Haaga-Helia
ammattikorkeakoulu Oy

Ampparit.com mobiilisovellus React Nativella

Miikka Huuskonen

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2018

Tiivistelmä

Haaga-Helia
ammattikorkeakoulu Oy

Tekijä(t) Miikka Huuskonen	
Koulutusohjelma Tietojenkäsittelyn koulutusohjelma	
Raportin/Opinnäytetyön nimi Ampparit.com mobiilisovellus React Nativella	**Sivu- ja liitesivumäärä** 31 + 2

Opinnäytetyön tarkoituksena on selvittää Ampparit.com sivuston toteuttamista React Native ohjelmistokehyksen avulla. Työssä käydään läpi Ampparit.com-sivuston arkkitehtuuria niiltä osin, kuin se on mobiilisovelluksen kannalta oleellista.

Työssä pohditaan erilaisia näkökulmia toteuttaa mobiilisovellus valmiin web-sivun pohjalta sekä vertaillaan eri tekniikoita React Nativeen. Työssä tutkitaan myös React Nativen ja ReactJS yhtäläisyyksiä ja eroavaisuuksia sekä minkälaisia vaikutuksia, hyötyjä tai haittoja, keskinäisessä kehityksessä on.

Opinnäytetyön ohessa toteutetaan myös mobiilisovellus, jonka aiheena on React Nativella toteutettu versio Ampparit.com-sivustosta, joka itsessään on ReactJS pohjainen. Mobiilisovelluksen toteutusprosessin osana esitellään React Nativella toteutettavan mobiilisovelluksen aloitus sekä keskeiset kehitettävät asiat, mutta myös tutustutaan työn edetessä tulleisiin haasteisiin.

Työn lopuksi pohditaan jatkokehitysideoita mobiilisovelluksen suhteen, mutta myös eteen tulleita haasteita ja työn edetessä tapahtunutta omaa oppimista.

Asiasanat
React, Mobiilikehitys, Javascript, Android, iOS

Sisällys

1 Johdanto

Ampparit Oy:n omistama Ampparit.com on vuonna 2004 avattu verkkosivusto, joka kerää yli 25 000 uutisotsikkoa viikossa noin 300 eri lähteestä lähteen hyväksynnällä ja kategorisoi ne aihealueiden mukaan eri kategorioihin. Palvelun perustivat vuonna 2004 Joensuussa Petteri Hannonen ja Florian Berger, jonka jälkeen palvelu on siirtynyt Otava-konsernin hallintaan vuonna 2012. Palvelulla on jopa 450 000 viikoittaista käyttäjää, joista noin 32% on mobiilikäyttäjiä sekä noin 15% tablettikäyttäjiä. Palvelusta on olemassa myös mobiilisovellukset iOS- ja Android-alustoille. (Isohanni, 2018)

Ampparit.com sivusto on kehitetty React JavaScript kirjastolla sekä mobiilisovellus iOS alustalle on kehitetty Swift-ohjelmointikielellä ja Android alustalle Java-ohjelmointikielellä. Työn kirjoitushetkellä mobiilisovelluksiin tehtävät muutokset tilataan alihankkijana toimivalta ohjelmistokehitystalolta. Sovellukset kulkevat omilla kehityspoluillaan ja ovat irrallaan muusta palvelun kehityksestä. Opinnäytetyössä tarkastellaan mahdollisuutta kehittää palvelusta mobiilisovellukset React Nativen avulla, jolloin yhdenmukainen kehittäminen sivuston ja mobiilisovellusten kesken voisi helpottua lähdekoodien yhdenmukaisuuden avulla. Samalla saataisiin myös parempi tuki eri kokoisille näytöille mobiililaitteissa React Nativen skaalautuvuuden avulla, jolloin tavoitettaisiin mahdollisesti myös enemmän tablettikäyttäjiä sovelluksen pariin.

Opinnäytetyön aiheen valinta on tullut omasta mielenkiinnosta React Nativella toteutettuja mobiilisovelluksia kohtaan, mutta myös aidosta tarpeesta selvittää voidaanko kehitystyötä Ampparit.com:in osalta keskittää ja sujuvoittaa käyttämällä sekä itse sivustolla, että mobiilisovelluksissa samaan ideologiaan keskittyviä teknologioita. Työn tekoa tukee myös työskentely asiakkaalle Ampparit.com -sivuston parissa, jonka perusteella osa tietoperusteesta rakentuu kokemukseen. Opinnäytetyön onnistumisen kannalta oleellista on, että tekijänä ymmärrän aiheen sekä opiskelen siihen liittyvät teknologiat.

1.1 Tavoitteet ja rajaukset

Tämän opinnäytetyön on tarkoitus selvittää migraation mahdollisuutta Ampparit.com sivusta React Nativella toteutettuun versioon mobiilisovelluksesta ja kuinka se sujuvoittaisi kehitystä erilaisten alustojen välillä. Opinnäytetyössä selvitetään myös yhdenmukaistamisen merkitys tulevalle kehitykselle. Työssä tutustutaan mobiilisovelluksen osalta oleellisiin osiin Ampparit.com arkkitehtuurista, jossa selvennetään tehtyjä valintoja teknologian osalta. Käsitellään myös vaihtoehtoisia toteutustapoja sekä teknologioita ja pohditaan vertaillen niihin liittyviä mahdollisuuksia.

Projektin yhteydessä luodaan kevennetty versio Ampparit.com mobiilisovelluksesta React Nativen avulla, jonka avulla kehitystä edellä mainitulla kielellä pyritään ymmärtämään paremmin sekä selvittämään minkälaisia mahdollisuuksia tai haasteita kehityksessä voi olla. Kehitettävä sovellus voisi toimia myös runkona kehitettävälle mobiilisovellukselle tulevaisuudessa.

Ampparit.com on mainosrahoitteinen palvelu, joten kehityksen yhteydessä tutkitaan myös mainosintegraation mahdollisuutta, mutta sitä ei toteuteta mobiilisovelluksen kevennettyyn versioon.

Opinnäytetyön dokumentaation ymmärtämiseksi aikaisempi kokemus ohjelmoinnista sekä Reactista teknologiana ei ole välttämätöntä, mutta siitä on hyötyä. Työ ei varsinaisesti käsittele React Nativea teknologiana, vaan sen käyttöä mobiilisovelluksessa, jonka idea periytyy Reactilla toteutetusta sivusta. Jatkokehityksen kannalta teknisiä ratkaisuja käydään kuitenkin läpi niihin kohdistuneiden valintojen ymmärtämiseksi, jotta jatkossa niiden hyödyntäminen olisi helpompaa.

1.2 Tietoperusta

Opinnäytetyön dokumentoitu osuus perustuu tekijän aikaisempaan kokemukseen ja työssä opittuihin asioihin Ampparit.com -sivuston kehityksen yhteydessä ja relevantteihin lähteisiin, kuten Facebookin tuottamaan React Nativen verkkodokumentaatioon ja asianmukaisiin artikkeleihin. Myös kirjallisia lähteitä käytetään, mutta johtuen käytettävän teknologian nopeasta kehitys- ja muutosvauhdista, ajantasaisin tieto on aina verkossa. Työssä on hyödynnetty myös asiantuntijalähteitä tekijän työnantajan, Aste Helsinki Oy:n puolelta sekä Ampparit Oy:n puolelta, jotta esitetyt tiedot Ampparit.com sivuston rakenteesta ja sovelluksiin vaikuttavista asioista olisivat mahdollisimman oikeita, mutta eivät kuitenkaan sisältäisi liiketoiminnan kannalta salaista tietoa.

Opinnäytetyössä esiteltävät lähdekoodiesimerkit pohjautuvat React Nativen alkuperäisen dokumentaation malliin, mutta myös lähdeartikkelien mukaisiin yleisiin käytäntöihin. Työssä ei esitellä varsinaisen toteutettavan sovelluksen lähdekoodin sellaisia kohtia suoraan, jotka perustuvat suojattuun tietoon, vaan ne esitellään malliesimerkeillä.

2 Ampparit.com arkkitehtuuri

Nykyisen Ampparit.com:in arkkitehtuuri rakentuu Reactilla toteutetusta käyttöliittymästä. Käyttöliittymää palvelevat uutiskeräimet, rajapinta ja tietokanta, joita hallitaan ylläpitokäyttöliittymistä. Kaiken tämän lisäksi Amppareihin tulee erillisiltä rajapinnoilta säätietoja, tv-ohjelmatietoja sekä urheilutuloksia.

2.1 React käyttöliittymä

Käyttöliittymäksi Ampparit.com:issa on valikoitunut yhteisöpalvelu Facebookin kehittämä ReactJS JavaScript kirjasto yhdessä Redux-tilanhallintakirjaston kanssa. React käyttöliittymän taustalla toimii NodeJS, joka suorittaa palvelimella ja generoi HTML-sivun sekä lähettää sivun tiedot kokonaisuudessaan selaimelle, joka piirtää (renderöi) sivun valmiiksi käyttäjää varten.

Reactin valintaan käyttöliittymäksi on vaikuttanut sen dynaamisuus ja modulaarisuus, jolloin yksittäisiä elementtejä sivulle on helpompi kehittää sekä responsiivisuus, joka on tärkeä kriteeri mobiilikäytön kannalta; mobiililaitteiden yleistymisen myötä sivuston käyttö on siirtynyt osittain mobiililaitteille. React on myös suosittu kehittäjien kesken, jolloin jatkokehityksen onnistuminen on myös varmempaa.

2.2 Ohjelmointirajapinta eli API

Amppareiden API tarjoilee React käyttöliittymälle uutisdatan sekä ylläpitää käyttäjädataa. API:a käyttävät hyödyksi myös jo olemassa olevat mobiilisovellukset sekä tämän projektin yhteydessä tehtävä React Nativella toteutettava mobiilisovellus. (Korhonen, 2018)

2.3 Tietokanta

Tietokantaan tallennetaan kaikki data, mitä ylläpitokäyttöliittymä saa uutiskeräimiltä sekä mitä React käyttöliittymä palauttaa API:n kautta käyttäjän toimesta, kuten personoidut uutislistat, uutisten saamat äänet, uutisten klikkaukset sekä muuta palvelun käyttöön liittyvää tietoa.

2.4 Ylläpitokäyttöliittymä

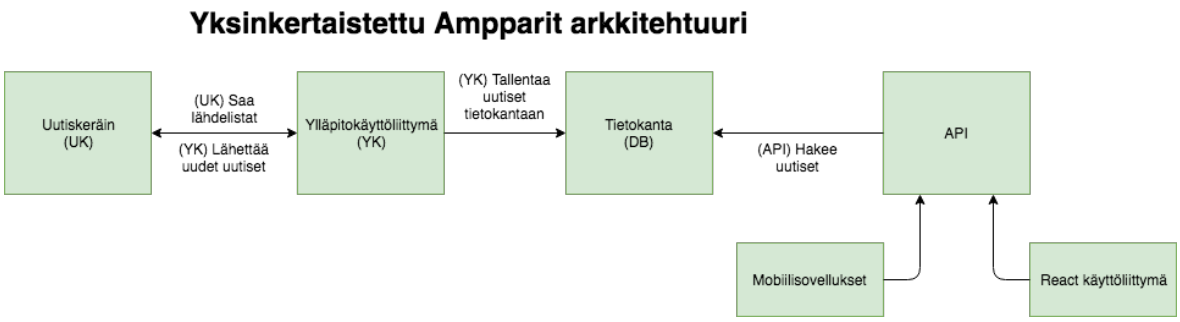
Ylläpitokäyttöliittymä pitää yllä lähdelistaa ja tallentaa uudet uutiset sekä muut tiedot tietokantaan. Käyttöliittymästä hallinnoidaan uutisten kategorioita, kategoriatasoja sekä määri-

tellään suosittuja uutisaiheita. Ylläpitokäyttöliittymä hallinnoi lähteitä ja sisältöä automatisoidusti, mutta siinä on myös mahdollista määritellä käsin uutisten näkyminen Ampparit.com:issa. Ylläpitokäyttöliittymän kautta hallitaan myös palvelun käyttäjien tietoja.

