TAMPEREEN
AMMATTIKORKEAKOULU

MOBIILISOVELLUSTEN ALUSTARIIPPUMATTOMAT OHJELMISTOKEHYKSET

Toni Nahkala

Opinnäytetyö
Joulukuu 2016
Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

NAHKALA, TONI:
Mobiilisovellusten alustariippumattomat ohjelmistokehykset

Opinnäytetyö 30 sivua
Joulukuu 2016

Tämän opinnäytetyön tavoitteena oli vertailla alustariippumattomia mobiilikehitys ohjelmistokehyksiä. Tarkempi vertailu suoritettiin Ionicin ja React Nativen välillä. Työn toimeksiantajana oli web- ja mobiilikehitykseen erikoistunut Haltu Oy. Vertailun tavoitteena oli tuoda toimeksiantajalle tietoa siitä, minkälaisiin projekteihin ohjelmistokehykset sopivat parhaiten.

Vertailussa kehitettiin yksinkertainen esimerkkisovellus, jonka avulla vertailua suoritettiin. Vertailussa etsittiin eroja ohjelmistokehyksien kehitystyökalujen, ohjelmistokehysten tekniikoiden, sovelluksen käyttöliittymän ja käyttöjärjestelmän rajapinnan hyödyntämisen välillä. Eroavaisuuksia hyödynnettiin lopputuloksien päättelemiseksi.

Vertailun tulosten myötä kävi ilmi, että tällä hetkellä etenkin React Native sopii hyvin toimeksiantajan tarpeisiin. React Native tarjoaa paremmat lähtökohdat vaativampien sovellusten kehitykseen antaen mahdollisuuden lisätä mobiilialustojen natiiveja komponentteja ja toimintoja helposti. Ionic sen sijaan sopii paremmin yksinkertaisempien projektien tai prototyyppien tekemisen.

Asiasanat: Mobiilisovelluskehitys, Ionic, React Native

ABSTRACT

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

NAHKALA, TONI:
Cross-Platform Mobile Development Frameworks

Bachelor's thesis 30 pages
December 2016

The purpose of this thesis was to compare cross-platform mobile development frameworks. The comparison was performed mainly between Ionic and React Native. The thesis was commissioned by Haltu Oy which is a company specializing in mobile and web development. The objective of the comparison was to provide guidelines on what types of projects the frameworks would be suitable for.

The comparison was carried out by developing a simple application which helped analyzing the differences between the frameworks. Differences in developer tools, technologies, user interface and access to platform-specific application programming interfaces were considered, when the final conclusions were made.

The results of the comparison determined that currently React Native is better suited for the client's needs. React Native allows much more elaborate applications to be developed with a possibility to add native components and functionality with ease. Ionic is better for creating simpler projects or prototypes.

Key words: Mobile Development, Ionic, React Native

SISÄLLYS

ERITYISSANASTO

ADB	Android Debug Bridge, komentorivi työkalu, joka tarjoaa yhteyden Android-laitteeseen.
AngularJS	JavaScript-ohjelmistokehys SPA-sovellusten kehitykseen MVC-arkkitehtuuria hyödyntäen.
CSS	Cascading Style Sheets, antaa mahdollisuuden määrittää tyyliohjeita.
DOM	Document Object Model, tapa, jolla voidaan kuvata dokumentin rakenne puuna.
FPS	Frames per second, näytölle piirrettyjen kuvien määrä sekunnissa.
HTML	Hypertext Markup Language, kuvauskieli, jota hyödynnetään web-sivuilla.
JavaScript	Ohjelmointikieli, joka tuo dynaamisen toiminnallisuuden web-sivustoihin.
JSX	JavaScript XML, mahdollistaa XML syntaksin käyttämäine JavaScriptissä.
MIT-lisenssi	Ohjelmistolisenssi, joka sallii teoksen vapaan käytön.
MVC	Model-View-Controller, ohjelmistoarkkitehtuurityyli, joka erottaa sovelluksen käyttöliittymän sovelluksen tiedostoista.
MVVM	Model-View-ViewModel, ohjelmistoarkkitehtyyrityyli, jota yleensä hyödynnetään asiakaspuolella kaksisuuntaisen datan sidonnan kanssa.
NPM	Paketinhallintajärjestelmä JavaScript-ympäristöön.
React	JavaScript-ohjelmistokehys käyttöliittymien kehitykseen.
Sass	Syntactically Awesome Stylesheets, skriptikieli, joka laajentaa CSS:ää tuomalla mahdollisuuden käyttää muun muassa muuttujia ja perintää.
SDK	Software Development Kit, sovelluskehitykseen vaadittava pakkaus.
SPA	Single-Page Application, web-sovellus, joka on mahdutettu ruudulle, yhteen sivuun.

WebView	Mobiilialustoilla käytettävä komponentti, joka renderöi web-sisältöä.
UI	User Interface. Käyttöliittymä, eli käyttäjälle näkyvä osuus ohjelmistosta.
XML	Extensible Markup Language, merkintäkieli, jolla kuvataan tietoa ja sen merkitystä.

1 JOHDANTO

Alustariippumaton mobiilisovelluskehitys on yleisesti ajateltuna kustannustehokkaampi sekä nopeampi tapa kehittää mobiilisovelluksia verrattuna natiiviin kehitykseen. Kyseiset tekniikat ovat tästä syystä tärkeitä silloin, kun budjetti tai aikarajoitteet ovat määräävänä tekijänä sovellusta kehittäessä. Ratkaisuksi tehokkaampaan mobiilisovelluskehitykseen on luotu useita erilaisia ohjelmistokehyksiä. Vaikka nopealla silmäyksellä voisi luulla, että ohjelmistokehykset ovat samanlaisia, pyrkimällä samaan lopputulokseen, löytyy ohjelmistokehyksistä kuitenkin huomattavia eroja.

Tämän työn tavoitteena on vertailla alustariippumattomista mobiilikehitys-ohjelmistokehyksistä Ioniciä ja React Nativea. Vertailun tavoitteena on tuoda toimeksiantajalle tietoa ja ohjeistusta siitä, mihin käyttötarkoituksiin ohjelmistokehykset sopivat. Työn tarkoituksena on kehittää esimerkkisovellus sekä Ionicillä että React Nativella. Esimerkkisovellusta hyödynnetään ohjelmistokehyksien vertailussa. Vertailtavat ohjelmistokehykset valittiin sen perusteella, että ne ovat suosituimpia JavaScript-pohjaisia ohjelmistokehyksiä kehittäjien keskuudessa alustariippumattomien mobiilisovellusten kehitykseen (Owens, 2016). Lisäksi toimeksiantaja on jo molempia ohjelmistokehyksiä kokeillut eri projekteissa. Vertailtavat ohjelmistokehykset edustavat erilaisia lähestymistapoja mobiilisovelluskehitykseen, joka tuo eri näkökulmia vertailuun.

