thunk } from 'redux-thunk'
import rootReducer from './reducers/index'

const rootReducer = combineReducers({
    example: exampleReducer,
    example2: exampleReducer2,
});

const store = createStore(rootReducer, applyMiddleware(thunk))
```

Redux Saga tekee samaa kuin Redux Thunk, mutta se hyödyntää ES6 generaattori ominaisuutta Promisen funktioiden sijaan. Generaattori funktio pystytään keskeyttämään ja jatkamaan milloin tahansa kuten koodi alla olevassa koodi esimerkissä. Yield eli odotus toiminnolla pystytään keskeyttämään sen, jonka jälkeen voidaan kutsua generaattorin next() funktio, joka kertoo, että koodia saa jatkaa (Mehul, 2023). Se tarjoaa aika samanlaisia kuin ominaisuuksia kuten Redux Thunkin tavan erotella logiikkaa käyttöliittymä koodista, mutta sen mukana tulee lisäksi enemmän toimintoja. Nämä toiminnot antavat mahdollisuuden käyttää näitä seuraavia efekti funktioita kuten put, call, takeLatest ja takeEvery, Tämä kirjasto on hyvä, kun halutaan tehdä monimutkaisempia yhtä aikaan tapahtuvia toimintoja. Redux Saga perustuu siihen, että ensin tehdään generaattori funktio ja sen sisällä voidaan lisätä kaikki aikaa vieviä toimintoja. (Herrera, 2022)

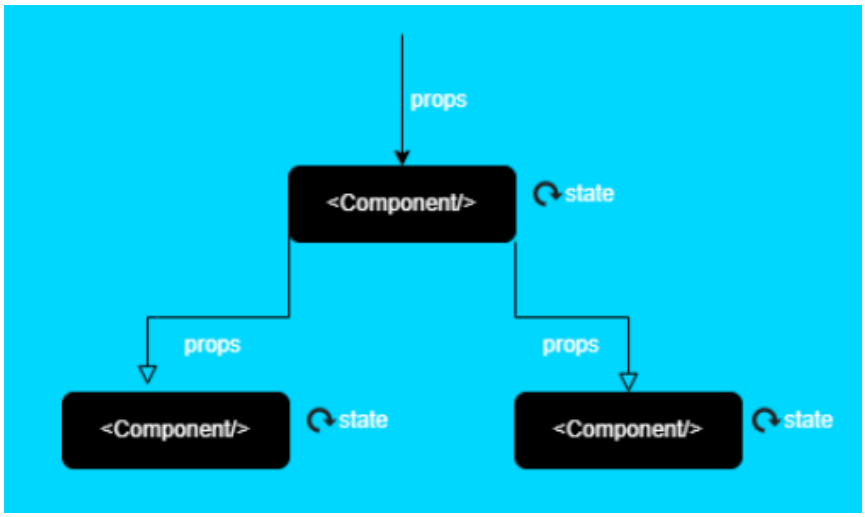
```
function* esimerkkiGenerattori() {
    yield 'Keskeytys 1';
    console.log("Odottaa seuraavaa next()");
    yield 'Keskeytys 2'
    console.log("Odottaa seuraavaa next()");
    return 'Funktion loppu';
}

const generaattori = esimerkkiGenerattori() ;

console.log(generaattori.next());  //tulostaa { value: 'Keskeytys 1', done:
false }

console.log(generaattori.next()); //tulostaa { value: 'Keskeytys 2', done:
false }

console.log(generaattori.next()); //tulostaa { value: 'Funktion loppu', done:
true }
```

5.2.3 Reduxin ja Props-parametrien haasteet

Redux tarjoaa monia hyötyjä kuten yhteinen muistitila, hyvät vianmääritystyökalut ja sitä voidaan skaalata helposti (Rahman, 2023). Kaiken tämän mukana usein tulee paljon ylimääräistä koodia yksinkertaisten asioiden tekemiseen kuten, vaikka esimerkiksi State-varaston muuttujan numeron kasvattaminen yhdellä (Moota, 2024). Pelkästään muuttujan kasvattamiseen pitää määrittää Redux-varastoa, Actionit ja Reducerin, jotta Redux toimintaympäristö lähtisi toimimaan. Ei ole suositeltavaa käyttää sitä pieniin sovelluksiin, joissa ei käytetä paljon yhteisiä ja jatkuvasti päivittyviä tietoja. Lisäksi Reduxin ymmärtämiseen kuluu aikaa, koska siihen kuuluu monenlaisia asioita kuten Middlewaret ja muiden lisäkirjastojen käyttö kuten redux-thunk ja redux-saga. Yleisesti virheitä tapahtuu enemmän, kun koodin määrä kasvaa ja siksi sen kanssa kannattaa olla varovainen. (Rahman, 2023)

Redux julkaistiin 2015 ja sen käyttö oli suosittua React.js kirjaston kanssa, sillä Reactissa usein projektin monimutkaistuessa tapahtuu ongelmia tietojen jakamisen kanssa komponenteissa, joissa tarvitaan yhteisiä muuttuvia tietoja. (Dvalashvili, 2024) Tämä ongelma johtui siitä, että Parent-komponentti voi antaa parametri vain yhteen suuntaan eli vain alaspäin Child-komponenteille. Tästä seuraa ongelmatilanne, kun halutaan päivittää Parent-komponentissa ja muissa komponenteissa olevia tietoja, koska nyt pitää miettiä manuaalisesti tiedon kulkua kuten kuvassa 12. (Atukorala, 2022-a) Ongelman ratkaisun vaihtoehtona on myös Context API, jonka tarkoitus on aika samanlainen kuin Redux, mutta se on paljon yksinkertaisempi. Se ei sisällä kuitenkaan kaikkea ominaisuuksia, jota Redux tarjoaa. Context API sopii esimerkiksi teema tilan jakamiseen ja muihin pieniin määriin tietojen hallitsemiseen.

Kuva 12 Props parametrien kulku komponenteissa. (Atukorala, 2022-b)



5.2.4 Redux Toolkit

Redux Toolkit eli RTK tuo mukanansa useita laajennuksia, joita pelkässä Reduxissa ei ole. Sen tarkoituksena on yksinkertaistaa ja vähentää tarvittavan koodin määrää sekä monimutkaisuutta. Se sisältää avustavia funktioita, jotka auttavat Reduceria luodessa, Storen konfiguroinnissa sekä Immutable eli ei muutettavien State-tietojen hallinnassa. Sitä käyttäessä ei tarvitse asentaa erikseen esimerkiksi Redux Thunk, Immer tai Reselect, koska kaikki nämä yleisemmät laajennukset sisältyvät siihen. (Redux, n.d.-a) Slice on Redux Toolkitin käyttämä ominaisuus, jossa State, Reducer ja Action sisältyvät valmiiksi sen sisälle. Toiminto createSlice mahdollistaa sen, että voidaan laittaa kaikki Redux liittyviä logiikoita yhteen tiedostoon. Alla oleva koodi on esimerkki Slicesta, joka on otettu projektista. Siellä näkyy oletustila eli initialState, jossa on määritelty valmiiksi muuttujille arvoja. Slicelle kuuluu antaa nimen ja määritellä Reducereita sen sisälle. Sen jälkeen, kun konfiguroinnit ovat valmiina, niin pystytään kutsumaan käyttöliittymänkoodin puolelta näitä Reducereita.