2.5 Uutiskeräimet

Uutiskeräimet lukevat lähteiden RSS-syötteet, joiden perusteella uutiset kategorisoidaan aihealueittain ylläpitokäyttöliittymässä. Uutiskeräimet hakevat listan luettavista lähteistä ylläpitokäyttöliittymältä ja tarkistavat lähteet säännöllisin välein. Löydetyt uudet uutiset lähetetään ylläpitokäyttöliittymälle, joka käsittelee ne. Uutiset kerätään niiden lähteiden suostumuksella, jolloin Ampparit.com saa sisältöä ja uutiset lukijoita. (Kinnunen, 2018)

Kuvassa 1 selvennetään uutisdatan päätyminen edellä esitettyjen eri vaiheiden kautta uutiskeräimiltä aina eri käyttöliittymiin asti.



Kuva 1. Yksinkertaistettu Ampparit arkkitehtuuri

3 Sovelluksen toteutustapoja

Mobiilisovelluksia voidaan toteuttaa eri tavoin. Mobiilisovellusten toteutustavat voidaan jakaa muutamaan eri kategoriaan: natiivisovelluksiin, web-pohjaisiin mobiilialustoihin sekä hybridisovelluksiin.

3.1 Natiivisovellus

Natiivisovellukset ovat yksittäisten alustojen omilla kehitysvälineillä toteutettuja sovelluksia. Yleensä Android-sovelluksia kehitetään Java-ohjelmointikielellä ja Applen iOS-sovelluksia Swift- tai Objective-C-ohjelmointikielillä. Molemmille alustoille löytyy oma kehitysympäristönsä, joista iOS:lle on ominaista myös kehitykselle vaadittu Applen macOS-käyttöjärjestelmä, koska kehitys vaatii Applen Xcode IDE.n, joka on saatavilla ainoastaan macOS -järjestelmiin. Android järjestelmälle kehitys onnistuu sekä Linux, macOS että Windows käyttöjärjestelmillä, joihin saadaan asennettua tarvittava kehitysympäristö, Android SDK.

Natiivisovellusta kehittäessä voidaan käyttää natiiveja komponentteja, jolloin saadaan aikaan myös mahdollisimman sujuva käyttökokemus. Samalla päästään myös sujuvasti käsiksi itse laitteistoon sekä eri sensoreihin, mikä takaa myös eri ominaisuuksien helpon lisäyksen sovellukseen. Myös laitteen suorituskykyä hyödynnetään mahdollisimman optimaalisesti natiivisovelluksissa.

Natiivisovelluksen kehityksessä haittapuolena on kuitenkin se, että jokaiselle alustalle kehitys on tehtävä erikseen, jolloin sovelluksen saaminen kaikille halutuille alustoille on hidasta. Kehitys saattaa vaatia laajaa osaamista yhdeltä kehittäjältä tai isomman tiimin, jossa kehitys tapahtuu jaetulla osaamisella. (Ristola, 2016)

3.2 Web-sovellus

Web-pohjaisia mobiilialustoja toteutetaan käyttämällä HTML:ää ja JavaScriptia sekä tyylit käyttämällä CSS:ää, jolloin kehitykseen ei tarvita natiivia osaamista, vaan se voidaan toteuttaa web-kehittäjien toimesta. Web-pohjaisten ohjelmien käyttökokemus jää usein jälkeen natiivisovelluksiin verrattuna, mutta niillä saadaan aikaiseksi laajempi laitetuki itse käyttöliittymän osalta. (Ristola, 2016) Natiivin toteutuksen hyötynä verrattuna web-pohjaiseen toteutukseen on kuitenkin jatkuva pääsy sijaintiin, ilmoitukset sekä tilan muistaminen. (Tolvanen, 2016)

3.3 Hybridisovellus

Natiivisovellusten kehityksen ollessa hidasta, on esille nostettu toteutuksia, jotka yrittävät hahmottaa natiiveja komponentteja yleisiksi komponenteiksi ja jotka toimisivat eri alustoilla samalla tavalla. Eri mobiilialustoja on muutamia erilaisia, joista vallitsevat tällä hetkellä ovat Android ja iOS, kuten taulukosta 1 voidaan päätellä.

Taulukko 1. Maailmanlaajuinen älypuhelimien myyntitilasto käyttöjärjestelmittäin 2017 (Gartner 2018)

Käyttöjärjes-telmä	2017 Yksikkömäärä (tuhatta)	2017 Mark-kinaosuus	2016 Yksikkömäärä (tuhatta)	2016 Mark-kinaosuus
Android	1,320,118.1	85.9%	1,268,562.7	84.8%
iOS	214,924.4	14.0%	216,064.0	14.4%
Muut	1,493.0	0.1%	11,332.2	0,8 %
Yhteensä	1,536,525,5	100 %	1,495,959,0	100 %

Hybridisovelluksia voidaan kehittää erilaisilla alustoilla, kuten esimerkiksi vapaanlähdekoodin projekti NativeScript, Microsoftin kehittämä Xamarin, Googlen kehittämä Flutter sekä Facebookin kehittämä React Native, jota myös käytetään tämän työn ohessa toteutettavassa Ampparit.com mobiilisovelluksessa.

Hybridisovellusten tavoitteena on, että toteutusta ei tarvitsisi tehdä kaikille alustoille kokonaan, vaan suurimmaksi osaksi käytetään yhtä koodikantaa ja vain se osa toteutuksesta, joka täytyy tehdä alustakohtaisesti, tehtäisiin erikseen. Tällaisia erikseen toteutettavia osia voivat olla esimerkiksi käyttöliittymäkohtaiset tyylit tai navigointielementit. Tällöin säästetään aikaa ja saadaan samankaltainen tuotos aikaiseksi vähemmällä vaivalla. Natiivikehittäjien tarve vähenee ja osa kehityksestä onnistuu esimerkiksi Web-kehittäjien tai muiden kehittäjien toimesta. Käyttökokemus saadaan lähelle natiivia sovellusta, mutta kehityksessä olisi hyvä olla myös natiivikehityksen osaaja, koska projekti täytyy paketoida alustojen omia mobiilikauppoja varten. (Ristola, 2016)

3.4 Vaihtoehtoja React Nativelle

Luvussa käydään läpi vaihtoehtoisia hybridisovellustekniikoita mukaan lukien Flutter, NativeScript, Ionic sekä Xamarin. Vaikka opinnäytetyö perustuu React Nativen käyttöön, on hyvä käydä läpi myös muita mahdollisia ohjelmistokehyksiä, jotka perustuvat avoimeenlähdekoodiin.

3.4.1 Flutter

Flutter on Googlen kehittämä ja 2016 julkaistu avoimenlähdekoodin mobiilikehitysympäristö, jolla voidaan tuottaa sovelluksia yhdellä koodikannalla sekä Android- että iOS-alustalle React Nativen tapaan. Flutter-sovellus kirjoitetaan Dart-ohjelmointikielellä, jonka syntaksi on samankaltaista kuin Java-, JavaScript-, C#- tai Swiftft-ohjelmointikielillä, ja alun perin Google onkin kehittänyt Dart:ia JavaScriptin korvaajaksi, mutta sovelluksissa käytetään myös C- ja C++-ohjelmointikieliä. Lisäksi 2D-kuvantamisesta huolehtii Skia-kuvantamismoottori. (Wikipedia, 2018b)

Flutter-sovelluksen näkyvimpiä elementtejä ovat widgetit eli pienet erilliset ohjelmat. Widgetit vaikuttavat ja kontrolloivat näkymiä sekä myös käsittelevät ja vastaavat käyttäjän toimintaan. Widgetit ovatkin suuressa osassa Flutter sovelluksen tehokkuudessa. Toisin kuin React Nativessa, Flutterissa ei erikseen oteta yhteyttä käyttöjärjestelmän komponentteihin, mikä takaa tehokkuuden.

Vaihtoehtona Flutter ei ehkä vielä korvaa esimerkiksi React Nativea hybridisovelluksen tuottamisessa, koska itse Flutterin kehitys on vielä kesken, mutta esimerkiksi Google käyttää sitä omissa sovelluksissaan ja ohjelmat julkaistaan loppukäyttäjille; esimerkiksi Google Ads mainonnanhallintaohjelma on luotu Flutterilla. Google on pyrkinyt Flutterissa tuomaan esille visuaalisuuden merkitystä kehityksessä, kuten kuvaa 2 tarkastelemalla voidaan todeta ja React Nativen perustuessa enemmän pelkkään komponentin sisältöön, kuten kuvasta 3 voidaan todeta. Flutterin tunnuslause onkin: *”Build beautiful native apps in record time”.* (Flutter.io, 2018a).

```dart
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Welcome to Flutter',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Welcome to Flutter'),
        ),
        body: Center(
          child: Text('Hello World'),
        ),
      ),
    );
  }
}
```

Kuva 2. Flutterin syntaksi verrattuna React Nativen syntaksiin. (Flutter.io, 2018b)

Kuva 3. React Native syntaksi (Facebook Inc, 2018h)

3.4.2 NativeScript

NativeScript on alun perin teknologiayhtiö Telerik:in kehittämä avoimenlähdekoodin ohjelmistokehys iOS- ja Android-käyttöjärjestelmille. NativeScript ohjelmat tuotetaan JavaScriptilla tai sen kaltaisilla kielillä, kuten TypeScript. NativeScriptissä on myös tuki Angular.js ja Vue.js ohjelmistokehyksille, joilla pääsääntöisesti tuotetaan verkkosivuja ReactJS tapaan, mutta NativeScript tuo mahdollisuuden tuottaa niillä myös mobiilisovelluksia. (NativeScript, 2018)

NativeScriptillä kirjoitettu sovellus voi olla huomattavasti kookkaampi, kuin React Nativella kirjoitettu vastaava sovellus. Tämä voi aiheuttaa ongelmia esimerkiksi sovelluksen kuvantamisessa, mikäli sovelluksen käyttäjällä on hidas internetyhteys. React Native sovellusta tuottaessa käytetään vain ReactJS kirjastoa, kun taas NativeScriptin kohdalla voidaan käyttää useampia eri menetelmiä ja kieliä, mikä saattaa aiheuttaa sekavuutta ohjelman lähdekoodiin. (Bohdan 2018)

3.4.3 Ionic

Ionic on avoimen lähdekoodin mobiiliohjelmaohjelmistokehys hybridisovellusten tuottamiseen, jonka ovat luoneet Max Lynch, Ben Sperry ja Adam Bradley Drifty yrityksessä vuonna 2013. Viimeisimmät versiot Ionicista perustuvat AngularJS ohjelmistokehykseen. Ionic sovelluksia voidaan kehittää muun muassa HTML ja CSS web-tekniikoiden avulla ja sitten julkaista eri alustojen sovelluskaupoissa, joista niitä voidaan asentaa laitteisiin hyödyntämällä Cordovaa, erästä mobiiliohjelmistokehystä. (Wikipedia, 2018c)

Ionic:lla voidaan kehittää iOS- ja Android-alustojen lisäksi Web-sovelluksia sekä työpöytäsovelluksia. Ionicin dokumentaatiota kehutaan ensiluokkaiseksi ja kehitystyötä helpoksi sen perustuessa TypeScriptiin. Monet Ionic:in komponentit ovat esityyliteltyjä, joten voisi

kuvitella sovellusten olevan hieman samankaltaisia. React Nativeen verrattuna Ionicin suorituskyky on heikompaa sen käyttäessä WebViewiä. (Aggarwal, A. 2017)

3.4.4 Xamarin

Xamarin on Microsoftin kehittämä ohjelmistokehys hybridisovellusten kehittämistä varten iOS, Android ja Windows alustoille. Xamarin-sovellus kehitetään C#-ohjelmointikielellä ja näin ollen se toimii täysin natiivisti jo lopetetuilla Windows phone ja Windows 10 Mobile alustoilla. Xamarinilla voidaan kirjoittaa myös macOS-ohjelmia, jotka perustuvat C#-kieleen.