Vertailtavista ohjelmistokehyksistä vertaillaan kehittäjän työkaluja, kirjastoja, joita ohjelmistokehykset hyödyntävät, niiden tarjoamia mahdollisuuksia ja haasteita mobiilialustojen rajapintaan liittyen, koodin rakennetta sekä mahdollisuudet ja haasteet käyttöliittymän osalta. Vertailtavat asiat ovat valittu siten, että vertailun tulokset auttavat luomaan johtopäätöksiä ohjelmistokehyksien käyttötapauksista toimeksiantajaa varten.

2 TAUSTAA

2.1 Toimeksiantaja

2.1.1 Toimeksiantajan esittely

Toimeksiantajana työlle toimii Haltu Oy, joka on mobiili- ja web-kehitykseen erikoistunut ohjelmistoalan yritys. Toimeksiantajalla on monesti mietitty eri ratkaisuja siihen, miten saadaan tehokkaimmalla tavalla aikaan mobiilisovelluksia useille eri alustoille. Asioita, joita on täytynyt ottaa projekteissa yleensä huomioon, ovat olleet mobiilialustavaatimukset, aikarajoitteet, tarvittavat toiminnot sovelluksissa sekä projektien budjetit. Nämä tekijät ovat ohjanneet etsimään uusia tekniikoita, jotka mahdollistaisivat mobiilisovelluskehityksen entistä tehokkaammin.

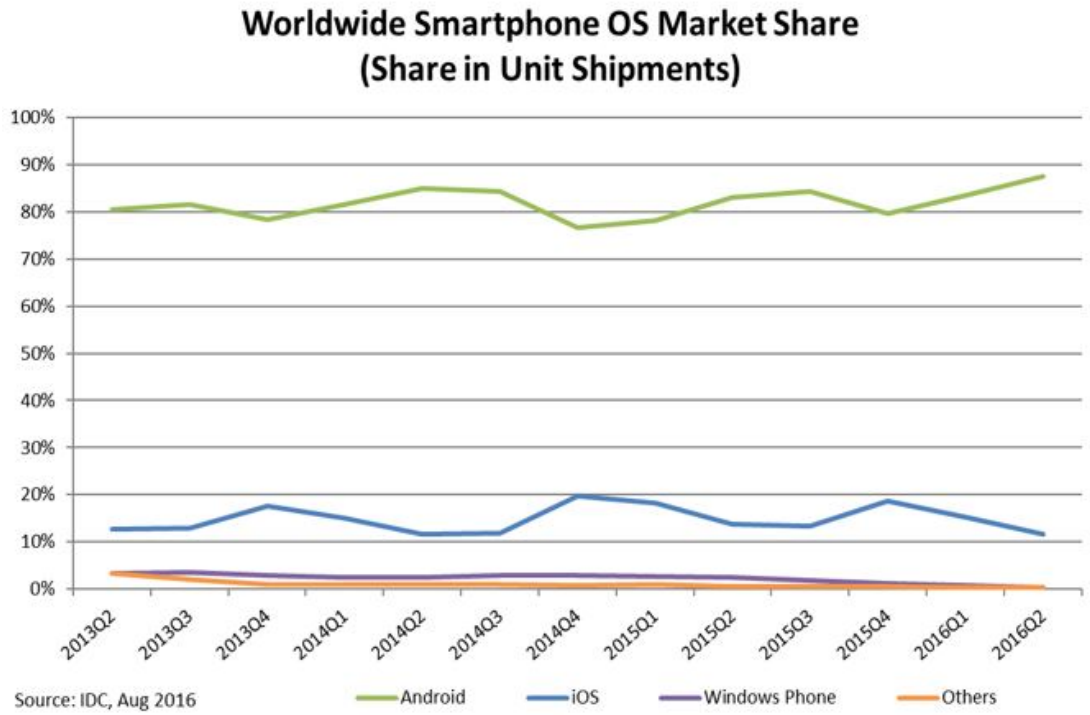
2.1.2 Nykyiset tekniikat

Tällä hetkellä lähes kaikki toimeksiantajan kehittämistä mobiilisovelluksista ovat tehty usealle eri mobiilialustalle. Useimmiten sovellukset ovat tehty alustariippumattomiksi luomalla natiivisti WebView-komponentti tarvittaviin alustoihin ja ottamalla yhteys palvelimeen, missä sovelluksen koodi sijaitsee. Nämä sovellukset ovat kehitetty hyödyntäen Django-ohjelmistokehystä MVC-arkkitehtuurina. Käyttöliittymä on tällöin rakennettu web-tekniikoita hyödyntäen. Uusimpina tekniikkoina mobiilisovelluksissa on kokeiltu Ioniciä ja React Nativea, joita on tässä työssä tarkoitus vertailla tarkemmin.

2.2 Mobiilialustat

Mobiilisovelluksia kehitettäessä on tärkeää tietää kohderyhmällä olevia mahdollisia eri käyttöjärjestelmiä. Tällä hetkellä mobiilikehityksestä puhuttaessa, mobiilialustat, jotka eniten merkitsevät, ovat Android, iOS ja Windows Phone. Näistä alustoista etenkin Android ja iOS hallitsevat maailmalla käyttäjämäärissä (kuva 1). Toimeksiantajalla tehdyistä mobiilisovelluksista, ovat versiot yleensä tehty Androidille sekä iOS:lle. Muilla