```
const initialState: State = {
    location: 'Helsinki',
    gender: Gender.female,
    status: Status.worker
}
```

```
const preferencesSlice = createSlice({
    name:'preferences',
    initialState,
    reducers:{
        setCity: (state, action: PayloadAction<string>) =>
        {
```

```
            state.location = action.payload
        },
        setGender: (state, action: PayloadAction<Gender>) =>
        {
            state.gender = action.payload
        },
        setStatus: (state, action: PayloadAction<Status>) =>
        {
            state.status = action.payload
        }
    }
})
```

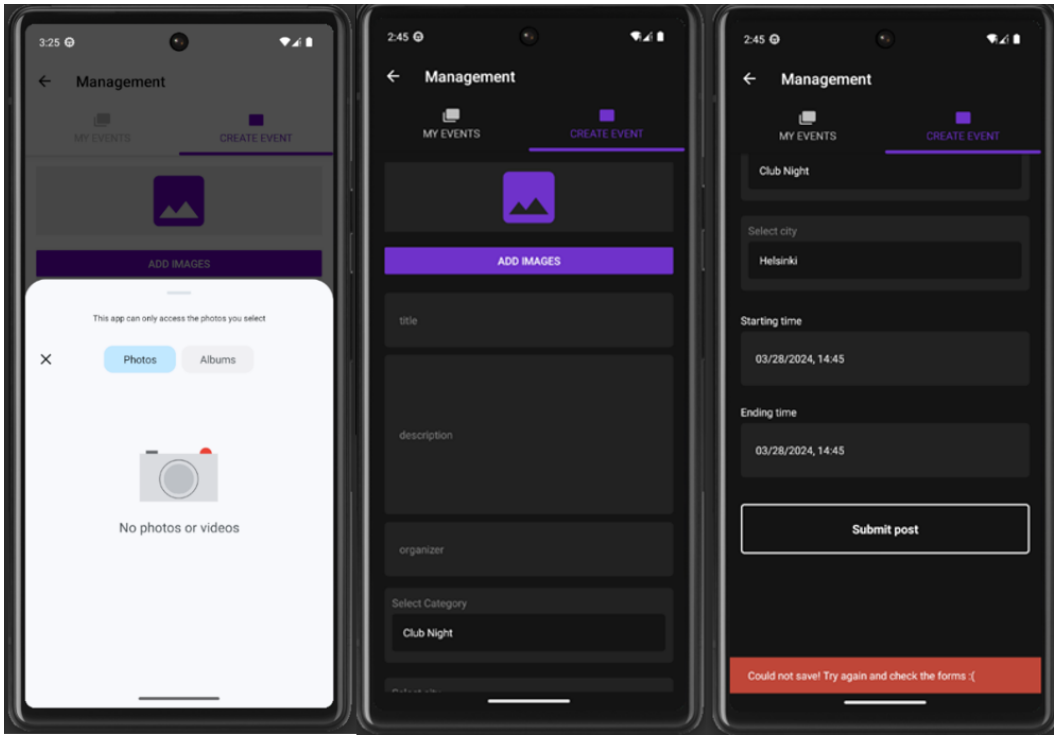
Redux Storen konfigurointi tapahtuu configureStore metodilla, joka yksinkertaistaa Storen luontia ja asettamista. Alla oleva esimerkki koodi on projektista otettu, jossa on määritelty API eli Acces Point Interface Middlewaret sekä muunneltu rootReducer, jossa käsitellään uloskirjautumista. API middlewaret tulevat RTK-querya hyödyntävästä koodin osasta, jonka tarkoituksena on helpottaa pyyntöjen tekemistä hallitsemalla niiden tilaa ilmoittamalla, että onko tieto vastaanotettu tai odotetaanko sitä palvelimelta. Koodi esimerkissä on käytössä persistReducer, joka tallentaa Reducerin tilaa seuraava käyttöä varten, kun sovellusta käynnistetään uudestaan.

```
const rootReducer = (state:any, action:any) => {
  if (action.type === SIGNOUT_REQUEST) {
    AsyncStorage.removeItem('persist:root')
    return reducers(undefined, action);
  }
  return reducers(state, action);
}

const persistedReducer = persistReducer(persistConfig, rootReducer)

export const store = configureStore({
  reducer: persistedReducer,
  middleware: (getDefaultMiddleware) =>
    getDefaultMiddleware({
         …
    })
    .concat([
      authApi.middleware,
      preferencesApi.middleware,
      eventApi.middleware,
      settings: settingsReducer,
      preferences: preferencesReducer,