Xamarin pohjautuu Microsoftin omistamaan Mono-projektiin, joka on hybridisovellustoteutus .NET ohjelmistokehyksestä. Xamarinilla kehitys on ilmaista yksityisesti ja pienillä yrityksillä, mutta suuremmalla yritystasolla kehitykseen tarvitsee maksullisen lisenssin. (Wikipedia, 2018d)

4 React ja React Native

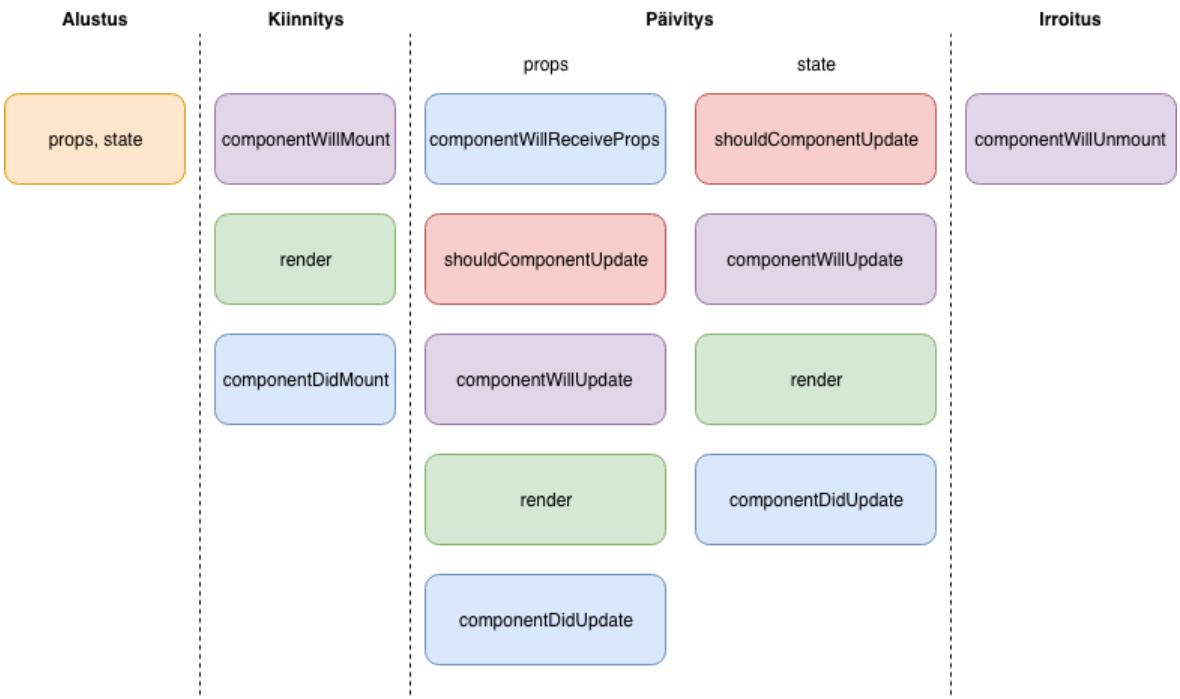
React Native kehitettiin Reactin arkkitehtuurin pohjalta mobiilisovellusten kehittämistä Android-, iOS- ja Windows-pohjaisia käyttöjärjestelmiä varten. Reactissa ja siihen perustavassa React Nativessa on paljon yhtäläisyyksiä, mutta myös paljon, jopa enemmän, eroavaisuuksia; ReactJS on JavaScript-kirjasto, kokoelma, josta kutsutaan erilaisia luokkia sekä funktioita ja joita voidaan käyttää verkkosivun rakentamiseen, kun taas React Native on ohjelmistokehys, jonka avulla muodostetaan runko kehitettävälle ohjelmalle. (Mangin, 2018)

4.1 Yhtäläisyyden ja eroavaisuudet

React Native ei käytä HTML-kuvauskieltä sovelluksen renderöimiseen eli esittämiseen, vaan saman tyyppisiä komponentteja, joita kuitenkin voidaan verrata HTML-elementteihin. Esimerkiksi React Nativessa käytettävää <View> komponenttia voidaan verrata HTML elementtiin; molemmat jakavat näkymän osioiksi, joissa varsinainen sisältö, kuten esimerkiksi tekstit, esitellään. Tästä syystä kuitenkaan ReactJS:ssä käytettäviä kirjastoja ei voida hyödyntää React Nativessa. (Sznajder, 2017)

Sekä ReactJS että React Native molemmat käyttävät hierarkiassaan komponentteja, jotka kuitenkin siis eroavat toisistaan; React Nativessa käytetään valmiita moduuleja ja komponentteja, jotka parantavat suorituskykyä. React Nativen tukeutuessa ReactJS hierarkiaan, molemmissa on samat tavat käyttää komponenttikohtaista tilaa (state) sekä ylemmiltä komponenteilta saatua attribuuttidataa (props), mutta myös samanlaiset komponenttien elämänkaarimetodit.

Reactilla ja React Nativella komponenttien elämänkaari on jaettavissa neljään vaiheeseen: alustus, kiinnitys (mounting), päivitys sekä irroitus (unmounting). Jokaiseen vaiheeseen on mahdollista lisätä metodi, jotka ajetaan eri vaiheissa elämänkaarta. Esimerkiksi dataa ei kannata hakea jatkuvalla syötöllä, koska se lisää verkkoyhteyden käyttöä ja kuluttaa myös laitteen akkua, vaan datan hakeminen voidaan sitoa esimerkiksi komponentin ”kiinnityksen” eli sen esittämisen yhteyteen, jolloin haettava data on myös ajan tasalla. Komponentin attribuutteihin liittyen on vielä lisäksi yksi metodi käytettävissä, componentWillReceiveProps, kuten kuvassa 4 voidaan todeta. (Haldar, 2017)



Kuva 4. Reactin ja React Nativen elinkaaren vaiheet ja metodit (Haldar, 2017)

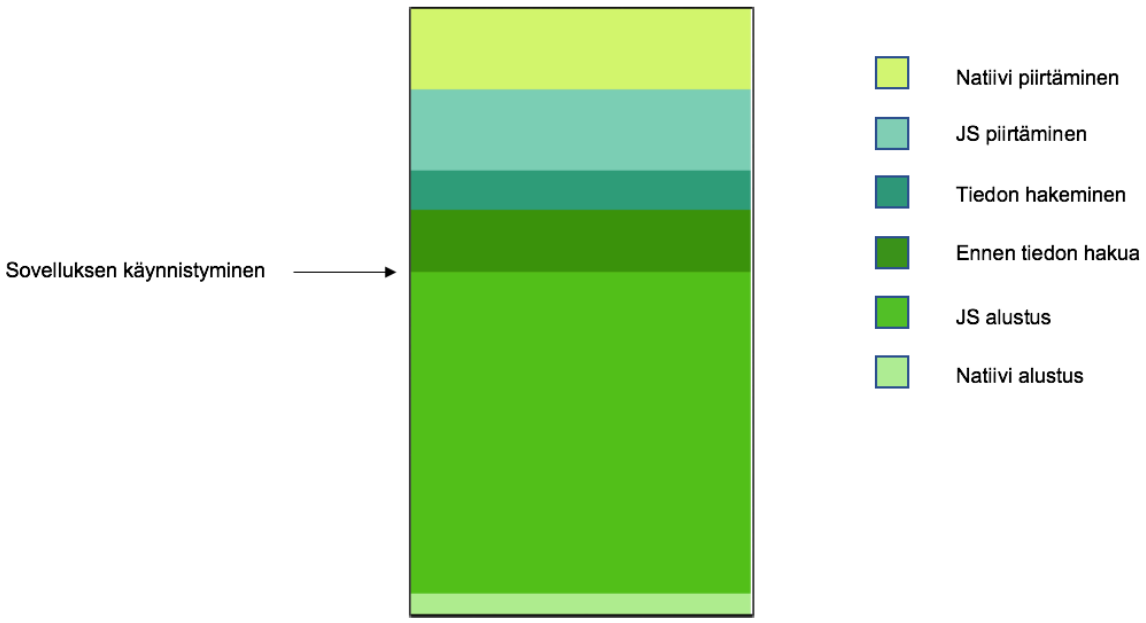
Reactin ja React Nativen palauttama koodi näyttää HTML-kielellä tehdyltä, mutta niitä kirjoitetaan yleensä JSX:ää käyttäen, joka on tapa kirjoittaa JavaScriptia ja se käännetäänkin puhtaaksi JavaScriptiksi ohjelmaa ajettaessa. JSX on myös Reactin dokumentaation mukaan suositeltu syntaksi, koska se auttaa visualisoimaan näkymän koodissa, mutta myös tuottaa parempia virheviestejä.

4.2 React Nativen suorituskyky

React Nativen suorituskyky on yksi merkittävimmistä asioista, jolla se poikkeaa monista muista hybridisovellusten toteutustavoista. React Nativen kehityksessä on pyritty ottamaan huomioon valmiiksi hyvä suorituskyky, jotta siihen ei tarvitsisi keskittyä itse sovelluksen toteuttamisen yhteydessä. React Nativella pyritään lähtökohtaisesti pääsemään suorituskyvyssä samaan lopputulemaan, kuin natiivisovelluksissa. Aina kuitenkaan samaan tasoon ei päästä. (Facebook Inc, 2018f)

Sovelluksen toimiessa nopeasti se myös lataa sisällön nopeammin. Käytöstä saadaan mukavampaa, animaatiot toimivat sujuvasti ja käyttäjät saavat enemmän vuorovaikutusaikaa sovelluksessa. Mikäli verkkoyhteys on heikko, näkyy ero nopean ja takkuilevan sovelluksen välillä heti, jolloin erotetaan myös käyttökelpoinen sovellus kelvottomasta. React Nativen kehitystiimi on panostanut etenkin vierityksen toimintaan, muistin käyttöön, käyttöliittymän reagointiin sekä ohjelman käynnistymisaikaan. Sovelluksen käynnistys on yleensä se ensimmäinen asia, jonka käyttäjä huomaa ja saa ensivaikutelman sovelluksen toiminnasta.

Kuvassa 5 nähdään sovelluksen käynnistyminen ja mitä kaikkea sen yhteydessä tapahtuu. Ensimmäisessä vaiheessa Natiivissa alustuksessa JavaScriptin virtuaalikone, natiivit moduulit alustetaan, mukaan lukien levyn välimuisti ja käyttöliittymän hallinta. Toisessa vaiheessa luetaan minifioitu JavaScript-tiedosto ja jäsennetään se. Kolmannessa vaiheessa ennen tiedon hakua ladataan ja suoritetaan sovelluskoodi. Neljännessä vaiheessa tieto (data) haetaan levyn välimuistista. JavaScriptin piirtämisessä asennetaan kaikki React komponentit ja viimeisessä vaiheessa lasketaan näkymän koko FlexBoxin asettelulle ja piirretään (renderöidään) näkymä. (De Baets, P. Lang, A. Sagnes, F. Zagallo, T. 2016)



Kuva 5. React Native sovelluksen käynnistyminen. (De Baets, P. Lang, A. Sagnes, F. Zagallo, T. 2016)

4.3 Hyödyt kehityksessä

Koska React Native on perinyt ReactJS:n arkkitehtuurin, on samanaikainen kehitys sekä web-sivujen puolella että mobiilisovelluksen puolella joustavampaa. Kehityksessä voidaan hyödyntää samaa rakennetta molemmissa tekniikoissa, jolloin uuden ominaisuuden lisääminen on helpompaa.