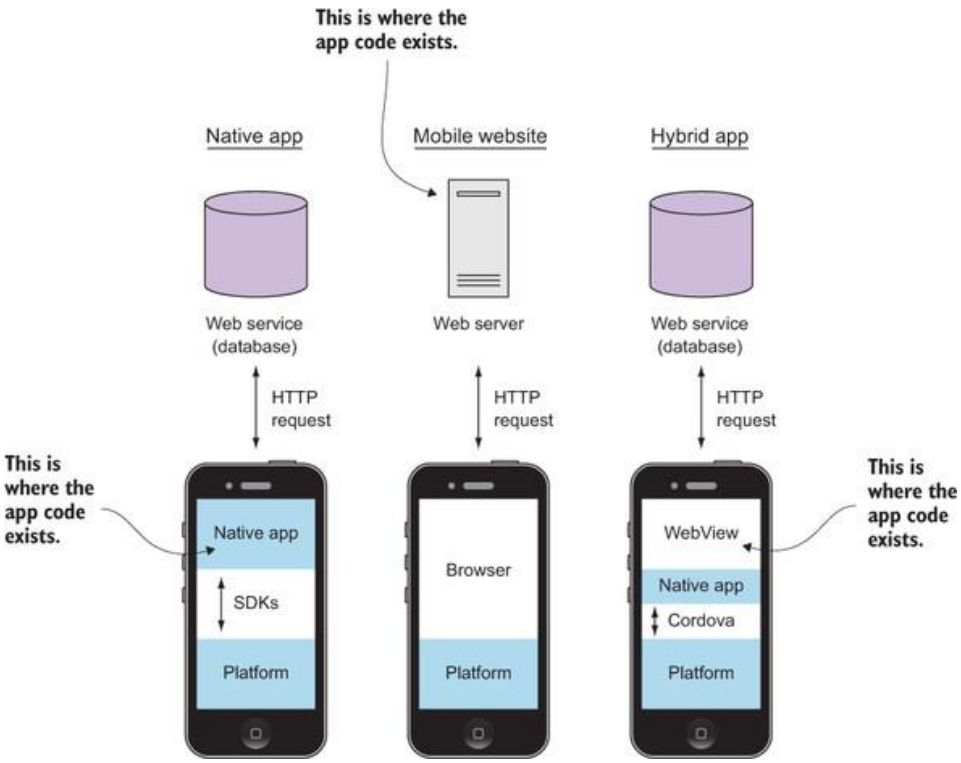
alustoilla, kuten Windows Phonella ovat käyttäjämäärät erittäin pieniä ja tästä syystä Windows Phone on toimeksiantajan sovelluksissa ollut vain tarvittaessa mukana. Toimeksiantajalla on kuitenkin ollut muutamia projekteja, joissa sovellus on tehty myös Windows Phonelle, asiakkaan näin toivoessa.



KUVA 1. Mobiilikäyttöjärjestelmien jakauma (Smartphone OS market share, 2016)

2.3 Mobiilikehitys usealle alustalle

Puhuttaessa mobiilikehityksestä usealle alustalle, on hyvä erottaa erilaiset keinot kehittää mobiilisovelluksia. Lähestymistavat jaetaan tyypillisesti kolmeen eri vaihtoehtoon. Kuten kuvassa 2 on esitetty, voi mobiilisovelluksen luoda natiivina, mobiiliverkkosivuna tai hybridisovelluksena.



KUVA 2. Mobiilisovellusarkkitehtuuri (Wilken, 2015)

2.3.1 Natiivi mobiilisovellus

Natiivit mobiilisovellukset tarkoittavat kehittäjän näkökulmasta sovelluksen koodaamista jokaiselle alustalle erikseen. Koodin lisäksi, kehitys vaatii myös käyttöliittymän suunnittelua alustoille erikseen. Kehittäessä natiivisti, pääsee kehittäjä kuitenkin hyödyntämään mobiilialustojen rajapintoja ja puhelimien toimintoja parhaiten. Suorituskyky on myös isossa roolissa, kun kehitetään sovellus natiivisti. Esimerkiksi pelit, jotka vaativat kovaa suorituskykyä pelimoottorin pyörittämiseen sekä sulavan käyttökokemuksen tarjoamiseksi, vaativat lähes poikkeuksetta natiivia kehitystä.

2.3.2 Mobiiliverkkosivu

Mobiiliverkkosivu tarkoittaa siis käyttäjän näkökulmasta normaalia verkkosivua. Käyttäjän huomaa eron perinteisiin mobiilisovelluksiin siinä, että hän ei sovellusta asenna puhelimeen missään vaiheessa. Kuten kuvasta 2 esitetään, mobiiliverkkosivuna kehitetyn sovelluksen koodi sijaitsee aina palvelimella ja puhelin hyödyntää verkkoselainta sovelluksen avaamiseen. Perinteisistä mobiilisovelluksista mobiiliverkkosivut eroavat myös

sen osalta, että sovelluksen päivitys ei vaadi sovellusten lataamista kauppapaikoille uudestaan. Kehittäjä ei kuitenkaan tällä tyylillä pysty hyödyntämään täysin kaikkia puhelimen ominaisuuksia. Esimerkiksi puhelimen yhteystietoihin ei verkkosivusto pääse käsiksi, mutta muun muassa kameran käyttö on sen sijaan mahdollista (Introduction to the Camera API, 2016). (Saleh, 2014)

2.3.3 Hybridi mobiilisovellus

Kolmantena vaihtoehtona on luoda hybridisovellus, joka tuo yhden teknologian, sovelluslogiikan yhdistämiseksi usealle eri mobiilialustalle. Yleensä hybridisovelluksissa käytetään web-teknologioita (HTML, CSS ja JavaScript). Muutamia poikkeuksia kuitenkin löytyy kuten Xamarin, joka hyödyntää C#:a ohjelmointikielenä ja luo pohjimmillaan natiivin sovelluksen. Hybridisovellusten arkkitehtuuri toimii yleensä siten, että ohjelmistokehys, joka on valittu, luo WebView-komponentin, jossa näytetään HTML-elementtejä. Käytännössä tämä on siis verkkosivu, joka on asennettu suoraan puhelimeen. Erona näissä sovelluksissa mobiiliverkkosivuihin on kuitenkin se, että ohjelmistokehykset kuten Apache Cordova, jonka päälle monet hybridi ohjelmistokehykset ovat rakennettu, hyödyntävät puhelimen käyttöjärjestelmän rajapintaa. Hybridi mobiilisovelluskehityksessä on mahdollista hyödyntää mobiilialustan rajapintaa lähes yhtä kattavasti kuin, kehittäessä sovellusta natiivisti. (Panhale, 2016)