    ]),
});
```

6 Sovelluksen yksityiskohtien esittely

Kuvassa 13 näkyy käyttöliittymän näkymä, jossa voidaan tehdä julkaisuja kaikille sovelluksen käyttäjille. Tässä näkymässä voidaan lisätä enintään viisi kuvaa ja vaihtoehtoisesti vähemmänkin käy. Vaatimuksena kuvan lisäksi on antaa tapahtumalle nimi, kuvailu, sijainti, alkamis- ja lopetuspäivämäärä. Virheilmoituksia tulee vastaan, kun ei olla täytetty vaadittavat syötteet oikein tai silloin kun ei saada yhteyttä palvelimeen verkkoyhteys syistä. Tallennuksen jälkeen voit aina palata pyyhkäisemällä tai painamalla ylhäällä olevalta ”management” painikkeelta katsomaan omat julkaisut. ”Management” näkymässä voit poistaa ja hallita julkaisuja. Joka tapauksessa onnistuessa saadaan vihreällä viesti, joka ilmoittaa käyttäjälle, että kaikki on kunnossa. ”Submit post” nappia painaessa, lähetetään HTTP pyynnön ja onnistuessa tämä palauttaa koodia 200, joka tarkoittaa sitä, että kaikki on onnistunut oikein. Muun HTTP palautuskoodin mukana tulee virheilmoitus. Animaatioita yleisesti löytyy ympärisovellusta, jotta käyttö kokemuksesta tulisi sulavampaa ja helpottaisi käyttäjän ymmärrystä näkymien vaihtuessa.

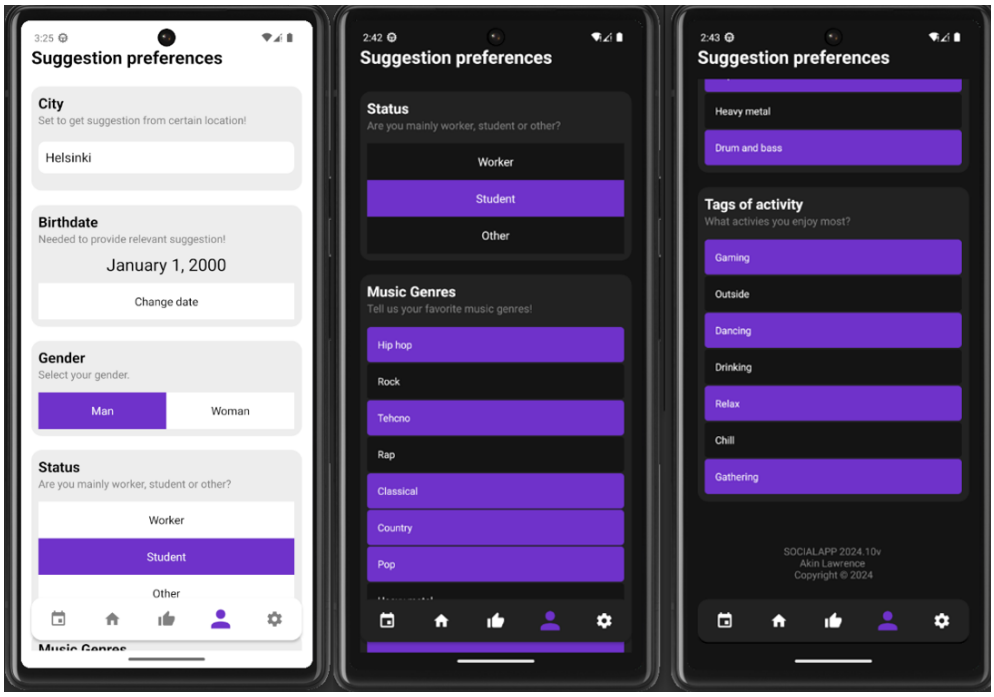
Kuva 13 Julkaisu näkymä.



”Suggestion preferences” näkymässä voidaan tehdä käyttäjäkohtaisia asetuksia. Kuvassa 14 näkyy kaupunki, syntymäpäivä, sukupuoli, nykyinen elämän tilanne, musiikki tyyppi sekä lempi aktiviteetti rajatulta listalta. Asetetun kaupungin mukaan tapahtuma suositukset tulevat. Kaikki tehdyt asetukset tässä näkymässä ei vaadi erillistä tallennusnappien painamista, vaan

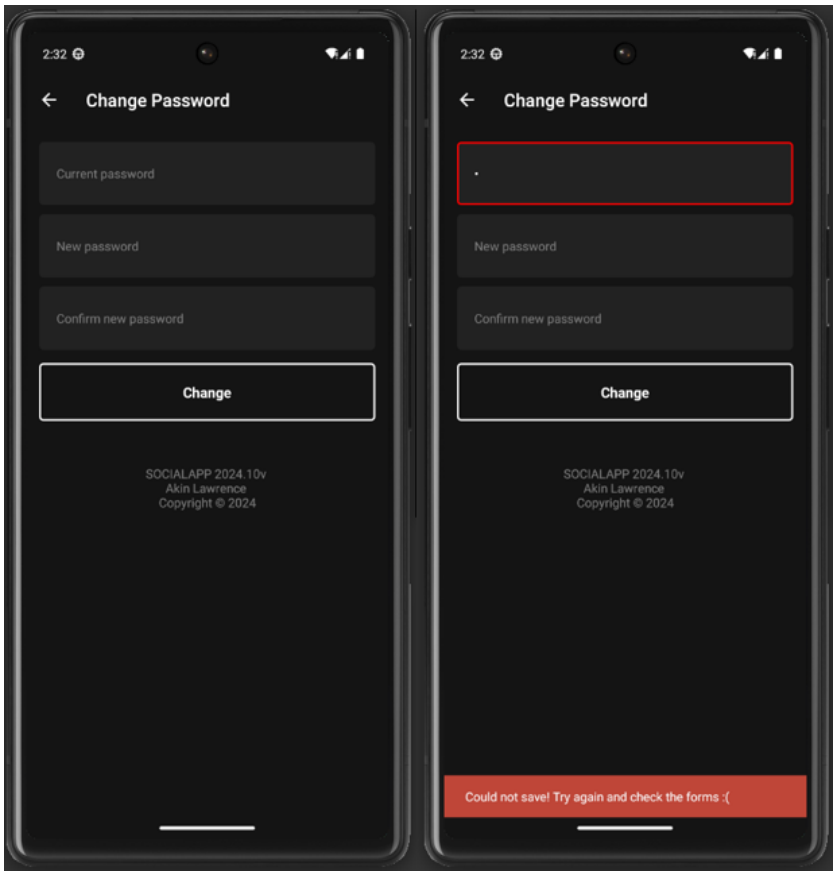
suoraan muutokset menevät palvelimelle ja tallentuvat myös paikallisesti sovellukseen. Sijainti nappia painaessa näkyviin tulee kaupunkeja sisältävä säiliö, joka tulee animaatioiden avulla.

Kuva 14 Ehdotus asetusten yksityiskohdat



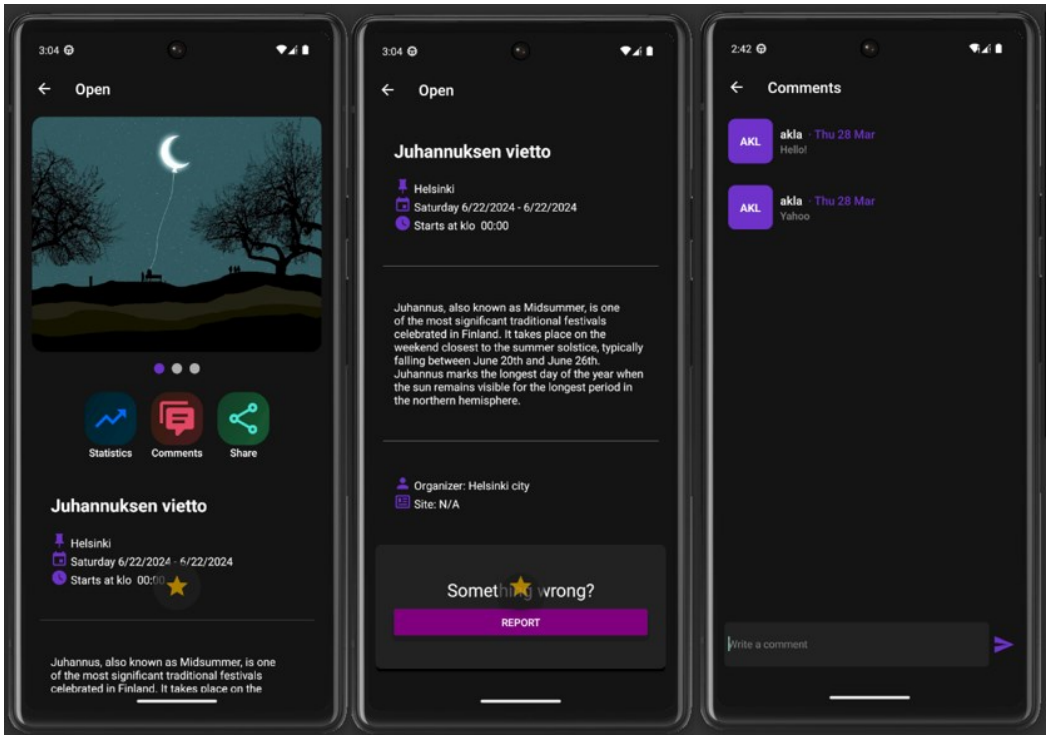
Nämä salasanaan liittyvät asetukset löytyvät ponnahdus ikkunan avulla asetusrattaan kohdalla (Kuva 15). Tämä ikkuna toimii siten. että validoidaan ensin käyttäjän puolen lomakkeet, jolloin virheilmoitukset tulevat näkyviin korostaen virheelliset syötteet punaisella laatikolla. Tämän lisäksi, jos palvelin hylkää pyyntöä silti jostain syystä, niin näytön alhaalta ponnahtaa toinen virheilmoitus, joka kertoo tarkemman ohjeen jatkoa varten. Onnistuessa tulee vihreällä ilmoitus ja ikkuna sulkeutuu automaattisesti sekä palaa takaisin muihin asetuksiin.

Kuva 15 Salasanan vaihto näkymä.



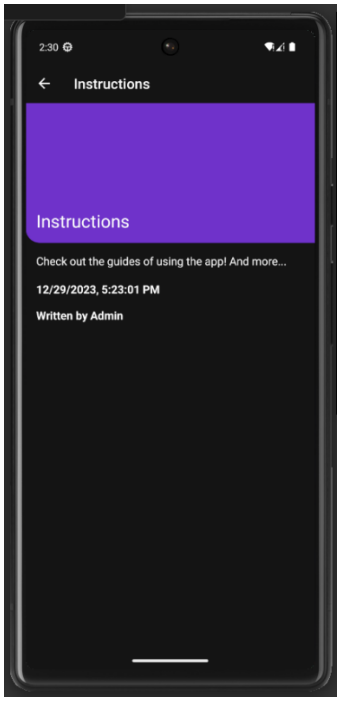