Kuten aiemmin on mainittu, Ampparit.com:in mobiilisovellukset on toteutettu natiiveilla kielillä, Androidille Java-ohjelmointikielellä ja iOS:lle Swift-ohjelmointikielellä. Kun muutoksia päätetään tehdä, tulee molemmat alustat ottaa huomioon erikseen, mikä vie aikaa yhdenkin muutoksen toteuttamisessa. Lisäksi toteutusten tuottaminen erikseen useammalla eri

kielellä vaatii myös osaamista niistä enemmän. React Nativella tuotettujen sovellusten perustuessa yhteen koodikantaan, saadaan muutoskin tehtyä nopeasti sekä Android että iOS alustoille. Koska Ampparit.com sivustona on toteutettu ReactJS:llä, olisi samaa kehittäjätyövoimaa mahdollista hyödyntää myös mobiilisovellusten kehitykseen, jolloin sivulle lisättyjen uusien ominaisuuksien migratoiminen sovelluksiin olisi vaivattomampaa.

4.4 React Native verrattuna natiivisovelluksiin

Vaikka React Nativessa suorituskyky on tärkeässä roolissa, ei sen käyttökokemus välttämättä ole yhtäläinen natiivisovelluksien kanssa. Koska natiivi koodi on optimoitu alustalle, käyttää se vähemmän järjestelmän ja laitteen resursseja, kuin React Nativella toteutettu sovellus. Laitteen toiminnassa muutokset näkyvät käyttäjälle siinä vaiheessa, kun esimerkiksi muita ohjelmia suoritetaan taustalla, jolloin myös ne vievät osan laitteen resursseista. Tällöin sovelluksen toiminta saattaa hidastua ja näkyä nykimisenä ruudulla.

Ampparideen mobiilisovelluksissa eniten liikettä tapahtuu näkymästä toiseen siirtyessä sekä mainoksissa, joissa esitellään liikkuvaa kuvaa ja jolloin merkittävää ongelmaa tai käytön sujuvuutta ei pitäisi päästä syntymään.

Suurimpia haittoja React Native kehityksessä on, että siinä ei tueta kaikkia natiiveja ohjelmointirajapintoja, mutta yleisimpiä kuitenkin. Tällöin tulee lisätä natiivia koodia, josta yleensä pyritään pääsemään eroon, kun valitaan React Native kehitysteknologiaksi.

5 Sovelluksen toiminnan kuvaus ja toteutusprosessi

Projektin tavoitteena on toteuttaa kevennetty Ampparit.com mobiilisovellus React Nativella iOS- ja Android-alustoille sekä mahdollistaa koodin käyttö runkona jatkokehitystä ajatellen, mutta toteutettavan sovelluksen ei ole ainakaan tässä vaiheessa tarkoitus korvata jo olemassa olevia Ampparit.com mobiilisovelluksia. Tavoitteena on myös ymmärtää mobiilikehitystä React Nativen avulla, sekä mahdollisia haasteita, mutta myös mahdollisuuksia, joita sovelluksen yhteinen koodikanta iOS- ja Android-alustojen kanssa luo.

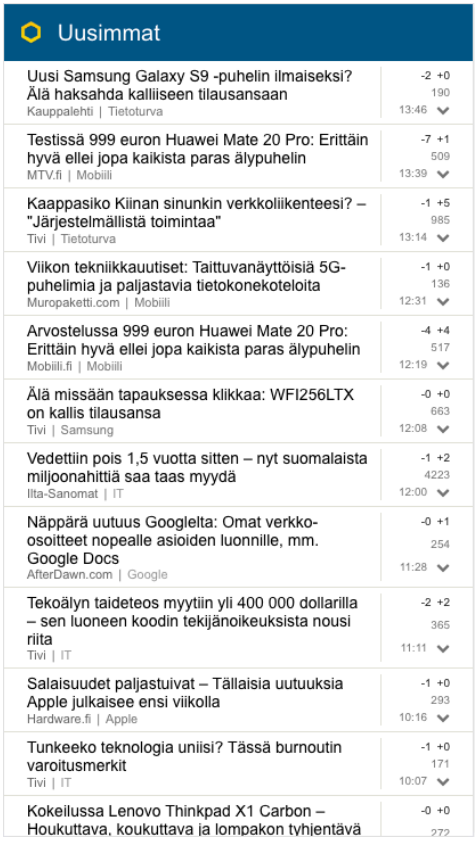
5.1 Aloitustilanne

Toimeksiantaja Aste Helsinki Oy:n Ville Sirkiä kysyi keväällä 2018 mahdollisuutta kehittää Ampparit.com sivuston pohjalta React Native sovellukset iOS- ja Android-alustoille. Tässä vaiheessa opinnäytetyön aihe ei muotoutunut, mutta tieto tarpeesta oli käsitelty. Vaikka sivuston pohjalta on jo olemassa natiivisti toteutetut mobiilisovellukset, Androidille Javalla ja iOS:lle Swiftillä, on niiden kehityksen vastuu ulkopuolisella toimijalla ja olisi myös hyvä saada yhtenäistettyä kehitystä sekä itse sivuston että mobiilisovellusten osalta. Koska Ampparit.com sivuston käyttöliittymä on kehitetty ReactJS kirjastolla, voitaisiin sivuston kehittäjäosaamista hyödyntää myös mobiilisovellusten puolella.

Työssäni kehitän Reactilla Amppareiden käyttöliittymää, joten tunnen sen toiminnallisuuden suhteellisen hyvin, mutta React Nativesta ja mobiilisovelluskehityksestä ylipäätään on vähemmän kokemusta. Olen aikaisemmin tehnyt React Nativella muutaman pienemmän sovelluksen, jolloin sen ymmärrys on kuitenkin kasvanut ja koska React Nativen ideologia periytyy Reactista, hyppy React kehityksestä React Nativeen ei ollut suuri. Päätös tehdä opinnäytetyö aiheenaan Ampparit.com React Nativella syntyi, kun pohdin eri aiheita ja muistin tarpeen ainakin selvitystyölle.

5.2 Sovelluksen tavoitenäkymä

Tavoitteena on yksinkertaisimmillaan listata uutisia Amppareiden Uusimmat sivulta niillä tiedoilla, mitä saadaan API:lta. Erillistä kategoriarakennetta ei toteuteta eikä sovellukseen vielä tässä vaiheessa lisätä muita Ampparit.com palvelusta löytyviä osioita, kuten Sää, TV ja Urheilu johtuen niiden erillisistä lisensseistä. Uusimmat sivu sovittiin käytettäväksi tässä projektissa projektin aloituskokouksessa, yhdessä Ampparit Oy:n kanssa, koska ideana on tarjota ajantasaisimmat otsikot. Kuvassa 6 esitellään mahdollista näkymää.



Kuva 6. Esimerkki tavoiteltavasta näkymästä sovellukselle

5.3 Sovelluksen aloitus

Aloittaessa projektia on loogista päättää, tehdäänkö sovellus kokonaan natiiviksi eri alustoille, jolloin asennetaan kehitysympäristö, kohdealustan perusteella vai tehdäänkö sovellus alustariippumattomasti. Kehitys Applen iOS-alustalle tapahtuu Xcode ympäristössä ja Android-alustalle Android Studio -ympäristössä. Mikäli toteutetaan alustariippumaton sovellus esimerkiksi juuri React Nativella, tarvitaan molemmat ympäristöt sekä Applen Mac-tietokone, johtuen Xcoden asentamisen rajoituksesta ainoastaan Mac-ympäristöön. Itse koodia voidaan kehittää millä tahansa tekstieditorilla, mutta sovelluksen lopullinen koonti tapahtuu kummankin alustan omalla ympäristöllä.

React Nativen dokumentaatiossa ”Getting Started” -osiossa on selkeästi kerrottu, miten kehityksessä pääsee alkuun nopeastikin. Ensimmäisenä pyydetään asentamaan Node.js, joka on Googlen V8 -JavaScript moottoriin perustuva serveripuolen sovellusalusta. (Tampereen Teknillinen Yliopisto, 2015, 3). Tämän jälkeen on valittavissa kahdesta eri tavasta aloittaa React Native -sovelluksen kehitys.

Yksinkertaisin tapa aloittaa React Native -projekti dokumentaation mukaan on käyttää ja asentaa CRNA-paketinhallintajärjestelmä NPM:n kautta, jolloin projektia ei tarvitse toteuttaa Xcoden taikka Android Studion avulla eikä niiden konfigurointia tarvitse tehdä. CRNA:n avulla aloitettu projekti pyörii Expo.io-alustan kautta, joka pyydetään asentamaan mobiililaitteelle, mutta on myös mahdollista käyttää iOS- tai Android-emulaattoria, mikäli ne on asennettu koneelle. (Facebook Inc, 2018a)

Huonona puolena CRNA:ssa on, että kehityksessä ei voida käyttää kustomoituja komponentteja, joita ei löydy React Nativen ohjelmointirajapinnasta tai Expo.io sovelluksesta. Kehitys voidaan kuitenkin aloittaa käyttämällä tätä menetelmää ja projekti voidaan myöhemmin ejektoida, jolloin edellä mainitut rajoitukset poistuvat. Omassa käytössä olen huomannut Expo.io sovelluksen kaatuvan iOS-laitteella suhteellisen usein, joten sen käyttö ei välttämättä ole mielekästä.

Toinen tapa aloittaa projekti on asentaa Node.js lisäksi Watchman-työkalu, joka seuraa muutoksia tiedostoissa ja React Native CLI sekä käyttää ”react-native init ProjektinNimi” -komentoa, jonka jälkeen ohjeistukset eriävät hieman käytettävän ohjelmointiympäristön ja kohdeympäristön mukaan. Tämä menettelytapa mahdollistaa omien komponenttien käytön, joita voidaan toteuttaa myös esimerkiksi Java- ja C-ohjelmointikielillä.

5.4 Sovelluksessa käytettävät React Nativen komponentit

Ampparit.com koostaa eri lähteiden uutiset yhteen näkymään, mutta itse uutiset luetaan niiden omista lähteistä eli eri verkkosivuilta. Sivustolla uutiset avautuvat uuteen välilehteen, mutta sovelluksissa ne avataan ohjelman sisäisellä selaimella näkymän vaihtuessa uutislistasta itse uutiseen. Uutislistaan takaisin navigointi tapahtuu ”Valmis” -painikkeen avulla.

Kirjoitettaessa Reactia voidaan käyttää tavallisia HTML-elementtejä (. <p>, , <a> jne.). React Nativen kanssa elementit korvataan alustakohtaisilla React komponenteilla. Koska kaikki käyttöliittymäelementit ovat React Nativen komponentteja, tulee ne tuoda erikseen jokaiseen työstettävään tiedostoon kuvan 7 mukaisesti.
(Eisenman, 2018, 9.)

```
import React, { Component } from 'react';
import {
  StyleSheet,
  Text,
  View,
  TouchableOpacity,
  Animated,
  Linking
} from 'react-native';
import { Icon, Divider } from 'react-native-elements'
```

Kuva 7. React Nativen komponenttien tuonti

Erilaisiin React Nativen komponentteihin on päädytty lähinnä koodiesimerkkien avulla, jotka on sittemmin muokattuina todettu soveltuviksi toteutettavan sovelluksen kohdalla. Jatkokehityksessä voidaan pohtia tarkemmin, minkälaiset eri komponentit soveltuvat parhaiten tämän kaltaiseen sovellukseen, mutta tähän versioon on pyritty valitsemaan sellaisia komponentteja, jotka löytyvät valmiiksi React Nativen kirjastosta. Sovelluksessa käytetään myös omia komponentteja, joiden avulla koodia saadaan pilkottua pienempiin osiin, jolloin saadaan toteutettua siistimpi kokonaisuus ja koodi on jatkokehityksen kannalta luettavampaa.

5.4.1 View

React Nativen yleisimpiä komponentteja on monialustaisena tuettu <View> ja jota voidaan käyttää joustavasti käyttöliittymässä. Kyseistä komponenttia käytetään kuvantamaan näkymän eri kohtia saman tyyppisesti, kuin HTML-kielessä käytetään elementtiä. (Eisenman, 2018, 8.)