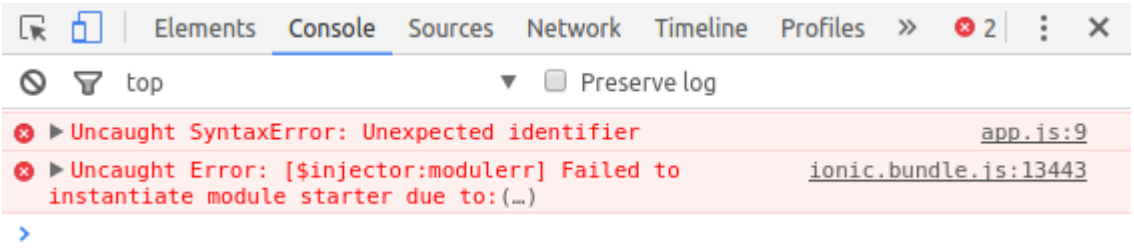
3 IONIC JA REACT NATIVE

3.1 Ionic

Ionic on Apache Cordovan päälle rakennettu ohjelmistokehys, joka mahdollistaa mobiilisovelluksien kehittämisen alustariippumattomasti. Ionic tarjoaa mahdollisuuden kääntää sovelluksen Android sekä iOS -laitteille. Toistaiseksi vielä beta-vaiheessa oleva Ionic 2, tulee tarjoamaan tuen myös Windows Phone -laitteille. Ionicillä tehdyt sovellukset toimivat luomalla puhelimeen natiivin WebView-komponentin, jossa web-tekniikoilla kehitetty sovellus ajetaan. Koska normaalilla web-sovelluksella on erittäin rajattu mahdollisuus hyödyntää puhelimen ominaisuuksia, on Cordovan osuus tuoda mahdollisuus kutsua mobiilikäyttöjärjestelmän rajapintaa JavaScript-koodilla (Wargo, 2014).

3.1.1 Kehittäjän työkalut

Kehityksen näkökulmasta Ionic-kehitys toimii pitkälti kuten front-end web-kehitys. Vaikka kyseessä on ohjelmistokehys, jolla tehdään mobiilisovelluksia, ei puhelin tai emulaattori ole pakollisia välineitä. Kehittämistä voi hoitaa paljon pelkän verkkoselaimen avulla. Tiettyjä puhelimien ominaisuuksia kuten kameraa tai värinää ei kuitenkaan pysty selaimen välityksellä testaamaan. Logitus onnistuu helpoiten käyttämällä selaimen konsolia, kuten kuvassa 3 on esitetty. Käyttämällä esimerkiksi Google Chromen kehittäjän työkaluja, pystyy kehittäjä debuggaamaan sovellusta mielestäni erittäin paljon helpommin, kuin käyttämällä komentoriviä. Google Chromen kehittäjän työkalujen avulla onnistuu myös esimerkiksi DOM-elementtien tai HTTP-kutsujen tutkiminen.



KUVA 3. Virhetilanne esitettynä Google Chromen kehittäjän työkalujen avulla

Kehittäessä Ionicillä, nopeutta prosessiin tuo Live Reload -toiminto, joka päivittää näkymän aina, kun taustalla havaitaan muutoksia lähdekoodissa. Tämä on tuttu toiminto web-kehityksessä ja toiminto, joka on pitkään puuttunut natiivista mobiilisovelluskehityksestä, vasta huhtikuussa 2016 julkaistun Android Studio 2.0:n myötä Instant Run -niminen toiminto, joka toimii kuten Live Reload, tuotiin natiiviin kehitykseen (Android Studio User Guide).

Ionic tarjoaa laajan kirjon komentorivitoimintoja. Toimintoihin kuuluu esimerkiksi sovelluksen lähettäminen Ionic Cloud -palveluun, joka on pilvipalvelu mobiilisovelluksien julkaisuun, statistiikan keräämiseen ja käyttäjien todentamiseen (Ionic Cloud, 2016). Komentoriviltä on myös mahdollisuus generoida sovelluksen ikonit ja sovelluksen käynnistyessä näkyvä splash screen. Generointi onnistuu asettamalla oikean kokoinen kuva projektihakemiston resurssi-kansioon. Generointi luo tarvittavat kuvat molemmille alustoille. (Ionic Documentation, CLI, 2016)

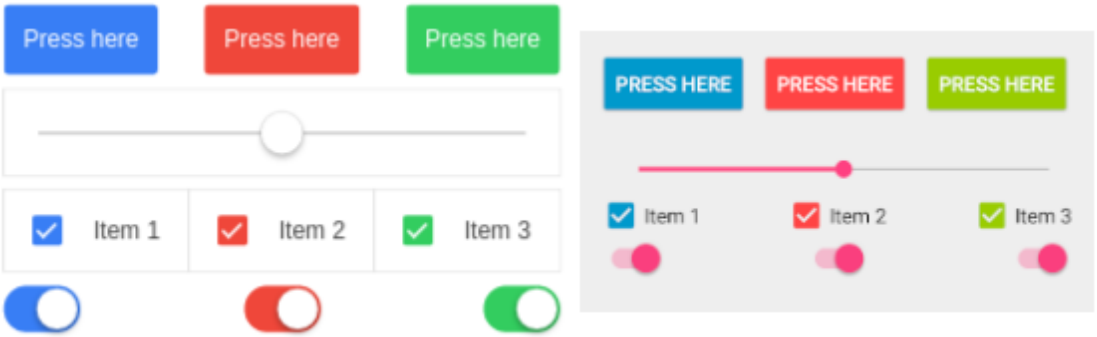
3.1.2 Tekniikat

Kehitys Ionicillä tapahtuu HTML:n, CSS:n ja JavaScriptin avulla. JavaScript-kirjastona Ionic hyödyntää AngularJS:ää, joka perustuu MVVM-malliin. AngularJS hyödyntää kaksisuuntaista datan sidontaa scopen avulla, joka tuo päivittää data mallin, kun käyttäjä tekee muutoksia käyttöliittymässä. Riippuvuuksien tuominen sovellukseen hoidetaan riippuvuus inejektioilla, jonka tarkoituksena on helpottaa riippuvuuksien muokkausta (AngularJS Developer Guide, Dependency Injection, 2016). Helppo testattavuus on myös yksi tärkeä osa-alue, johon AngularJS pyrkii. AngularJS tuo muun muassa omat mock-objektit, jotka auttavat emuloimaan esimerkiksi HTTP-kutsuja. Ionic tulee esiin yhtenä AngularJS:n moduuleista. Se tuodaan riippuvuutena sovellukseen, jonka jälkeen sovelluksessa voidaan hyödyntää Ionicin ominaisuuksia (Wilken, 2015). (Tarasiewicz & Böhm, 2014)

3.1.3 Käyttöliittymä

Ionic toimii mobiilisovelluksessa UI-kirjastona, joka tarjoaa valmiita elementtejä ja tyylejä, joita voi helposti ottaa käyttöön sovelluksessa. Ionicin tarjoamia elementtejä ovat

muun muassa painikkeet, listat ja välilehdet. Käyttöliittymän tyylittelyyn käytetään CSS:ää, mutta projektiin on mahdollista ottaa mukaan Sass helposti. Etenkin Sassia hyödyntäen värimaailma on mahdollista kustomoida laaja-alaisesti todella nopeasti. Kustomointia pystyy näiden lisäksi tekemään myös käyttöjärjestelmäkohtaisesti, joka auttaa seuraamaan mobiilialustojen omia UI periaatteita (Phan, 2015). Esimerkiksi välilehtien navigaatio kuuluu iOS-laitteilla olla puhelimen alareunassa (iOS Human Interface Guidelines, 2016) ja Android-laitteilla se kuuluu yläreunaan (Android Design Principles, 2016).