Tapahtuma katselu näkymä on käytössä useammassa paikassa, koska helpompaa on uudelleen käyttää samaa komponenttia (Kuva 16). Kuvaa pystyy vierittämään vasemmalle tai oikealle riippuen kohdasta, jossa ollaan. Kuvan järjestys ilmenee pallon muotoisista komponenteista, jotka ovat peräkkäin järjestyksessä ja aktiivinen kuva on korostettu liilalla. Tämän jälkeen löytyy kommentit, tilastot sekä julkaisun jakonappi. Kommentti ikkunaa avatessa voidaan katsoa palvelimeen tallennettuja kommentteja sekä laittaa omia kommentteja. Tällä hetkellä kommenteista näkyy vain julkaisu päivämäärä ja viesti. Tätä näkymää pystyy myös vierittämään alas, josta löytyy lisätietoa sekä valitus lomake nappi.

Kuva 16 Tapahtuma näkymän yksityiskohdat.



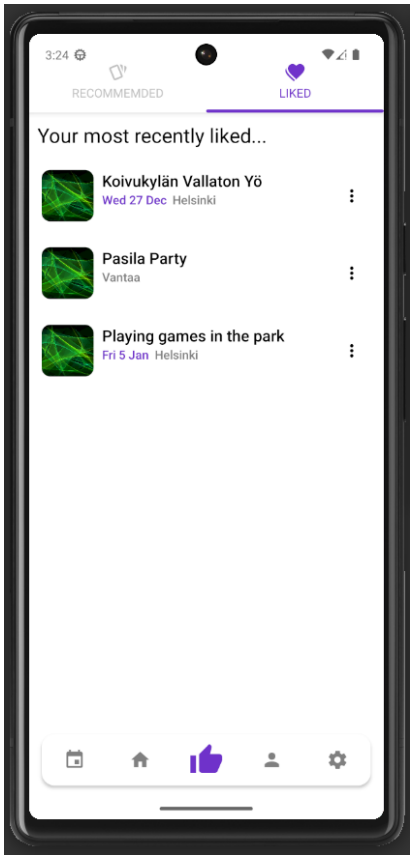
Etusivu uutisista painaessa tulee näkyviin (Kuva 17). Tämä näkymäkin hyödyntää Redux Toolkittiä ja tallennus ominaisuutta, koska on tärkeää aina nähdä jotain tietoa etusivulla, vaikka verkkoa ei löytyisikään. Näitä sisältö haetaan palvelimesta kuten muutkin asiat sovelluksessa.

Kuva 17 Uutiset ponnahdus ikkuna



Käyttäjä pystyy katsomaan omia tykättyjä tapahtumia ja poistaa tykkäyksistä sen pois. Tykkäykset päivittyvät automaattisesti tänne (Kuva 18), jos ne eivät näy, niin vaihtoehtoisesti käyttäjä voi vetää näytön ylhäältä alaspäin. Tällä tavalla voidaan välittömästi päivittää ja saada vastaan uusimpia tykkäyksiä. Nämä tykätyt julkaisut eivät tule enää ehdotuksiin, kun niitä on tykätty, mutta poistaessa niin ne palaavat. Niiden toistumista ehdotuksissa voidaan välttää kokonaan, kun painetaan, ”ei tykkää” painiketta tapahtumaehdotus näkymässä.

Kuva 18 Tykätyt julkaisut näkymä



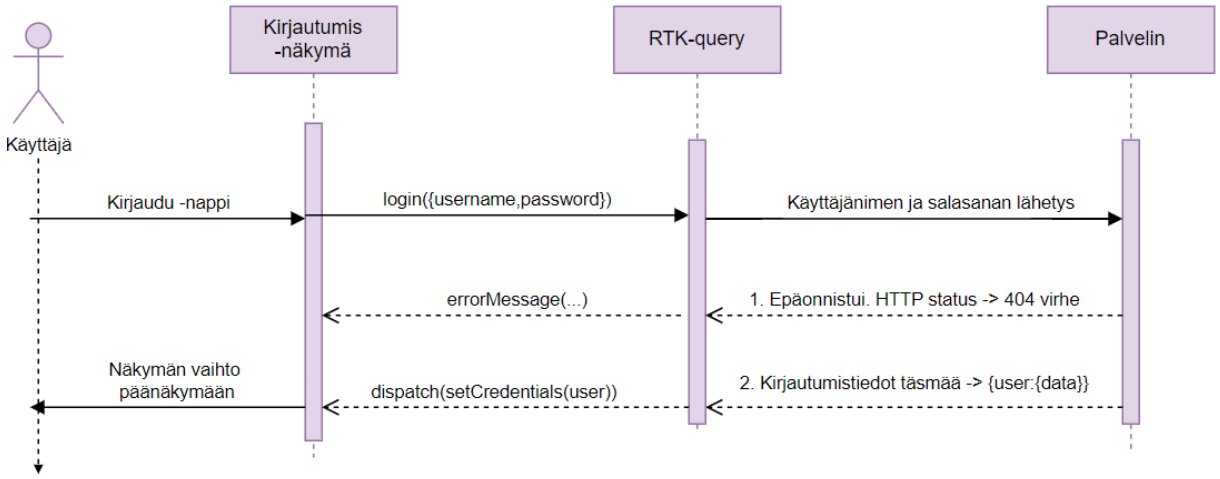
7 Projektin toteutus

7.1 Datan kulku palvelimelle ja käyttö

7.1.1 Kirjautuminen

Kirjautumisprosessi alkaa napin painalluksella, jolloin käyttäjänimi sekä salasana päätyvät lopulta palvelimelle. Välissä kuitenkin pitää ottaa käyttöliittymältä käyttäjän syöttämät kentät, jolloin voidaan antaa niitä eteenpäin käsiteltäväksi RTK-Query:ssa, joka vie nämä tiedot palvelimelle käyttäen JSONia. Palvelin varmistaa vastaanottaessaan tietoa asiakaspuoleisesta sovelluksesta, että salasana ja käyttäjänimi täsmäävät. Pyynnön suorittamisen jälkeen RTK-Query hallitsee sitä, että palautetaanko virhe viestiä käyttäjälle tai vaihdetaanko näkymää kohti koti näkymää. RTK-Query tietää palvelimen käyttäjätiedon hyväksymisen tilannetta HTTP-status koodin perusteella ja lisäksi onnistuneesta pyynnöstä palautuu käyttäjän tiedot sekä Token-avain. Kuvassa 19 on kaaviolla kuvailtu tätä prosessia.

Kuva 19 Sekvenssikaavio kirjautumis näkymän toiminnasta.



Alla on kirjautumista hallitseva funktio, jossa ensin tehdään pyyntö RTK-Queryn avulla ja palautusarvon tullessa asetetaan tiedot käyttäjätieto-Sliceen hyödyntäen Try-Catchin poikkeushallintaa. Tärkein kohta on setCredentials funktio, jossa asetetaan käyttäjää, sillä tämä laukaisee näkymän muutokset pääkomponentissa, jonka vastuuna on tehdä näkymän vaihto kohti koti näkymään.