View komponentti on elementti, joka tukee asettelua Flexboxilla, erilaisia tyylejä, kosketuksen hallintaa sekä erilaisia käyttöapuohjauksia. View kääntyy suoraan alustariippumattomasti sopivaksi natiiviksi näyttökomponentiksi. View:lle voidaan asettaa useita lapsielementtejä, mukaan lukien View itsessään. Kuvassa 8 esimerkki View:n käytöstä Ampparit React Native sovelluksessa:

```
export default class Logo extends Component {
  render() {
    return (
      <View style={styles.container}>
        <Image source={ require('../assets/ampparit-logo-white.png') } style={ styles.image} />
      </View>
    );
  }
}
```

Kuva 8. React Nativen View -komponentin käyttö Ampparit sovelluksen Logo -komponentissa

5.4.2 Text

Erilaisten tekstien piirtäminen on perustoimintoja lähes missä tahansa sovelluksessa. HTML -kielessä tekstejä määritellään eri tavoin, mutta yleisimmin <p> -elementillä, jonka lisäksi tekstin sisällä voidaan määritellä ominaisuuksia vaikka - ja -elementeillä. React Nativessa erilaiset tekstipaikat esitellään <Text> -komponentilla, jossa ei voida suoraan käyttää samanlaisia muokkauksia, kuin HTML -kielessä vaan ne voidaan asettaa tyyleihin erikseen. Mikäli on tarve käyttää useita erilaisia muokkauksia, on kannattavaa tehdä niille omat tyyliluokkansa, joihin tarpeen tullen viitataan.
(Eisenman, 2018, 40. – 41.)

Ampparit-sovelluksessa <Text> komponenttia käytetään mm. uutisten otikoissa sekä lähteiden näyttämisessä ja aikaleimassa. Ampparit-sovellukseen data tulee JSON-formaatissa, joten lähes kaikki näytettävät asiat ovat tekstimuotoisia.

5.4.3 Animated

React Nativen Animated-kirjasto on suunniteltu erilaisten animaatioiden näyttämiseksi sovelluksessa helposti. Animated keskittyy syötteisiin (input) ja ulostuloihin (output), joita voidaan konfiguroida yksinkertaisilla start- ja stop-metodeilla aikapohjaisten animaatioiden toteutuksien hallintaan. Animated-kirjastoa voidaan käyttää esimerkiksi View -komponentin kanssa, jolloin koko View:n esittämä näkymä saadaan animoinnin piiriin.
(Facebook Inc, 2018e)

Esitellessäni toteutettavaa Ampparit-sovellusta Amppareiden tuoteomistajalle, saimme idean, että uutislistauksen uutiset voisivat olla käännettäviä kortteja, joiden takaa löytyisi lisää infoa uutisen klikeistä ja äänistä. Idean taustalla oli nykyisten Ampparit mobiilisovellusten näkymän käyttö, joka toisinaan on sekavan oloinen kaiken informaation ollessa samassa näkymässä. Tutkin soveltuvaa tapaa kääntää kortti 180 astetta sivuttaissuunnassa ja löysin Jason Brownin toteutuksen käännettävästä kortista, jota hyödynsin toteutettavan sovelluksen uutiskorttien käännössä. Toteutuksessa on käytetty React Nativen Animated -kirjastoa, jolloin kortin käännöstä saadaan sulava ja liikkeestä yhdenmukainen.

5.4.4 TouchableOpacity

Mobiiliaplikaatioita käytetään nykyään pääsääntöisesti kosketuksilla, joihin sisältyy nappien painamista, listojen kelausta sekä karttojen zoomausta. Monesti kosketettavia elementtejä kuvataan napeilla, mutta aina niitä ei ole mielekästä käyttää, joten tätä varten React Nativessa on erilaisia kosketuskomponentteja. (Facebook Inc, 2018d)

Ampparit-sovelluksessa on käytetty TouchableOpacity -komponenttia, joka toimii sekä iOS- että Android-alustalla. Kyseistä komponenttia on käytetty myös siksi, koska uutislistauksessa näytettävät uutiset ovat käännettäviä kortteja, joiden taustapuolella on lisäinfoa kyseisestä uutisesta, kuten esimerkiksi päivämäärä, uutisen Amppareiden kautta tehdyt klikit sekä annetut äänet.

5.4.5 FlatList

FlatList -komponentilla voidaan helposti esittää dataa, kuten esimerkiksi uutisia, listamuotoisesti. Käytettäessä FlatList:a tulee sille antaa kaksi attribuuttia: ”data”, joka on nimensä mukaisesti se tieto, jota halutaan esittää ja ”renderItem” -funktio, joka palauttaa komponentin perustuen dataan. Palautettavan datan tulisi olla lista, jonka jokaisella elementillä on yksilöllinen avain attribuutti. (Eisenman, 2018, 56.)

Ampparit mobiilisovelluksessa FlatList -komponentille annetaan datana uutissyöte, joka saadaan Amppareiden API:lta ja renderItem palauttaa Item -komponentin, joka määrittelee uutisen näkymisen listauksessa.

5.4.6 WebView

React Nativesta löytyy sisäänrakennettu komponentti nimeltään WebView, jonka avulla kuvannetaan verkkosivujen sisältöä sovelluksessa natiivisti. WebView poikkeaa normaalista selaimesta esimerkiksi siten, että siitä puuttuu osoiterivi sekä hallintapainikkeet, mutta toki ne voisi erikseen rakentaa näkymän ympärille.

Navigointi React Nativella toteutettavassa sovelluksessa on varsin kevyttä, lähinnä uutislistauksen ja uutisnäkymän välillä, mutta tuotannossa olevissa natiivisovelluksissa on useampi erilainen sivu, kuten esimerkiksi Sää ja TV-opas, jotka otetaan huomioon myös toteutettavassa sovelluksessa. React Nativen dokumentaation mukaan yksinkertaisin tapa navigoida eri näkymien välillä on käyttää yhteisökehitteistä React Navigation –komponenttia. (Facebook Inc, 2018c)

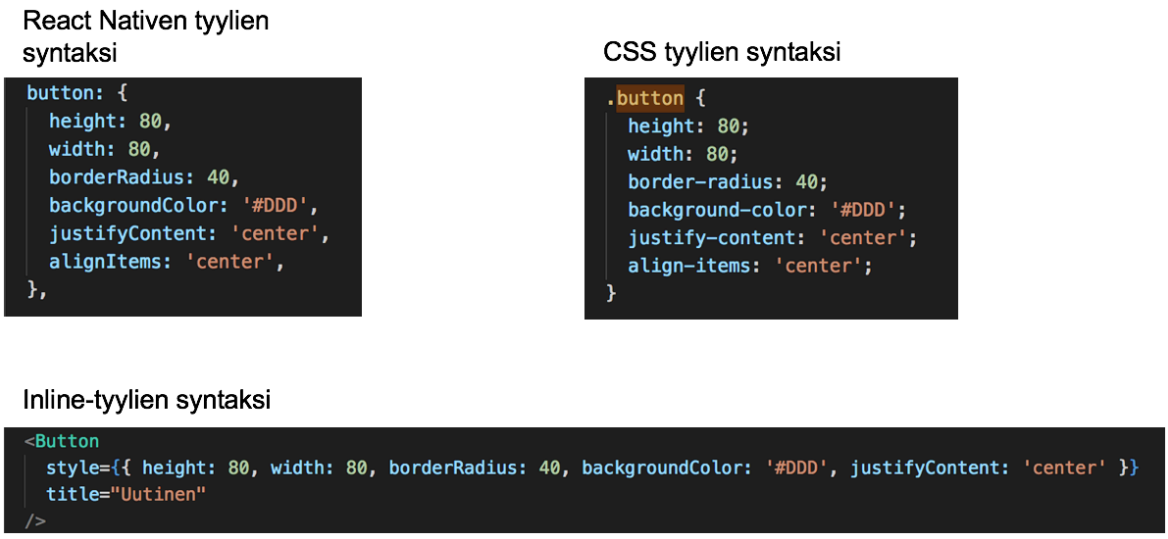
5.5 React Nativen ja sovelluksen tyylit

Ampparit.com sivustolla on käytetty useita tyyliltään erilaisia elementtejä, mutta samoja elementtejä kuitenkin hyödynnetään useammassa näkymässä. Elementeille on määritelty perustyylit, joiden lisäksi käyttäjän valittavana on muutama erilainen teema, jonka mukaan lähinnä elementtien värimaailma muuttuu. Toteutettavassa mobiilisovelluksessa ei kuiten-

kaan toteuteta eri teemoja, mutta pyritään mahdollistamaan niiden käyttö myöhemmin. Ulkoasussa yhteneväiset elementit, kuten esimerkiksi napit, tehdään käyttäen samaa tyyliä hyödyntäen, jotta lopputuloksesta saataisiin yhdenmukainen jokaisessa näkymässä.

React Native elementtejä voidaan tyylitellä muutamalla eri tavalla; tyylit voidaan sijoittaa yksittäisiin elementteihin, komponentin yhteyteen samaan JavaScript-tiedostoon tai täysin erilliseen tyylitiedostoon. Toteutettavassa sovelluksessa olen päätynyt määrittelemään tyylit komponentin yhteyteen, koska ainakin itse koen alkuvaiheessa sen selkeämpänä. Mikäli sovellukseen myöhemmin lisättäisiin erilaisia käyttöliittymäelementtejä, jotka toistuvat, voisi olla järkevää tehdä niille yhteen tyylitiedostoon erilliset määritykset.

React Nativessa tyyleihin ei käytetä mitään erityistä kieltä, kuten CSS, tyylien määritykseen, mutta syntaksi on hyvin samankaltainen CSS:n kanssa. Suurimpana erona on, miten tyylit esitellään; React Nativessa tyylien nimet ovat yhteen kirjoitettuna CamelCase tapaan sen sijaan, että ne kirjoitettaisiin CSS:n tapaan esimerkiksi väliviivalla kuten kuvasta 9 voidaan todeta. Myös Inline-tyylejä on mahdollista käyttää, jotka määritellään elementteihin style-attribuuttia käyttämällä. Style-attribuuttia tulisi kuitenkin käyttää harkiten, koska silloin menetetään tyylitiedostojen eduista; samankaltaisten elementtien tyylien määritykset tulisi tehdä aina erikseen, jolloin elementtien tyyleistä saattaa tulla erilaisia ja muokattavuus kärsii. Samalla myös koodin luettavuus kärsii, jos rivien välissä on paljon tyylimäärittelyjä.