KUVA 4. Ionicin (vasemalla) sekä Androidin (oikealla) UI-komponentteja

Kuvassa 4 esitetään muutamia eri UI-komponentteja, joita Ionic tarjoaa suoraan ohjelmoijalle. Kuvassa näkyvät komponentit ovat perinteisiä HTML-elementtejä, joita Ionicin kehittäjät ovat tyylitelleet valmiiksi. Näitä elementtejä voi ottaa käyttöön käyttämällä dokumentaatiosta löytyvillä CSS-luokilla tai Ionicin omilla tunnisteilla (kuva 5). Sassia hyödyntäen voi helposti muuttaa valmiiden CSS-luokkien tyyliä. Kuvassa 6 on esimerkki kuinka painikkeen assertive-luokan värit ovat määritelty. Komponenttien lisäksi Ionicin mukana tulee myös ikoneita ilman erillistä latausta. Ionicons-nimellä kulkeva kokonaisuus on MIT-lisenssin alaisena, joten ikoneita voi vapaasti käyttää sovelluksissa.

```
<ion-checkbox class="checkbox-balanced">Item 3</ion-checkbox>
<button class="button button-assertive">Press here</button>
```

KUVA 5. Ionicin valmiiden tyylien käyttöönotto

```
$assertive:                    #ef473a !default;

$button-assertive-bg:          $assertive !default;
$button-assertive-text:        #fff !default;
$button-assertive-border:      darken($assertive, 10%) !default;
$button-assertive-active-bg:   darken($assertive, 10%) !default;
$button-assertive-active-border: darken($assertive, 10%) !default;
```

KUVA 6. Sassin käyttö Ionicissa

Reititykseen Ionic ei käytä AngularJS:n virallista reititintä, ngRoutea. Virallisen reitittimen kanssa ei onnistuisi kaikki mobiilisovelluksessa tarvittava näkymien muokkaus. Sen sijaan Ionic hyödyntää UI-Router -kirjastoa, jonka avulla Ionicissä voidaan tehdä esimerkiksi sovellus, jossa on usea välilehti ja jokainen välilehdistä on oma näkymä. Näkymien vaihto toimii kuten SPA-sovelluksessa. Koodissa asetetaan ion-view-tunniste, jonka sisällä toimii näkymä, joka halutaan näkyville tai piiloon. Kuvassa 7 esitetään sovellus, jossa on kaksi eri näkymää. Kutsumalla changeView-funktiota, vaihdetaan secondView-näkymä näkyviin. Näkymät esitellään $stateProvider-palvelulle ja $urlRouterProvider-palvelu antaa mahdollisuuden esittää osoitteen, johon navigoidaan silloin, jos tehdään esimerkiksi epäkelpoinen kutsu. (Wilken, 2015)

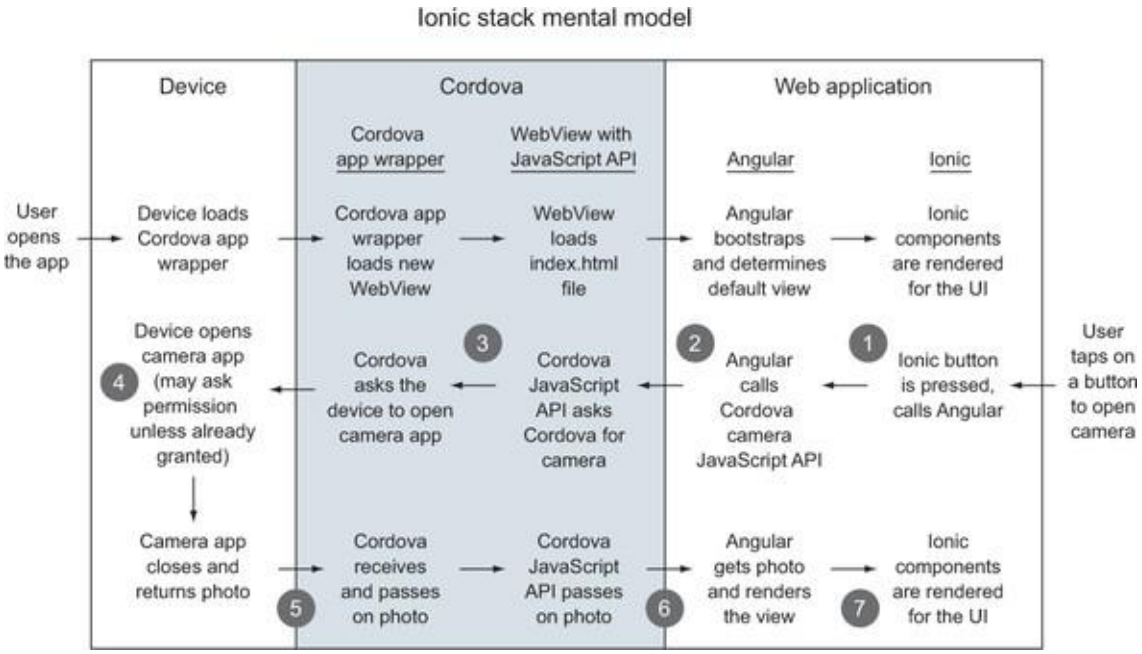
```
$scope.changeView = function() {
  $state.go('secondView');
}

.config(function($stateProvider, $urlRouterProvider) {
  $stateProvider
    .state('firstView', {
      url: '/first-view',
      templateUrl: 'views/first-view.html',
      controller: 'MainCtrl'
    })
    .state('secondView', {
      url: '/second-view',
      templateUrl: 'views/second-view.html',
      controller: 'MainCtrl'
    })
  $urlRouterProvider.otherwise('/first-view');
})
```