```
import { useAppDispatch } from "./src/app/hooks"
import { useLoginMutation } from "./src/app/services/authApi"
import { errorMessage } from "./src/constant/messages"
```

```
import { setCredentials } from "./src/features/authSlice"
import { setFirstname, setLastname } from "./src/features/settingsSlice"

…
const [login, { isLoading }] = useLoginMutation()
const dispatch = useAppDispatch()

const login = async ({ username, password }:any) => {
    try {
        const user = await login({ identifier: username, password }).unwrap()
        dispatch(setCredentials(user))
        dispatch(setFirstname(user.user.firstname))
        dispatch(setLastname(user.user.lastname))
    }
    catch (err) {
        errorMessage("Login failed! Check your password or email!")
    }

}
```

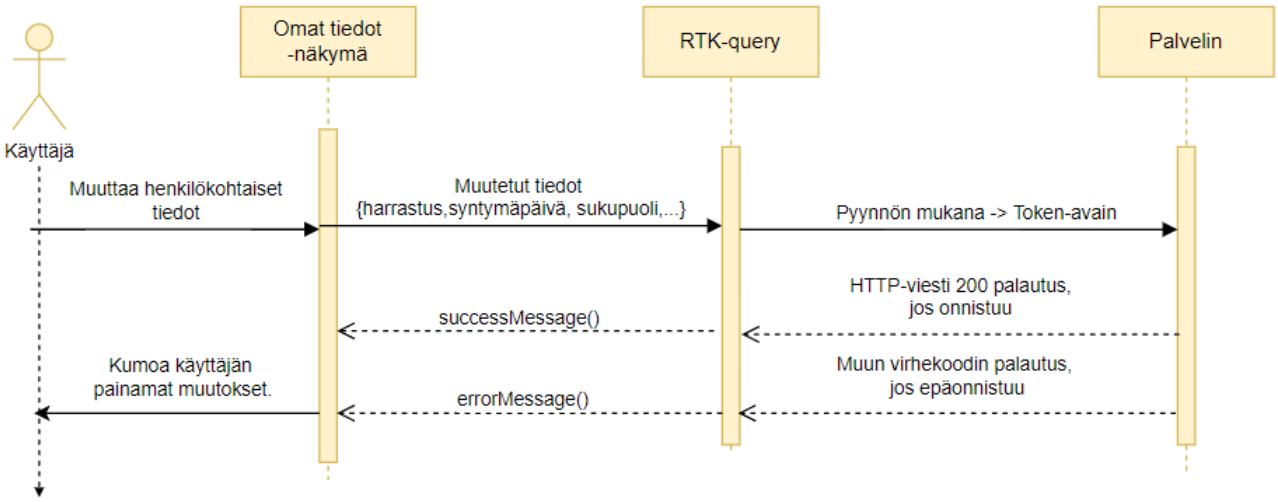
Pääkomponentti App:issa user-muuttuja päivittyy automaattisesti useAppSelectorin avulla. JSX syntaksissa on If-lause, joka näyttää koti näkymää, jos User-muuttuja ei ole tyhjä. Oletuksena user on tyhjä, mutta kirjautumisfunktio asettaa tämän kirjautumispyynnön onnistuessa.

```
function App(): JSX.Element {
  const [theme, setTheme] = useState(false);

  ….
  // Palauttaa data vastaan, jos ollaan kirjautunut
  const user = useAppSelector(selectCurrentUser)
  return (
    <GestureHandlerRootView style={{ flex: 1 }}>
      <ThemeContext.Provider value={theme}>
        {user ?
          (<NavigationContainer>
            <RootStack></RootStack>
          </NavigationContainer>) :
          (
            <LoginScreen />
          )
        }
      </ThemeContext.Provider>
    </GestureHandlerRootView>
  );
}
```

7.1.2 Käyttäjä asetukset

”Preferences”-näkymässä muutokset tehdään suoraan palvelimelle valitsemalla esimerkiksi monivalintalistasta vaihtoehtoa ilman, että pitää painaa erikseen tallennusnappia. Palvelin vastaa takaisin joko virheilmoitusta tai viesti onnistumisesta. Virheilmoitus kumoaa käyttäjän tekemät muutokset sekä asetukset palautuvat entiselle tilaan näkymässä ja taas onnistumisilmoitus tapauksessa ilmoitus tulee käyttäjälle, että kaikki on kunnossa. Kuvassa 20 on kuvailtu tätä samaa toimintaa kaaviolla.

Kuva 20 Sekvenssikaavio "Preferences"-näkymän toiminnasta.



Alla on funktioita, jotka hallitsevat kaupungin muutosta ja syntymäpäivän muuttamista. dateRequest funktiossa muutetaan päivämäärää sopivaan muotoon, jotta palvelin osaa tulkita sitä. Samalla hallitaan näkymän tietoa molemmissa funktioissa setCity ja setOpen, jotka tekevät useStaten avulla tarvittavat päivitykset näkymään.

```
const dateRequest = async (date: Date) => {
   try {
     await updatePreference({ birthdate: dateFormatter(date) }).unwrap()
     await setDate(date)
     setOpen(false)
   } catch (e) {
     setOpen(false)
     errorMessage()
   }

 }

 const cityChangeRequest = async (city: string) => {
   try {
     await updatePreference({ activeCity: city }).unwrap()
     await setCity(city)
```

```
    } catch (e) {
      errorMessage()
    }
  }
```

Alla on harrastuksien ja aktiviteettien valintaan käytetty logiikka, josta on tehty erillinen komponenttia uudelleenkäyttöä varten. Lista palauttaa tasan samaa tietoa mitä näytöllä näkyy valittuna palvelimelle ja sitten seuraavaksi palvelimella tapahtuu käsittely, jossa joko poistetaan valintoja tai lisätään käyttäjän uudet tehdyt asetukset.

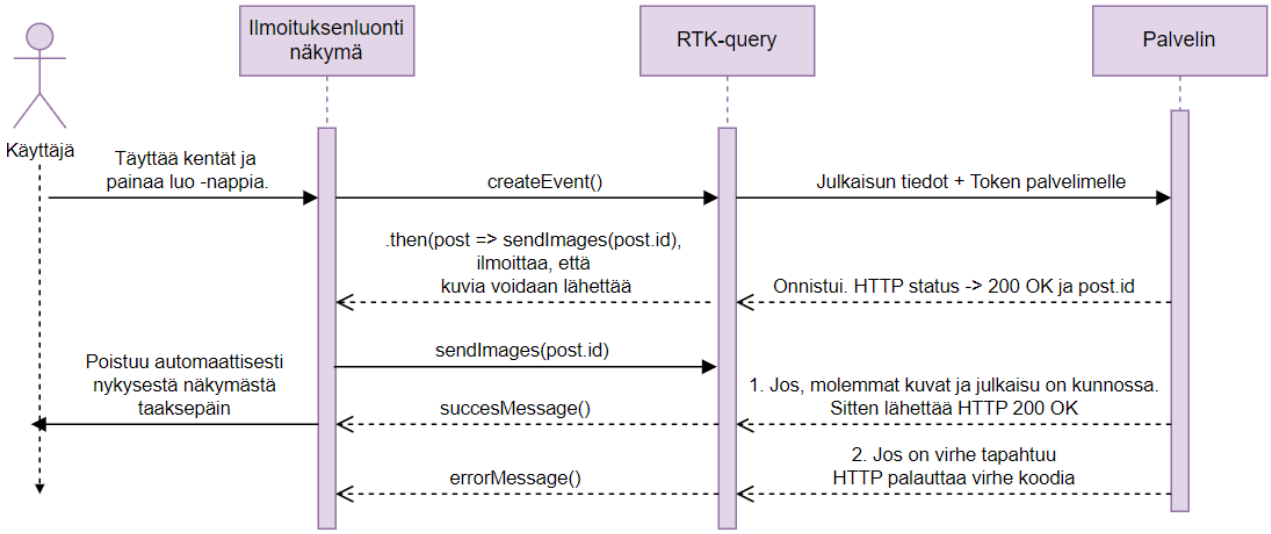
```
const toggleList = React.useCallback(async (id: number) => {
        const found = list.find((element: number) => element == id);
        let bkp: number[] = []
        Object.assign(bkp,list)