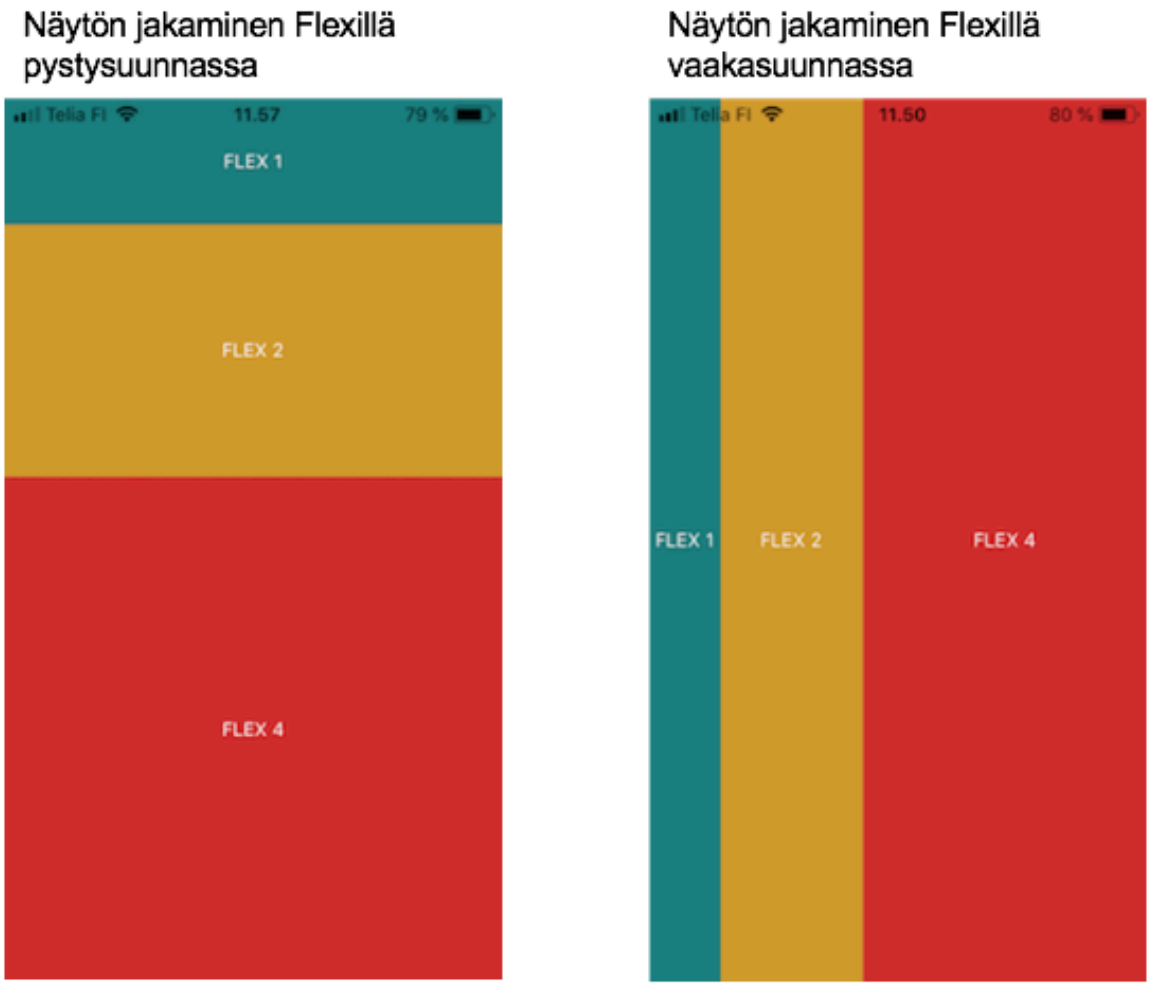


Kuva 9. React Nativen, CSS:n ja Inline-tyylien syntaksit

5.5.1 Skaalaus

React Nativessa sovelluksen sisällön näyttäminen jaetaan esimerkiksi Flexboxin avulla, jolloin eri elementit toimivat näytön koosta huolimatta, eikä erillisiä media queryjä tarvita, kuten Web-kehityksessä on yleensä tapana käyttää. Flexbox ei ole ainoa tapa tyylittää ja jakaa näyttö, mutta se on erittäin kätevä tapa tehdä responsiivisia näkymiä useammalle erilaiselle näytölle.

Flexbox toimii pääsääntöisesti samalla tavalla, kuin CSS yhteydessä muutamia poikkeuksia lukuun ottamatta; oletuksena flexDirection parametri palauttaa vaakasuunnan sijaan pystysuunnan ja flex parametri hyväksyy vain yhden arvon. (Facebook Inc, 2018b) Elementin koko määräytyy flex arvon mukaan, mikäli sille ei ole erikseen määritelty kokoa width -tai height-arvojen mukaan. Kuvassa 10 esimerkki näytön jakamisesta Flexboxilla.



Kuva 10. Flexboxilla skaalaus

Flexboxista löytyy myös muutama muu tapa käsitellä elementtien sijaintia näytöllä. Kun tarpeelliset elementit on piirretty näytölle, voidaan niiden sijaintia käsitellä esimerkiksi justifyContent arvolla, jolla voidaan kohdistaa flexin alaiset sisällöt vaakasuunnassa silloin,

kun ne eivät käytä kaikkea käytettävissä olevaa tilaa ja vastaavasti alignItems arvolla voidaan kohdistaa elementit pystysuunnassa.

Toteutettavassa sovelluksessa jaetaan näyttöä lähinnä otsikkopalkin sekä uutislistausnäkymän kesken, mutta tulevaisuudessa sovellukseen tullaan lisäämään enemmän navigointia esimerkiksi Sää- TV- ja Urheilusivuille, jolloin niissä Flexboxin käyttö varmasti korostuu.

5.6 Sovelluksen API integraatio

Sovelluksen toteuttamista aloittaessa, oli tarkoitus tukeutua Ampparit.com sivuston lähdekoodissa toteutettuun tapaan, jossa dataa haetaan Axios-pyynnöillä, mutta React Native tukee ainoastaa Fetch-API:a, joka poikkeaa hieman Axioksesta. Fetchiä käytettäessä kuvan 7 tapaan, annetaan haettavan API:n osoitteen URL muuttujana fetchille, jonka jälkeen se asetetaan json():lle jolloin saadaan data palautuksena konsoliin. (Arnold, 2017)

Esimerkki Fetch-API:n käytöstä:

```
const url = 'https://api.verkkosivu.com/v2/haettava/asia/'

fetch(url).then(response => response.json()).then(data => con-
sole.log(data));
```

Axiosta käytettäessä se ensin asennetaan paketinhallinasta (NPM), jonka jälkeen pyyntö voidaan suorittaa kuvan 8 mukaisesti.

Esimerkki Axioksen käytöstä:

```
const url = 'https://api.verkkosivu.com/v2/haettava/asia/'

axios.get(url).then(response => console.log(response));
```

Yhteyttä otettaessa Amppareiden API on autentikaation takana, jonka avulla pyyntö todennetaan. Pyynnön mukana voidaan laittaa myös erilaisia parametreja, jonka perusteella API palauttaa tietoa tarkemmin määritellysti. Tämän jälkeen data palautuu JSON muotoisena, jolloin sitä voidaan käsitellä halutulla tavalla. Toteutettavan sovelluksen ainoana pyyntönä on lista uusimmista uutisista. Vetämällä listaa alaspäin ja vapauttamalla sovellus tekee uuden pyynnön listasta.

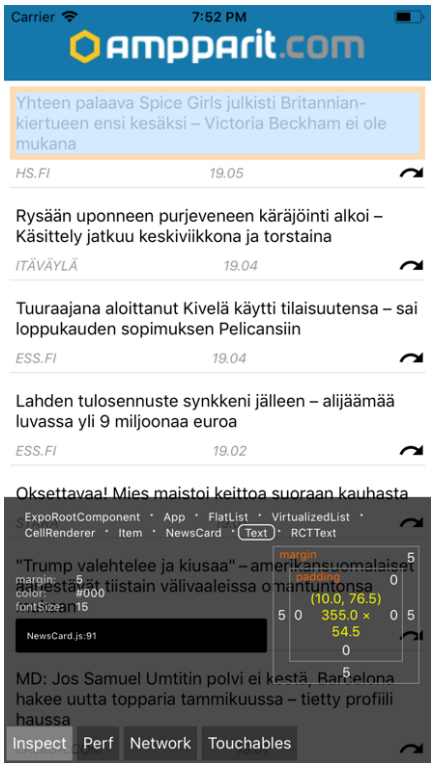
5.7 Kehityksen työkalut

React Nativella sovellusta kehitettäessä tukena on monia erilaisia työkaluja, joilla voidaan seurata sovelluksen suorituskykyä, lukea sovelluksen koodiin tehtyjä lokituksia, tutkia elementtejä sekä saadaan varoitus- ja virheilmoituksia.

Koska suorituskyvyn ylläpitäminen on yksi keskeisimmistä asioista React Nativella tuotetuissa sovelluksissa, on sen seurantaa varten olemassa myös kuvan 11 mukainen ohjelman ajon aikainen suorituskykymonitori. Monitorilla voidaan seurata sovelluksen käyttämää muistia sekä ruudunpäivitysnopeutta, joiden avulla voidaan saada kiinni ne kohdat sovelluksen käytössä, jotka aiheuttavat hitautta tai muuta viivettä.

Kuva 11. React Nativen suorituskykymonitori

Inspector-työkalun avulla voidaan tutkia sovelluksen elementtejä samaan tapaan kuin Web-kehityksessä, jolloin saadaan tietoa eri elementtien sijoittelusta sekä esimerkiksi kosketukseen vastaavista elementeistä. Kuvan 12 mukaan Inspector-työkalu näyttää myös elementtien väliset suhteet, kuten monissa selainpohjaisissa kehittäjätyökaluissa näytetään.



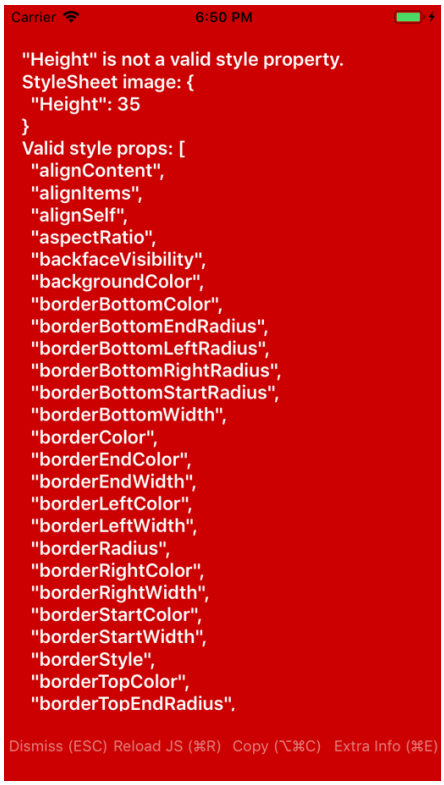
Kuva 12. React Nativen Inspector-työkalu

5.7.1 Virhe- ja varoitusilmoitukset

Virheilmoitukset tulevat React Nativessa joko simulaattorin tai puhelimen näytölle, riippuen kumpaa käytetään. Virheilmoitukset tulevat selkeästi punaisella taustavärillä ja itse virheellä, josta yleensä nähdään, missä virheellinen koodinosa sijaitsee. Samalla tulee myös ehdotus virheen korjaamiseksi. Virheilmoitukset estävät ohjelman piirtämisen kokonaan, jolloin virhe tulee ratkaista ensin kehitystyön jatkumiseksi. Mikäli virhe halutaan laukaista automaattisesti tietyssä osassa koodissa, voidaan käyttää console.error() -komentoa.

Virheilmoituksia saattaa esiintyä kehitystyön yhteydessä tiuhaan tahtiin, mikäli React Native sovellus ladataan uudelleen, kun muutoksia koodissa havaitaan. Automaattinen uudelleenlataus voidaan ottaa käyttöön valitsemalla "Enable Live Reload" kehittäjävalikosta. (Facebook Inc, 2018g)

Kuvassa 13 on virheilmoitus, jossa kerrotaan vääränlaisesta tyyliominaisuudesta ”Height” ja missä se sijaitsee. Virheilmoitus ei kuitenkaan osaa suoraan sanoa, mikä on oikea muoto, mutta antaa listan valideista tyyliattribuuteista, joista saadaan ainakin vihjettä oikeasta muodosta.



Kuva 13. React Nativen virheilmoitus

Varoitusilmoitukset (kuva 14) tulevat keltaisella pohjavärillä ja ne eivät estä sovelluksen piirtämistä, mutta sovellus ei välttämättä toimi oikein. Varoitusilmoituksia voidaan virheilmoitusten tapaan laukaista halutussa koodin osassa console.warn() -komennolla.

Kuva 14. React Nativen varoitusilmoitus

5.8 Haasteet

Vaikka kehitystyö Reactin ideologian mukaan on sinänsä tuttua, on React Nativella kehittäminen hieman erilaista. Myös sovelluksen aloittaminen ”tyhjästä” vaati ensin hieman suunnittelua, koska tarkoituksena on rakentaa runko tulevaa varten. Vaikka Ampparit.com sivuston lähdekoodi on hyödynnettävissä, ei siitä pysty pystynyt hyödyntämään ehkä niitä haasteellisimpia asioita omalla kohdalla.