KUVA 7. Näkymän vaihto Ionicissa

3.1.4 Käyttöjärjestelmän rajapinta

Kun Ionicilla halutaan käyttää mobiilialustan omaa rajapintaa, hyödynnetään tällöin Apache Cordovaa. Cordova mahdollistaa puhelimen toiminnallisuuksien hyödyntämisen JavaScriptin avulla. Cordovan tukemat toiminnallisuudet vaihtelevat paljon mobiilialustoittain, mutta tärkeimpiä toimintoja puuttuu lähinnä vain vähemmän käytetyistä käyttöjärjestelmistä. Käyttöjärjestelmiä, joita Cordova tukee, ovat muun muassa Android, iOS, Symbian, Firefox OS ja BlackBerry. Ionic sen sijaan tukee näistä käyttöjärjestelmistä vain Androidia ja iOS:ia. (Wargo, 2014)



KUVA 8. Ionicin toimintamalli (Wilken, 2015)

Kuvassa 8 esitetään Ionicin käyttämää arkkitehtuuria, kun hyödynnetään puhelimen toimintoja. Kuvan esimerkissä käyttäjä ottaa kuva painamalla alkuun käyttöliittymässä olevaa painiketta, jonka jälkeen Ionic kutsuu Cordovan rajapintaa ja Cordova tekee lopullisen kutsun mobiilialustan rajapintaan. Laitteen antaessa vastauksen, sama pino mennään päinvastoin läpi, jonka jälkeen Ionic hoitaa lopullisen kuvan renderöinnin käyttöliittymään.

Kuvassa 9 näytetään kuvan 8 esimerkki koodin avulla. Kuvassa on käytetty kameraa, joka vaatii vain yhden funktion kutsun, jossa on parametreina annettu kameran asetukset. Asetuksiin on mahdollista muokata muun muassa kuvan laatua tai kuvan kokoa. Funktio palauttaa kuvan ja se asetetaan scopen avulla näkyville img-tunnisteeseen.

```
$scope.takePicture = function() {
  const options = {
     quality : 100,
     targetWidth: 300,
     targetHeight: 300,
     sourceType: 1,
     cameraDirection: 0,
     encodingType: 0,
     allowEdit: false,
     saveToPhotoAlbum: false,
     correctOrientation: false,
  };

  $cordovaCamera.getPicture(options)
    .then(function(imageData) {
       $scope.image = imageData;
    }, function(error) {
       alert(error)
    });
}

<img ng-src="{{image}}">
```

KUVA 9. Esimerkki kameran käytöstä Ionicissä

3.2 React Native

React Native on Facebookin työntekijöiden kehittämä avoimen lähdekoodin ohjelmistokehys. Se on kehitetty hyödyntäen React-arkkitehtuuria, joka niin ikään on Facebookin työntekijöiden kehittämä. React Native mahdollistaa JavaScript-kehityksen Android ja iOS -laitteille, hyödyntäen alustojen natiiveja UI-elementtejä.

3.2.1 Kehittäjän työkalut

Kehitys React Nativella toimii pitkälti kuten natiivi mobiilikehitys. Koska kyseessä on pohjimmiltaan natiivi mobiilisovellus, kehittäjän tulee käyttää puhelinta tai emulaattoria sovelluksen testaamiseen. Logitus toimii myös kuten natiivissa kehityksessä esimerkiksi Anroidilla voi hyödyntää ADB:n logcat-komentoa, jonka avulla saa komentoriville järjestelmän viestejä kuten sovelluksen virheilmoitukset.

KUVA 10. React Native virheenhallinta

Kun kehittäjä tekee jonkinnäköisen virheen ohjelmoidessaan, tuo React Native puhelimeen punaisen ruudun, joka antaa virheen lisäksi pinovedoksen näkyviin (kuva 10). Kyseiseen näkymään on lisätty myös toiminto, joka antaa kehittäjän hypätä suoraan tiedostoon, jonka hän valitsee pinovedoslistasta. React Native tarjoaa virheenhallinan lisäksi myös paljon tarkkaa tietoa sovelluksen kehitystä varten. Sovelluksen ollessa auki, kehittäjä voi poistaa asetusvalikosta muun muassa Dev mode -asetuksen, joka ei tuo virheitä näkyviin ja näin mahdollistaa sovelluksen suorituskyvyn testauksen. Mahdollista on myös tuoda näkyviin esimerkiksi FPS-monitorointi. Natiiviin kehitykseen verrattuna nopeutta tuo Hot Reloading -toiminto, joka tuo muutokset näkyviin heti tiedoston tallennuksen jälkeen. Tämä toiminto kuitenkin vaatii sovelluksen uudelleen kääntämisen, jos sovellukseen lisätään toimintoja, jotka ovat vaatineet natiivia koodia.

3.2.2 Tekniikat

React Native perustuu Reactiin, joka on JavaScript-kirjasto koottavien käyttöliittymien tekemiseen (de Sousa Antonio, 2015). React on siis MVC-arkkitehtuurin View-osuus eli näkymä. Reactin ideana on jakaa sovellus komponentteihin. Komponenteilla tarkoitetaan Reactissa HTML-elementtejä. Komponentti on JavaScript funktio, jolle voi syöttää prop-argumentteja, jotka tuo dataa komponenttiin. Kuvassa 11 on esitetty navigaatiopalkki komponenttina, jolle on annettu näkymän nimi proppina. Näkymän nimi tuodaan näkyville navigaatiopalkin sisälle Text-komponenttiin. (Feldman, Bagnardi, Højberg & Hall, 2016)

```
export default class Navbar extends Component {
  render() {
    return (
      <View style={styles.container}>
        <Text>{this.props.title}</Text>
      </View>
    );
  }
}

<Navbar title={'View title'} />
```

KUVA 11. Navigaatiopalkki-komponentti React Nativessa

React tuo perinteiseen DOM-manipulointiin uuden tavan VirtualDOMin ansiota. VirtualDOM on kirjasto, joka mahdollistaa muutosten tekemisen vain niihin elementteihin, joissa on muutosta tapahtunut. Taustalla kyseisessä kirjastossa toimii algoritmi, joka tarkastaa jokaisen tilan muutoksen yhteydessä eroja edelliseen tilaan ja tämän avulla osaa tehdä muutokset vain siihen elementtiin, jota on oikeasti muutettu. Tämä muutoksen on tarkoitus tuoda parempaa suorituskykyä sovellukselle, koska liiallinen DOM-manipulointi käy raskaaksi puhelimelle tai verkkoselaimelle. (Eisenman, 2015)