        if(found){
          let newList = list.filter((element: number) => element != id)
          setList([...newList])

          const res = await getActive(newList)
           if(!res){
            setList([...bkp])
            error()
          }

        }else{
          let newList = list;
          newList.push(id)
          setList([...newList])

          const res = await getActive(newList)
          if(!res){
           setList([...bkp])
           error()
         }
        }
 }, [list]);
```

7.1.3 Ilmoituksen luonti

Ilmoituksen luontiin kuuluu julkaisun nimi, kuvailu, järjestäjä, kategoria, sijainti, alkamis- ja loppumisaika sekä mahdollisia kuvia. Käyttäjän tehtävä on täyttää nämä kentät ja painaa lopuksi tallenna. Tallennusnapin painamisen jälkeen createEvent() funktio käynnistyy ja pyynnön mukana tulee kaikki täydennetyt tiedot sekä Token-avain, joka päätyy lopuksi palvelimeen. Palvelin vastaa takaisin, että on onnistunut ja antaa ID-tunnuksen julkaisusta,

jota käytetään uudestaan kuvien lataamiselle palvelimelle. Molempien onnistuessa niin näytetään asiakaspuolen koodissa onnistumisviesti ja palataan taaksepäin näkymässä. Kuvassa 21 on kuvailtu tätä samaa toimintaa kaaviolla.

Kuva 21 Sekvenssikaavio ilmoituksenluonti näkymän toiminnasta.



Koodi alla on käyttöliittymän hyödyntämä.

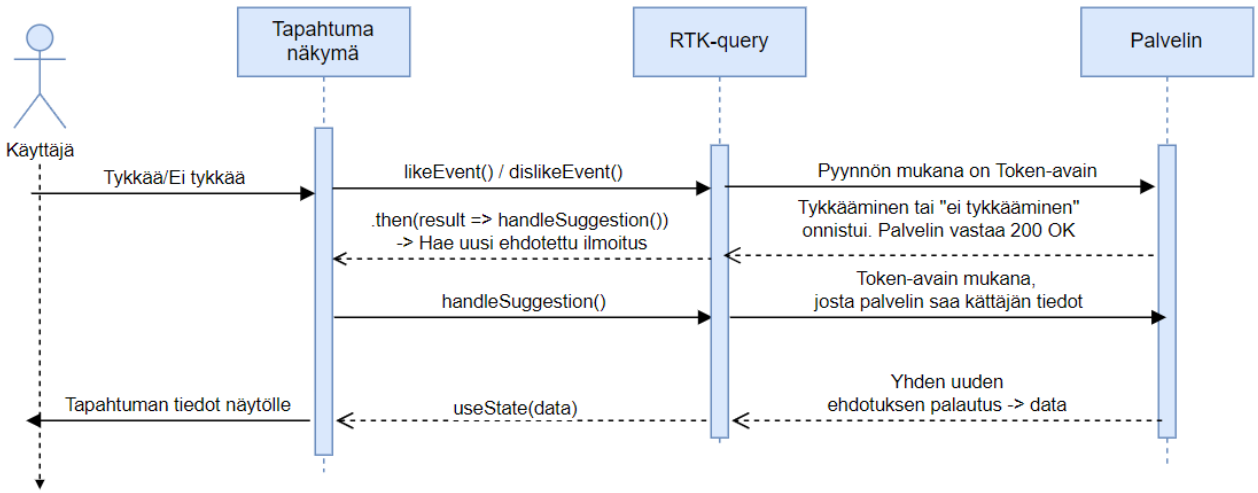
```
import { useCreateEventMutation, useUploadMutation } from
'../../../app/services/eventApi';
....

const [upload] = useUploadMutation()
const [createEvent, {isLoading}] = useCreateEventMutation()


const submitEventButton = () => {
        const data = images.map(({ base, ...images }) => images)
         createEvent({
            title: title,
            description:description,
            organizer:organizer,
            categorie: category,
            location: currentCity,
            startingTime:  date,
            endingTime: date2
        }).unwrap()
        .then((res:any)=> upload( {id: res.id, file: data})
        .unwrap().then(() => {
            successMessage();
            navigation.goBack()
        }))
        .catch(() => errorMessage())
```

```
}
....
```

7.1.4 Tapahtuman toiminta

Tapahtuman ehdotus alkaa heti, kun tapahtuma näkymää avataan ja käyttäjä voi valita, että tykkääkö tai ”ei tykkää” julkaisusta. Valinnan jälkeen uusi ilmoitus tulee näkyviin, jonka jälkeen käyttäjä voi taas lähettää uuden tykkäyspyynnön sisällyttäen Token-avaimen palvelimelle. Tykkäyspyynnön jälkeen palautuu onnistumisviesti palvelimelta, joka käynnistää asiakaspuolen sovelluksesta toisen pyynnön uudesta ehdotetusta tapahtumasta. Tämän jälkeen saadaan julkaisun tiedot, jota laitetaan näkyviin käyttäjälle useStaten avulla. Sama prosessi toistuu aina uudestaan, kun tykätään tai ”ei tykätä”. Kuvassa 22 on kuvailtu tätä samaa toimintaa kaaviolla.

Kuva 22 Sekvenssikaavio tapahtuma näkymän toiminnasta.



Funktio handleSuggestion() hallitsee tapahtumien vaihtamista näytölle. Sen sisällä on getSuggestion(), jonka tehtävänä on hakea pelkästään ehdotetun tapahtuman ID ja sen jälkeen tällä ID:n avulla haetaan varsinaista julkaisua getEvent() API-funktiolla. setOpenEvent() on sitä varten, että komponentissa voidaan tietää aktiivisena näkyvissä oleva julkaisu tallentamalla siihen tapahtuman ID numeroa. Tätä ID numero arvoa hyödynnetään esimerkiksi oikean julkaisun kommenttien avaamista varten näkymässä. Tykkäykseen liittyvät funktiot laukaisevat uuden ilmoituksen hakemista käyttämällä Promisen then ominaisuutta ja sen sisällä kutsutaan handleSuggestion().