Haastellisinta omalla kohdalla oli ymmärtää, miten React Native sovellukseen saadaan dataa ulkopuoliselta API:lta, jonka käyttämiseen tarvitaan erillinen API-avain. Ymmärrettyäni toteutustavan lähti kehitystyö etenemään hyvällä tahdilla.

Web-kehityksessä käytetyt virheenkorjaustavat, kuten konsoliin lokittaminen, oli alkuun hukassa. Pienellä tutkimisella asiaan tuli kuitenkin selvyys; ohjelman sisäisessä kehittäjä menussa oli mahdollisuus suorittaa virheenkorjausta etänä tietokoneen selaimella, jolloin lokitusviestitkin tulivat näytölle.

6 Pohdinta

Opinnäytetyön aiheeksi valikoitui tutustua React Nativeen ja tuottaa sillä sovellus Ampparit.com sivustosta, jolloin saataisiin selkeytettyä, miten mobiilisovellusten kehitystä sivuston rinnalla voitaisiin tehostaa. React Nativen käyttöä oli myös pohdittu työyhteisössä, etenkin silloin, kun ollaan mietitty uusia ominaisuuksia sivustolle, jotka haluttaisiin myös mahdollisesti mobiilisovelluksiin. Hybridisovellusalustat kehittyvät jatkuvasti, jolloin on hyvä aika alkaa pohtimaan niiden hyödyntämistä sovelluskehityksessä, mutta myös sitä, minkälaisissa palveluissa on hyvä olla erikseen sivusto sekä mobiilisovellukset eri alustoille.

6.1 Tavoitetta vasten tehty oppiminen

Projektin aloitus vaikutti helpolta, koska palkkatyössäni kehitän Ampparit.com sivustoa Reactilla, jolloin syntaksi on tuttu. Tämän lisäksi olen aikaisemmin tehnyt omissa projekteissani pieniä mobiilisovelluksia, jolloin olen tutustunut React Nativeen tarkemmin. Itse teoriapohja olikin helppoa kirjoittaa, mutta aloittaessani sovelluksen toteuttamista oli opeteltava uusia asioita, kuten miten dataa loppujen lopuksi haetaan API:sta sekä minkälaista dataa sieltä kaiken kaikkiaan palautuu.

Vaikka tukena oli varsinaisen sivuston koodikanta, sen hyödyntäminen ei ainakaan aluksi ollut helppoa. Opettavaisin osa onkin ollut ymmärtää miten asiat eroavat Reactissa ja React Nativessa, mutta tutustuessani erilaisiin sovelluksen toteuttamisen vaihtoehtoihin React Nativen rinnalla, ymmärrys muista alustoista syveni kuten myös erilaisten valintojen merkitys sovelluskehityksessä.

Oppimisen tukena työn aikana olivat kollegani, jotka tarvittaessa selvensivät niitä Ampparei den arkkitehtuurin osia, joista itselläni ei ollut selvää kuvaa. Myös käyttöliittymän suunnittelun oppiminen on ollut antoisaa ja jotain sellaista, mitä pystyn hyödyntämään myös varsinaisessa työssäni.

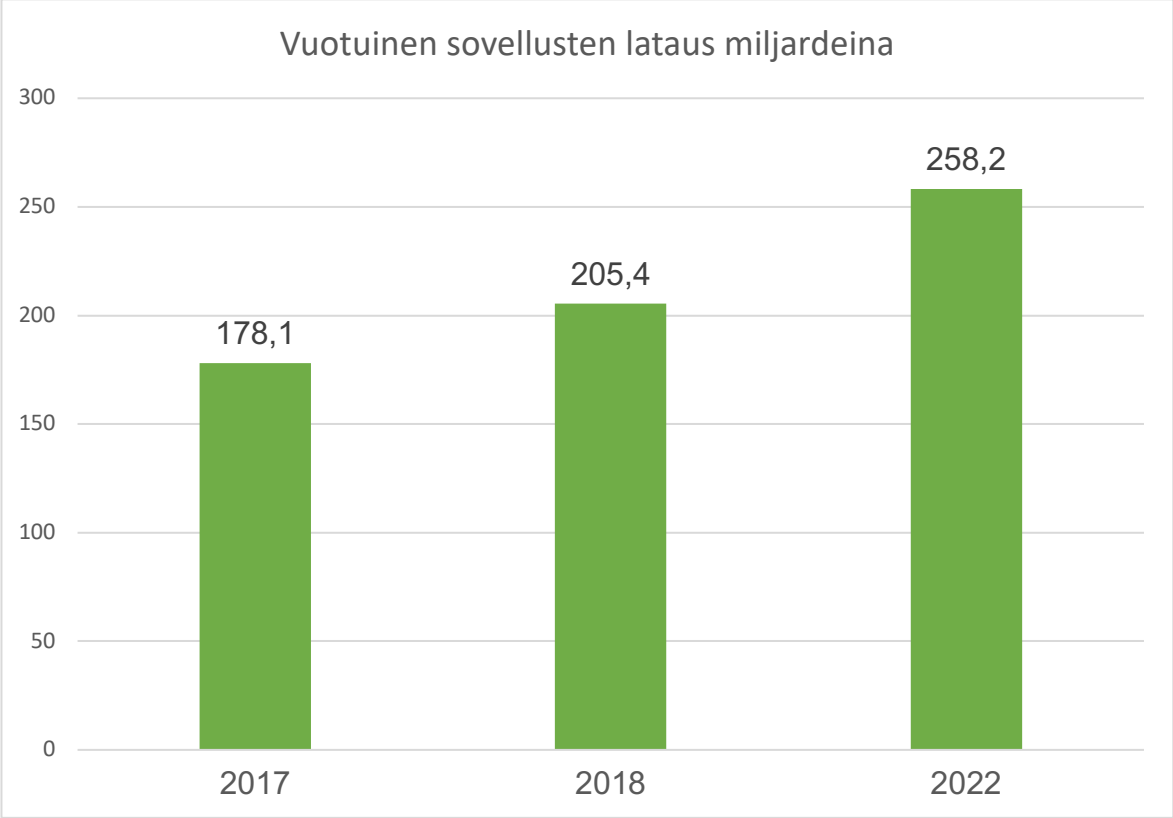
6.2 Mobiilin käyttöliittymän merkitys

Mobiililaitteista on pikkuhiljaa tullut se ensisijainen tapa hankkia tietoa internetin välityksellä. Laitteet kulkevat helposti mukana ja varsinkin Suomessa mobiiliverkko kattaa laajan alueen koko maasta, jolloin käyttö on myös helppoa ja niiden nopeutuessa kiinteät liittymät saatavat vähentyä. Tablet-laitteet korvaavat jo osittain tietokoneen niiden ollessa aina päällä, pienempikokoisia ja kevyempiä, jolloin tietoa saadaan nopeasti isommalta näytöltä

menettämättä välttämättä sivun käyttökokemusta, mutta taskuun mahtuvan puhelimen näytöllä käyttöliittymän hyvän suunnittelun merkitys kasvaa.

Mobiilisovellusten käyttömäärät kasvavat jatkuvasti ja seuraavien vuosien aikana kasvua odotetaankin tapahtuvan jopa 45%, kuten Taulukko 2 voidaan todeta. Markkinoiden kehittyessä sovellusten laatuvaatimukset kasvavat, joka on hyvä ottaa huomioon sovelluskehityksessä sekä käyttäjäkokemuksen suunnittelussa. (Cheney S. & Thompson, E. 2018)

Taulukko 2. Mobiilisovellusten latausten ennustemäärä 2017 – 2022 (Cheney S. & Thompson, E. 2018)



6.3 Jatkokehitys

Toteutetussa mobiilisovelluksessa on päädytty käyttämään mahdollisimman paljon React Nativen verkkodokumentaatioon pohjautuvia ratkaisuja. Monet erikseen lisättävät paketit sovellukseen saatavat kadottaa tukensa ajan myötä, joten niitä on pyritty välttämään. Virallista dokumentaatiota päivitetään React Nativen kehittäjien toimesta, joten siellä on myös kaikkein ajantasaisin tieto optimaalisesti toimivista ratkaisuista. Ampparit.com on sivustona kuitenkin monia muitakin palveluita sisältävä kokonaisuus, joita tässä työssä ei erikseen eritellä. Tämän myötä erilaisten ulkopuolisten pakettien tai komponenttien käyttö

on jatkokehityksessä perusteltua, kuten varsinaisen sivuston puolellakin on käytetty. Toteutetussa sovelluksessa ei välttämättä ole sovelluksen lopullisen version kannalta optimaalisimpia ratkaisuja, mutta se palvelee tämän dokumentaation ymmärtämistä.

Reactin ja React Nativen ollessa rakenteeltaan lähellä toisiaan, voitaisiin pohtia myös, olisiko mahdollista käyttää täysin samaa koodikantaa ainakin niiltä osin, jossa elementin piirtämistä ei käsitellä. Huomioitavaa olisivat siis itse elementin piirtämiseen vaikuttavat asiat, kuten div-elementtien muokkaaminen View-komponenteiksi. Näin ollen saataisiin kehitystyöstä kahden erilaisen alustan välillä vieläkin sujuvampaa, kun sama muutos voitaisiin toteuttaa yhtäaikaisesti sekä sivustolle että mobiilisovellukseen.

6.3.1 Ajankäyttö

Toteutetun sovelluksen nykymuotoon saattamiseen ja kehittämiseen aikaa on mennyt n. 60 tuntia ottaen huomioon selvitystyön tekniikan osalta sekä lähtökohdat; valmis käyttöliittymämalli selkeytti tavoitetta, mutta Reactin ja React Nativen pienet eroavaisuudet tuottivat toisinaan enemmän työtä selvitykseen, miten jokin asia toteutetaan. Hyvä suunnittelu varsinkin käyttöliittymän tyylien osalta on merkittävässä asemassa ajankäytön optimoimiseksi.

Koska sovelluksen käyttöliittymälle löytyy malli osittain Ampparit.com-sivustosta, on eri ominaisuuksien lisääminen lähtökohdallisesti nopeaa, mutta sovelluksen ollessa kyseessä ja luvussa 6.1 olevan maininnan mukaan laatuvaatimuksista, käyttöliittymäsuunnitteluun kannattaa panostaa vielä erikseen, jotta kokemuksesta saadaan sulava sekä sovelluksesta miellyttävä käyttää. Myös pieni massasta erottuvaisuus ei liene pahitteeksi.

Ampparit.com-sivuston kehityksessä käytetään keskimäärin kahden viikon kehitysjaksoja eli sprinttejä. Sovelluksen kehittämiseen nykymuotoonsa on siis kulunut n. 75% yhdestä sprintistä, mutta toki on otettava huomioon, että sovellus ei ole vielä julkaisukunnossa. Yhden uuden näkymäkokonaisuuden eli esimerkiksi Sää- tai TV-sivun luontiin voisi mennä 1-3 sprinttiä / henkilö pelkän kehityksen osalta. Toki osaamisen karttuessa kehitystyökin nopeutuu. Tämän lisäksi tulee ottaa huomioon vielä käyttöliittymäsuunnitteluun kuluva aika erikseen, jonka voisin arvioida olevan puolesta sprintistä yhteen sprinttiin / henkilö, riippuen käytetäänkö sivustoa tai nykyisiä sovelluksia pohjalla vai tehdäänkö jotain täysin uutta.

6.3.2 Mainosintegraatio

Ampparit.com:in ollessa mainosrahoitteinen palvelu, on mainosintegraatiokin sovellukseen edessä, mikäli siihen ei sovelleta muuta rahoitusmenetelmää. Ampparit palveluna käyttää Googlen Ad Manager -järjestelmää mainoksiin. Koska Amppareiden sivuston puolella on jo käytetty Reactilla rakennettuja ratkaisuja, niiden migratoiminen React Nativella toimivaksi voisi olla suhteellisen vaivatonta luokkarakenteiden ollessa samanlaisia. React Nativelle löytyy myös muutama erilainen NPM-paketti, joiden avulla saadaan nopeasti toteutettua mainospaikat mobiilisovellukseen, mutta mainoskoot näyttäisivät osittain poikkeavan siitä, mitä esimerkiksi Amppareissa on käytetty.