Ensisilmäyksellä React Nativen koodi saattaa näyttää melko epäselvälle, joka johtuu osittain JSX:n käytöstä. JSX toimii luokan render-funktiossa, joka palautetaan ja render-funktiossa olevat elementit tuodaan ruudulle. Yksinkertaisimmillaan JSX:llä luotu komponentti näyttää melko paljon perinteiseltä HTML:ltä, mutta jos mukaan tuodaan esimerkiksi konditionaaleja, jotka vaikuttavat siihen miten elementtejä tulee näyttää, muuttuu koodi erittäin sekavaksi. Kuvassa 12 esitetään konditionaalien käyttämisestä esimerkki kahdella eri tavalla. Esimerkissä komponenttiin lisätään käyttäjän mobiilialustasta riippuen tekstikenttä ja taustaväriä muutetaan.

```
class View extends Component {
  render() {

    var platformText = null;

    if (Platform.OS === 'android') {
      platformText = <Text>Android</Text>;
    }

    return (
      <View style={{
        backgroundColor: Platform.OS === 'android' ? 'red' : 'blue'
      }}>
        <Text>Text...</Text>
        {platformText}
      </View>
    );
  }
}
```

KUVA 12. JSX React Nativessa

3.2.3 Käyttöliittymä

React Native eroaa hieman useimmista JavaScript-pohjaisista mobiilikehitys-ohjelmistokehyksistä sen ansiota, että se mahdollistaa natiivien UI-komponenttien käytön, mikä ei ole mahdollista ohjelmistokehyksissä, jotka käyttävät WebViewiä. React Nativessa on silta, joka yhdistää JavaScript koodin Androidin ja iOS:n rajapintoihin. Tämä tuo React Nativeen natiiveja komponentteja, kuten esimerkiksi painikkeet tai tekstisyötteet. Tämä mahdollistaa natiivin mobiilisovelluskehityksen yhdellä ohjelmointikielellä. Kuvassa 13 esitetään natiivin painikkeen luonti Android-puhelimille. Ensin tulee tarkastaa, että käyttäjän alusta on Android, jonka jälkeen TouchableNativeFeedback-komponentti luo tyhjän painikkeen. Kyseisen painikkeen lapsielementtinä oleva View, luo tarvittavan ulkoasun. (Eisenman, 2015)

```
{Platform.OS === 'android' &&
  <TouchableNativeFeedback onPress={this.props.onPress}>
    <View style={styles.button}>
      {this.props.children}
    </View>
  </TouchableNativeFeedback>
}
```

KUVA 13. Natiivin painikkeen luominen React Nativessa

Näkymien vaihtoon voi käyttää Navigator-komponenttia, joka on JavaScriptillä implementoitu alustariippumaton komponentti tähän tarkoitukseen. Navigator-komponentti ei siis luo natiiveja näkymiä. NavigatorIOS-komponenttia käyttämällä on mahdollista kuitenkin saada iOS-laitteille natiivi navigaatiojärjestelmä, joka mahdollistaa iOS-käyttöjärjestelmän animaatiot ja toiminnot (React Native Docs, NavigatorIOS, 2016). Kuvassa 14 esitetään, kuinka napin painalluksen yhteydessä tapahtuu vaihto SecondView-näkymään. Käyttämällä push-funktiota, voi SecondView-näkymästä palata takaisin käyttämällä pop-funktiota. Näiden lisäksi on mahdollista käyttää esimerkiksi resetTo-funktiota, jonka avulla näkymään voidaan siirtyä ja nollata samalla näkymähistoria. (Eisenman, 2015)

```
onButtonClick() {
  this.props.navigator.push({
    id: 'SecondView',
  });
}

render() {
  return (
    <Navigator
      initialRoute={{id: 'FirstView'}}
      renderScene={this.navigatorRenderScene} />
  );
}

navigatorRenderScene(route, navigator) {
  switch (route.id) {
    case 'FirstView':
      return(<FirstView navigator={navigator} route={route} title='FirstView' />);
    case 'SecondView':
      return(<SecondView navigator={navigator} route={route} title='SecondView' />);
  }
}
```

KUVA 14. Navigator-komponentin käyttö React Nativessa

3.2.4 Käyttöjärjestelmän rajapinta

Monet tärkeimmistä mobiilialustojen toiminnoista sekä komponenteista on jo implementoitu React Nativen rajapintaan, mutta kehittäjä joutuu melko pian joko käyttämään kolmannen osapuolen tekemiä moduuleja tai kirjoittamaan itse tarvittavia natiivimoduuleja saadakseen puuttuvia toimintoja lisättyä. React Nativen dokumentaatiossakin kerrotaan kuinka kaikkia tarvittavia komponentteja ei ole suoraan saatavilla edes kolmannen osapuolen kirjastojen avulla. Tätä varten dokumentaatiosta löytyy ohjeet kuinka kehittäjä voi luoda omia natiivimoduuleja. Nativiimoduulit yhdistävät kehittäjän JavaScript-koodin

sekä mobiilialustan natiivin koodiin. Natiivimoduulit ovat tärkeä osa React Nativen toiminnallisuuksia ja tästä syystä dokumentaatiossa on ohjeet, kuinka kehittäjä voi oman moduulin luoda. Natiivimoduulit vaativat luonnollisesti osaamista natiivista mobiilisovelluskehityksestä. Natiivinmoduulin luodessaan kehittäjän tulee seurata React Nativen dokumentaatiosta löytyviä ohjeita, jotta lisätty moduuli toimii oikein React Nativen kanssa. (React Native Docs, Native Modules, 2016)