```
const handleSuggestion = (ifReset = true) => {
    if (ifReset) {
      eventReset()
      reset()
    }
```

```
    if (!loading)
      setLoading(true)

    getSuggestion().then(async (suggestData) => {
      if (suggestData?.data[0]?.id == null) {
        return reset()
      }
      await getEvent(suggestData.data[0].id)
      setOpenEvent(suggestData.data[0].id)
})
```

7.2 Käyttöliittymän teema

Projektissa käytetään teeman ja tyylien määrittämiseen Context API, jonka tarkoituksena on jakaa kaikille komponenteille sen puun sisälle oleville teema tiedon, kuten koodissa alla. Tätä koodia on laitettu Root-komponentin ytimeen, sillä sen tarkoituksena on, että kaikki komponentti puussa perivät tätä ominaisuutta. Muuttuja Theme arvoksi on laitettu True tai False riippuen teeman sävystä, jos se on vaalea tai tumma.

```
<ThemeContext.Provider value={theme}>
            // Kaikki komponentit/komponentti puu tämän alle
</ThemeContext.Provider>
```

Contextia luodaan alla olevan koodin tavoin. Tämän jälkeen sitä voidaan ottaa käyttöön Root-komponentissa.

```
//Contextin luonti. ThemeContext.ts
import { createContext } from 'react';
export const ThemeContext = createContext(false);
```

Komponentit, kuten alla oleva pystyvät käyttämään ”useContext(ThemeContext)”, jonka avulla ne saavat teematilan, kun teemaatila muuttuu niin Contextin arvo muuttuu ympäri sovellusta. Style-tyylien kohdassa hyödynnetään isDark, jonka perusteella valitaan tumma tai vaalea teema. Nämä mainColors- ja primaryColor-värit ovat määritelty toiseen tiedostoon, josta niitä helposti voidaan ottaa käyttöön uudelleen muissa komponenteissa ja pitää sama sävyä ympäri sovellusta.