6.3.3 Relevantti sisältö

Opinnäytetyön käynnistämisen yhteydessä tuli ajatusta toimeksiantajalta sekä asiakkaan edustajalta, voitaisiinko uutissisältöä näyttää sen mukaan, mikä on ollut suosittua käyttäjien keskuudessa. Sovellus laskisi suosituimmat uutiset sekä niiden saamat käyttäjien antamat äänet, jonka perusteella lista muodostuisi. Tästä oma ajatukseni oli vielä miettiä sisältö käyttäjän mieltymysten mukaan tarkemmin eli osittain kohdennetusti. Omassa käytössäni korostuisivat esimerkiksi IT-alan ja tekniikan uutiset ja jollain toisella esimerkiksi urheiluun liittyvät uutiset. Näin saataisiin myös käyttäjän näkökulmasta relevanttia sisältöä ja mielenkiinto sovelluksen käyttöön varmasti kasvaisi. Monia samankaltaisia sovelluksia on olemassa, esimerkiksi Flipboard, mutta näissä lähteet ovat ulkomaisia ja suomenkielisissä vastaavissa sovelluksissa näytetään yhden lähteen uutisia.

6.4 Yhteenveto

Opinnäytetyön tavoitteena oli tutkia, kuinka Ampparit.com-sivustosta saataisiin React Nativella toteutettu mobiilisovellus. Työssä vertaillaan vaihtoehtoisia teknologioita, mutta myös käydään läpi, mitä hyötyjä React Nativesta on teknologiana, kun samanaikaisesti kehitetään ReactJS-teknologiaan perustuvaa sivua. Yksi mahdollisista eduista voisi olla käyttää samaa kehittäjäosaamista sekä sivuston että mobiilisovelluksen puolella, jolloin muutokset olisivat joustavampia.

Tavoitteena oli myös dokumentaation tueksi toteuttaa React Nativella kevennetty versio Ampparit.com-palvelusta. Sovellus itsessään ei ole valmis julkaistavaksi, mutta ainakin se voisi toimia runkona jatkokehitystä ajatellen. Sovelluksen toteutuksen yhteydessä erilaiset teknologiset ratkaisut ovat selkeytyneet ja ovat toimineet myös oppimisen tukena. Mahdol-

lisuus hieman leikitellä käyttöliittymällä tuotti myös idean käyttää uutislistauksessa kortteja, joiden taakse voidaan piilottaa lisää informaatiota, mutta samalla saadaan väljyyttä listaukseen.

Oma ymmärrys React Nativesta teknologiana, mutta myös ReactJS:stä on kasvanut suuresti ja mielenkiinto hybridisovelluksien toteuttamiseen on syventynyt. Erityisesti käyttöliittymäsuunnittelun merkitys on kasvanut omalla kohdalla ja siihen kiinnitänkin enemmän huomiota jatkossa. Tulevaisuudessa aion jatkaa mobiilisovellusten kehitystä ja opettelemista ainakin harrastuksena.

7 Lähteet

Aggarwal, A. 2017. Ionic vs React Native. Luettavissa: https://medium.com/@ankushaggarwal/ionic-vs-react-native-3eb62f8943f8. Luettu: 19.11.2018

Arnold, J. 2017. Fetch vs. Axios.js for making http requests. Luettavissa: https://medium.com/@thejasonfile/fetch-vs-axios-js-for-making-http-requests-2b261cdd3af5. Luettu: 27.10.2018

Bohdan. 2018. React Native vs NativeScript: The Battle Of Mobile Development Tools. Luettavissa: https://jelvix.com/blog/react-native-vs-nativescript. Luettu 21.10.2018

Cheney S. & Thompson, E. 2018. The 2017-2022 App Economy Forecast: 6 Billion Devices, $157 Billion in Spend & More. Luettavissa: https://www.appannie.com/en/insights/market-data/app-annie-2017-2022-forecast/. Luettu: 18.11.2018

De Baets, P. Lang, A. Sagnes, F. Zagallo, T. 2016. Dive into React Native performance. Luettavissa: https://code.fb.com/android/dive-into-react-native-performance/. Luettu: 20.11.2018

Eisenman, B. 2018. Learning React Native, 2nd Edition Building Native Mobile Apps with JavaScript. O'Reilly Media. California.

Facebook Inc, 2018a. Getting Started, React Native verkkodokumentaatio. Luettavissa: https://facebook.github.io/react-native/docs/getting-started.html. Luettu 30.9.2018

Facebook Inc, 2018b. Layout with Flexbox, React Native verkkodokumentaatio. Facebook Inc. Luettavissa: https://facebook.github.io/react-native/docs/flexbox. Luettu: 22.10.2018

Facebook Inc, 2018c. Navigating Between Screens, React Native verkkodokumentaatio. Luettavissa: https://facebook.github.io/react-native/docs/navigation. Luettu 30.9.2018

Facebook Inc, 2018d. Handling Touches, React Native verkkodokumentaatio. Luettavissa: https://facebook.github.io/react-native/docs/handling-touches. Luettu 4.11.2018

Facebook Inc, 2018e. Animated, React Native verkkodokumentaatio. Luettavissa: https://facebook.github.io/react-native/docs/animated. Luettu 4.11.2018

Facebook Inc, 2018f. Performance, React Native verkkodokumentaatio. Luettavissa: https://facebook.github.io/react-native/docs/performance. Luettu 4.11.2018

Facebook Inc, 2018g. Debugging, React Native verkkodokumentaatio. Luettavissa: https://facebook.github.io/react-native/docs/debugging. Luettu 19.11.2018

Facebook Inc, 2018h. Learn the Basics. React Native verkkodokumentaatio. Luettavissa: https://facebook.github.io/react-native/docs/tutorial.html. Luettu: 21.10.2018

Flutter.io, 2018a. FAQ, verkkodokumentaatio, Luettavissa: https://flutter.io/faq/#what-is-flutter. Luettu: 21.10.2018

Flutter.io, 2018b. Write your first Flutter app, part1, verkkodokumentaatio. Luettavissa: https://flutter.io/docs/get-started/codelab. Luettu: 21.10.2018

Gartner, 2018. Worldwide Smartphone Sales to End Users by Operating System in 2017 (Thousands of Units). Luettavissa: https://www.gartner.com/en/newsroom/press-releases/2018-02-22-gartner-says-worldwide-sales-of-smartphones-recorded-first-ever-decline-during-the-fourth-quarter-of-2017. Luettu 31.10.2018

Haldar, M. 2017. ReactJs component lifecycle methods—A deep dive. Luettavissa: https://hackernoon.com/reactjs-component-lifecycle-methods-a-deep-dive-38275d9d13c0. Luettu: 2.11.2018

Isohanni, K. 2018. Ampparit.com palvelun käyttäjämäärät. Asiantuntijahaastattelu.

Kinnunen, M. 2018. Ampparit.com uutiskeräimien toiminta ja ylläpitokäyttöliittymä. Asiantuntijahaastattelu.

Korhonen, S. 2018. Ampparit.com ohjelmointirajapinnan toiminta. Asiantuntijahaastattelu.

Mangin, A. 2016. What are the main differences between ReactJS and React-Native? Luettavissa: https://medium.com/@alexmngn/from-reactjs-to-react-native-what-are-the-main-differences-between-both-d6e8e88ebf24. Luettu 16.10.2018

NativeScript, 2018. Verkkodokumentaatio. Luettavissa: https://www.nativescript.org/. Luettu: 21.10.2018

Ristola, T. 2016. Kuinka toteutustapa vaikuttaa mobiilisovelluksen käytön sujuvuuteen? Luettavissa: https://gofore.com/toteutustapa-vaikuttaa-mobiilisovelluksen-kayton-sujuvuuteen/. Luettu 17.10.2018

Sznajder, D. 2017. ReactJS and React Native similarities and differences Luettavissa: https://medium.com/selleo/reactjs-and-react-native-similarities-and-differences-5a922032c4fb. Luettu: 22.9.2018

Tampereen Teknillinen Yliopisto. 2015. Node.js Luettavissa: http://www.cs.tut.fi/~seitti/2015/kalvot/nodejs/#1. Luettu: 30.9.2018

Tolvanen, P. 2016. Kolme syytä olla panostamatta mobiilisovelluksiin. Luettavissa: https://web-ostajanopas.fi/2016/05/30/kolme-syyta-olla-panostamatta-mobiilisovelluksiin/. Luettu: 19.10.2018

Wikipedia, 2018a. CamelCase. Luettavissa: https://fi.wikipedia.org/wiki/CamelCase. Luettu: 7.10.2018

Wikipedia, 2018b. Dart. Luettavissa: https://en.wikipedia.org/wiki/Dart_(programming_language). Luettu: 21.10.2018

Wikipedia, 2018c. Ionic. Luettavissa: https://en.wikipedia.org/wiki/Ionic_(mobile_app_framework). Luettu 19.11.2018

Wikipedia, 2018d. Xamarin. Luettavissa: https://en.wikipedia.org/wiki/Xamarin. Luettu: 21.10.2018

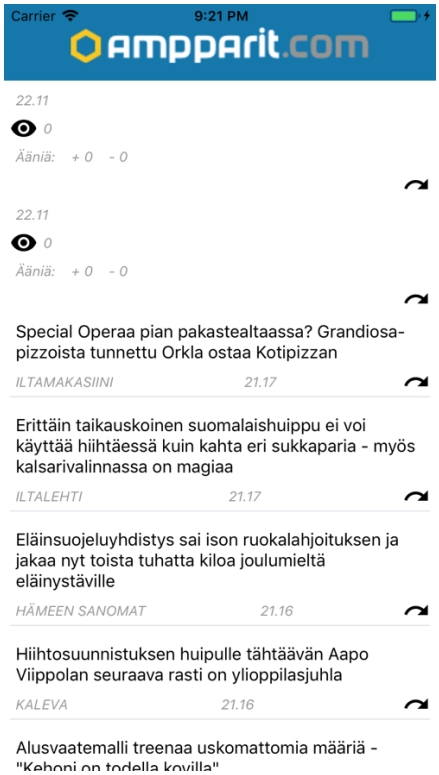
Liitteet

Liite 1. Lyhenteet

API	Application programming interface eli ohjelmointirajapinta
CamelCase	CamelCasella viitataan tapaan kirjoittaa useista sanoista koostuvat muuttujien, funktioiden tai luokkien nimet siten, että sanat on kirjoitettu yhteen isoin alkukirjaimin. (Wikipedia, 2018a)
CSS	Sanoista Cascade Style Sheet, määrittelee tyylit sivulle tai ohjelmalle
CRNA	create-react-native-app, tapa aloittaa React Native sovellus
Flexbox	Tyylimääritys elementtien asemointiin näkymässä
HTML	Hypertext Markup Language, standardi kieli, jolla tehdään verkkosivujen käyttöliittymiä.
Inline-tyylit	Elementissä määritellyt tyylit style-attribuutin avulla.
Media Query	Mediakyselyjä, joilla määritetään käytettävän näyttölaitteen suuruus ja sivun käyttäytyminen
NPM	Node Package Management Paketinhallinta
Render	Renderöinti eli sovelluksen piirtäminen näkymään
SDK	Ohjelmistokehityspaketti, jonka avulla luodaan sovelluksia
YARN	Samankaltainen paketinhallinta, kuin NPM mutta nopeampi
Widget	suomennettuna vimpaimet ovat pieniä erillisiä ohjelmia/palveluita

Liite 2. Ampparit.com React Native mobiilisovelluksen näkymiä

Kuva 1. Sovelluksen perusnäkymä



Kuva 2. Uutislistaksessa kortteja käännettynä