```
export function CustomButton({style, title, onPress}:Props) {

  const isDark = useContext(ThemeContext);
```

```
  return (
    <TouchableOpacity
    style={[{...style,backgroundColor:
selected?primaryColor:(isDark?mainColors.DarkBG.backgroundColor:mainColors.lig
htBG.backgroundColor)},]}
    onPress={onPress}>
      <Text
      style={[styles.text,
isDark?mainColors.DarkText:mainColors.lightText,selected?mainColors.DarkText:n
ull]}>
          {title}
      </Text>
    </TouchableOpacity>
  );
}
```

8 Haasteet ja johtopäätökset

Projektissa haasteita oli paljon, kuten eri näytöille optimointia, käytettävien pakettien valinta ja paljon huolia tuli vastaan ylläpitämisestä tulevaisuudessa. Näytöillä ongelmana on, että on monta erikokoista/muotoista puhelinta ja jokaisella on vähän eri käyttöliittymä varsinkin Android järjestelmää käyttävillä puhelimilla, jossa esimerkiksi navigointi systeemi toimii eri tavoin. Lisäksi pienemmillä näytöillä komponentti koko saattoi olla liian suuri haluttuun suhteeseen verrattuna, vaikka siitä voisikin tehdä suhteellisesti skaalautuvan, niin haasteena saattaisi tulla vastaa esimerkiksi liian pieni fonttikoko tai liian pieni sisältö. Vältin myös ongelmia käyttämällä NPM eli Node Package Managerin avulla asennettuja paketteja, sillä natiivikoodin kirjoittaminen on aikaa vievää ja nopeasti vanhentuvaa. Tärkein asia niissä on selvittää, että onko paketilla iso ja hyvä yhteisö sekä katsomalla sitä, että milloin pakettia on viimeksi päivitetty. Nämä huomattavasti vaikuttavat projektin ikään.

Lisäksi suurin ongelma, joka esiintyy JavaScriptiä käyttävissä kehitysalustoissa, on nopea kehitys ja muutos. Jo kuukausien sisällä tulee paljon uusia ominaisuuksia, jotka saattavat vanhentaa jo tuttujen pakettien ominaisuuksia. Tämä pakottaa yleisesti kehittäjiä joko tekemään säännöllisiä päivityksiä tai edes pysymään, jollain tavalla mukana muutosten kanssa. Tulevaisuuden kannalta joutuu miettimään varmaankin jopa kokonaan uudelleen kirjoitusta, kun joitakin päivityksiä ei vaan voida integroida sovellukseen mahdollisen rikkoutumisen vuoksi. Uudelleenkehitys on kallis prosessi ja haittana on, että tämä pakottaa kehittäjiä sellaiseen tilaan, että koodia ei muuteta, jos se ei ole rikki. Onneksi turvallisuuden kannalta uudet tulevat haavoittuvuudet eivät ole niin vakavia, sillä React Native on oikeastaan käyttäjäpuolen koodia, jossa turvallisuus haavoittumat eivät tuota suurta ongelmaa niin kauan kun yksityistiedot ovat enkryptoituja eli salattuja.

Natiivi Android ja IOS sovelluksen kehittäminen on haastavampaa ja aikaa vievää, koska se vaatii kahden eri alustan oppimista sekä projektin valmistuessa pitää ylläpitää nämä koodit. Android ja IOS natiivi APIt eivät myöskään ole täydellinen ratkaisu, vaan joka vuosi tulee uusia API versioita, joissa asioita vanhenee ja uusia ominaisuuksia tulee, jotka saattavat helpottaa tai jopa vaikeuttaa tunnettuja asioita.

NPM paketteja valittaessa on tärkeää huomioida lisenssiä, sillä jotkut antavat oikeuksia ja toiset ovat enemmän rajoittavampia sekä vaativat lisätoimenpiteitä. MIT-lisenssi on aika yleisesti käytetty suurimmassa osassa kirjastoissa, koska sitä pystyy käyttämään esimerkiksi myyntikäyttöön ja se ei vaadi sitä, että kirjoittamasi tai muokkaamasi koodia pitää laittaa julkiseksi kuten jotkut lisenssit saattavat pyytää. Nämä asiat kannattaa huomioida jo ennen projektin alkua.

Pysyin itse projektin aikataulun kanssa mukana hyvin ja projektissa on kaikki tarvittavat ominaisuudet, joita olin suunnitellut työtä varten. Jatkokehityksessä voisi tehdä palvelin puolella monimutkaisempia algoritmeja julkaisujen ehdotukseen. Lisäksi chat-palvelua voisi olla sekä jonkinlainen kaverilista, jonka avulla pystytään näkemään mistä omat kaverit tykkäävät ja lähetellä tapahtumia toisilleen. Julkaisuihin voisi sisältää käyttäjille jonkinlaisen ”Verified” merkinnän, jossa erotellaan viralliset tapahtumat yksityisistä tapahtumista. Sovellusta on tehty myös jatkokehitystä varten helposti laajennettavan, sillä tarvetta on web-sovellukseen, jossa palvelun käyttäjät pystyisivät katsomaan tapahtumia selaimen kautta käyttäjällä tai ilman. Tämä mahdollistaisi sitä, että tapahtumia pystyttäisiin katsomaan mistä tahansa laitteelta, jossa on nettiyhteys.

Lopulta projekti onnistui hyvin ja se on melko toiminnallinen sekä sitä pystyy kokeilemaan. Työn aikana paljon oppimista tapahtui ja koin uusia asioita sekä myös elämään yleisesti liittyviä havaintoja tuli vastaa. Aina ei kannata miettiä parasta mahdollista ratkaisua, vaan kannattaa lähteä tekemään sopivalla tavalla ja jokaisella työkalulla on omat vahvuudet ja heikkoudet. Tulevaisuuden kannalta mikään ei ole pysyvä ratkaisu sekä aina kannattaa ottaa mieluummin sellaisen asenteen, että ollaan valmiina uusiin muutoksiin ja haasteisiin. Sama pätee myös ohjelmointikieliin, että tietoisuus monista työkaluista auttaa varmistamaan mukana pysymistä jatkossa. Huonopuoli tästä on se, että nykyään kehitys on nopeatahtista ja eikä pääse erikoistumaan erityisesti mihinkään.

Lähteet

Abdallah, A. (14.04.2023). *How the New Architecture in React Native Improves Performance and Developer Experience*. https://www.linkedin.com/pulse/how-new-architecture-react-native-improves-developer-abdallah

Aranda, M. (28.10.2017). *What's the difference between JavaScript and ECMAScript?* https://www.freecodecamp.org/news/whats-the-difference-between-javascript-and-ecmascript-cba48c73a2b5/

Atukorala, S. (15.1.2022-a). *React Components and Props Explained!* https://medium.com/@SeanAT19/react-components-and-props-explained-872adadee64f

Atukorala, S. (15.1.2022-b). *React Components and Props Explained!* [Kuva]. https://medium.com/@SeanAT19/react-components-and-props-explained-872adadee64f

Briggs, T. (7.12.2023). *Persist state with Redux Persist using Redux Toolkit in React.* https://blog.logrocket.com/persist-state-redux-persist-redux-toolkit-react/

Budziński, M. (2023). *What Is React Native? Complex Guide for 2023*. Haettu 1.4.2024 osoitteesta https://www.netguru.com/glossary/react-native

De Roy, S. (27.7.2022-a). *What is Redux? Store, Actions, and Reducers Explained for Beginners*. https://www.freecodecamp.org/news/what-is-redux-store-actions-reducers-explained/

De Roy, S. (27.7.2022-b). *What is Redux? Store, Actions, and Reducers Explained for Beginners* [Kuva]. https://www.freecodecamp.org/news/what-is-redux-store-actions-reducers-explained/

Dvalashvili, D. (7.3.2024). *Exploring Redux with React: From Basics to Advanced with Redux Toolkit*. https://davitdvalashvili1996.medium.com/exploring-redux-from-basics-to-advanced-with-redux-toolkit-3c3b2feb8b85

Esa, M. (17.10.2023). *What is useState in React?* https://dev.to/mikhaelesa/what-is-usestate-in-react-47io

Flutter. (n.d.). *Impeller rendering engine*. Haettu 13.8.2024 osoitteesta https://docs.flutter.dev/perf/impeller

Gupta, A. (12.4.2024). *The Top Frameworks to Use in C# Development.* https://medium.com/@sharad.qwi/the-top-frameworks-to-use-in-c-development-345ea841eb66

Herrera, E. (18.3.2022). *Understanding Redux Saga: From action creators to sagas.* https://blog.logrocket.com/understanding-redux-saga-action-creators-sagas/

IBM, M. (n.d.). *What is a REST API?* https://www.ibm.com/topics/rest-apis

Khan, S. (13.4.2021). *How to Secure Sensitive Mobile App Data by using React Native Keychain*. https://javascript.plainenglish.io/securing-sensitive-mobile-app-data-by-using-react-native-keychain-dc0e23f7e331

Kinsta. (25.5.2023). *The Uncomplicated Guide To JSX Syntax*. https://kinsta.com/knowledgebase/what-is-jsx/

Madushan, A. (26.1.2024). *Vite vs Create React App in 2024*. https://blog.bitsrc.io/vite-vs-create-react-app-326e8cc2c46b

Mehul, M. (7.5.2023). *Understanding JavaScript Generators and Their Use in Node.js*. https://codedamn.com/news/nodejs/javascript-generators

Meta. (28.10.2021). *The Facebook Company Is Now Meta.* https://about.fb.com/news/2021/10/facebook-company-is-now-meta/

Microsoft, (28.05.2020-a). *What is Xamarin?* https://learn.microsoft.com/en-us/xamarin/get-started/what-is-xamarin

Microsoft, (28.05.2020-b). *What is Xamarin?* [Kuva]. https://learn.microsoft.com/en-us/xamarin/get-started/what-is-xamarin

Mohan, M. (9.10.2023). *What is JSX? Why React.js uses JSX?* https://codedamn.com/news/reactjs/what-is-jsx-why-react-js-uses-jsx

Moota, A. kumar. (9.2.2024). *Building a React Redux Counter App: A Step-by-Step Guide*. https://medium.com/@m.anilkumar51/building-a-react-redux-counter-app-a-step-by-step-guide-f187e7104986

Pregasen, M. (16.2.2023). *Putting the Expo vs React Native debate to rest*. https://retool.com/blog/expo-cli-vs-react-native-cli

Rahman, A. (25.8.2023). *Problems & Benifits with Redux.* https://satiqurrahman.medium.com/problems-benifits-with-redux-5cd84254b344

React Native (n.d.-a). *AsyncStorage React Native*. Haettu 21.7.2024 osoitteesta https://reactnative.dev/docs/asyncstorage

React Native. (n.d.-b). *Setting up the development environment*. Haettu 3.4.2024 osoitteesta https://reactnative.dev/docs/environment-setup

Redux. (n.d.-a). *Redux Toolkit: Overview*. https://redux.js.org/redux-toolkit/overview

Redux. (n.d.-b). *Writing Logic with Thunks.* https://redux.js.org/usage/writing-logic-thunks

Reduxjs/redux-thunk. (2024). *redux-thunk*. Haettu 29.6.2024 osoitteesta. https://github.com/reduxjs/redux-thunk

Saini, S. (11.1.2022-a). *Benefits of Using React Native for Mobile App Development*. https://www.grazitti.com/blog/7-benefits-of-using-react-native-for-mobile-app-development/

Saini, S. (11.1.2022-b). *Benefits of Using React Native for Mobile App Development* [Kuva]. https://www.grazitti.com/blog/7-benefits-of-using-react-native-for-mobile-app-development/

Shah, V. (24.2.2024). *Why Use React?* Haettu 28.3.2024 osoitteesta https://www.tatvasoft.com/blog/why-use-react/

Shubel, M. (26.7.2022). *What Is TypeScript?* https://thenewstack.io/what-is-typescript/

Soeiro, C. (23.10.2023). *A Brief Summary of the Main Features of TypeScript*. https://blog.stackademic.com/a-brief-summary-of-the-main-features-of-typescript-7fd97d576297

Thirunaukkarasu, R. (25.5.2024). *NET MAUI vs Xamarin: A Comparison of the Key Differences and Advantages*. https://kanini.com/blog/net-maui-vs-xamarin/

Vite. (n.d.). *Getting Started.* https://vitejs.dev/guide

Wadhwani, P. & Verma, R. (2024). *NativeScript vs React Native: Which one to choose in 2024?* Haettu 3.4.2024 osoitteesta https://www.bacancytechnology.com/blog/nativescript-vs-react-native

Walburg, M. (6.2.2022). *What are the best alternatives to React Native?* https://binarapps.com/what-are-the-best-alternatives-to-react-native/

Web Reference. (n.d.). *ECMAScript Versions in JavaScript*. Haettu 29.3.2024 osoitteesta https://webreference.com/javascript/basics/versions/