Kun haluttu moduuli on löydetty, tulee se ensin asentaa NPM:n avulla. Tämän jälkeen moduuli tulee linkittää projektiin. Linkitys toimii joko React Nativen link-komennolla tai manuaali linkitys, joka vaatii natiivikoodin pientä muokkaamista. Kun tarvittava moduuli on lisätty projektiin, on puhelimen ominaisuuksia yleensä erittäin helppo käyttää. Kuvassa 15 on lisätty kolmannen osapuolen luoma react-native-camera-niminen moduuli, joka mahdollistaa kameran kuvan näyttämisen ruudulla. Kuvan esimerkissä painiketta painamalla kutsutaan takePicture-funktiota, joka ottaa kuvan ja esittää sen Image-komponentissa. Kun React Nativella käytetään toimintoja, kuten kameraa, tulee tarvittavat asetukset toiminnolle lähettää proppeina. Kuvan 15 esimerkissä puhelimelle ilmoitetaan muun muassa siitä, että puhelimen tulee käyttää takakameraa ja soittaa äänimerkki, kun kuva on otettu.

```
takePicture() {
  this.camera.capture()
    .then((data) => {
      this.setState({
        imageUri: data.path,
      });
    })
    .catch((error) => {
      alert(error);
    });
}

<Camera
  ref={(cam) => { this.camera = cam; }}
  type='back'
  orientation='portrait'
  keepAwake={false}
  mirrorImage={false}
  playSoundOnCapture={true}
  aspect={Camera.constants.Aspect.fill}
  captureTarget={Camera.constants.CaptureTarget.disk}
  flashMode={Camera.constants.FlashMode.auto}
  captureQuality={Camera.constants.CaptureQuality.medium}
  captureMode={Camera.constants.CaptureMode.still}
  style={styles.camera}>
</Camera>
<Image
  source={{ uri: this.state.imageUri }}
  style={styles.image} />
<Button onPress={this.takePicture.bind(this)}>
  <Text>Take picture</Text>
</Button>
```

KUVA 15. Kameran käyttö React Nativessa kolmannen osapuolen moduulin avulla

4 YHTEENVETO

4.1 Erot ja yhteneväisyydet

4.1.1 Sovelluskehitys

Kehittäjille tarkoitetut työkalut eroavat vertailtavilla ohjelmistokehyksillä merkittävästi. React Native tarjoaa paljon tietoa ja asetuksia, jotka auttavat kehittämään ja testaamaan eri osuuksia sovelluksista. Tästä hyvänä esimerkkinä toimii Dev Mode -toiminto, joka päällä ollessaan näyttää virheet ja varoitukset kuten kuvassa 5 on esitetty. Jos tämä asetus asetetaan pois päältä, voi kehittäjä testata sovelluksen suoritustehoa ilman, että sovelluksesta käännetään erikseen tuotantoversiota. Ionic ei puolestaan vastaavanlaisia mahdollisuuksia tarjoa, mutta tarjoaa mahdollisuuden kehittää sovellusta ilman puhelinta tai emulaattoria, joka ei React Nativella ole mahdollista.

Yhtäläisyyksiä löytyy siinä, kuinka molemmat ohjelmistokehyksistä tarjoavat toiminnon, joka tuo muutoksia ruutuun, kun kehittäjä muutoksia tehnyt. Tämä on asia missä natiivi mobiilisovelluskehitys on pitkään ollut jäljessä.

4.1.2 Tekniikat

Vertailtaessa Ionicin ja React Nativen arkkitehtuuria päädytään paljon mielipidekysymyksiin siitä, onko AngularJS:n vai Reactin arkkitehtuuri. Toimeksiantajan näkökulmasta, etenkin mobiilisovelluskehityksen osalta ei arkkitehtuurilla ole niinkään väliä. Mutta esimerkiksi AngularJS:n osalta kaikkia vahvuuksia ei pysty täysin hyödyntämään. Esimerkiksi AngularJS:n tarjoama helppo testattavuus jää hyödyntämättä, sillä toimeksiantajan projekteissa ei aina testausta suoriteta.

4.1.3 Käyttöliittymä

Käyttöliittymän rakentamisessa erot näkyvät siinä, kuinka ohjelmistokehykset ovat pohjimmiltaan rakenettu. Ioncin ollessa WebView-pohjainen sovellus, tarkoittaa sovelluksen käyttäjälle sitä, että sovelluksessa näkyvät komponentit ovat vain tyyliteltyjä HTML-elementtejä, kun taas React Nativessa elementit ovat natiiveja. Sovelluksen käyttäjältä tämä ero voi kuitenkin jäädä huomaamatta. Käyttöliittymän myötä ohjelmistokehyksissä ilmenee erot myös ohjelmistokehyksien tavoitteessa. Ionicin ollessa kirjasto, joka tarjoaa valmiiksi tyyliteltyjä UI-komponentteja, tarjoten laajat muokkaus mahdollisuudet. React Native sen sijaan toimii kuten natiivikehitys. Komponentit ovat natiiveja, mutta kehittäjän tulee määrittää tyylit kokonaan itse.

Yhteneväisyyksiä löytyy käyttöliittymään liittyen ohjelmistokehyksien navigaatiojärjestelmästä. Molemmat kehyksistä hoitavat näkymän vaihtamisen SPA-sovelluksista tuttuun tyyliin, vaihtamalla ruudusta vain tietty osuus. Kun kyseessä on mobiilisovellus tämä tarkoittaa lähes koko ruudun näkymän vaihtoa, mutta esimerkiksi navigaatiopalkkia ei kuitenkaan yleisesti haluta luoda erikseen joka näkymälle.

4.1.4 Käyttöjärjestelmän rajapinta

Ohjelmistokehykset toimivat molemmat samalla periaatteella puhelimen toimintoja kuten kameraa hyödynnettäessä. Kehittäjän tulee etsiä sopiva moduuli mikä projektiin lisätään. Toiminnot voi tehdä itse, mutta käyttäessä JavaScript-pohjaisia ohjelmistokehyksiä on usein hyödynnettävä kolmannen osapuolen tekemiä kirjastoja, aikaa säästääkseen. Kehittäjältä menisi erittäin paljon aikaa turhaan siihen, että lähtisi tekemään jokaisen kirjaston itse.

Ionicin ja React Nativen suhtautuminen natiiviin koodiin eroaa selkeästi. React Nativea kehittäessä osaaminen natiivista mobiilisovelluskehityksestä nousee suurempaan rooliin kuin Ionicilla kehittäessä. React Nativen dokumentaatioon lisätyt ohjeet siitä, kuinka natiiveja moduuleja voi projektiin lisätä, auttaa suuresti kehittämään React Native sovellusta toimintojen puolesta paremmaksi. Ionicin dokumentaatiosta ei vastaavanlaisia ohjeita löydy.

React Native -kehityksessä hyötyy paljon, jos kehittäjä osaa natiivia mobiilisovelluskehitystä. Natiivimoduulien lisääminen saattaa vaatia natiivikoodiin koskemista. Moduulien asennusohjeet kertovat yleensä tarkalleen mihin tiedostoon kehittäjän tarvitsee koodia kopioida, mutta ongelmatilanteissa ilman osaamista Android- tai iOS-kehityksestä ei pärjää.

4.2 Lopputulos

4.2.1 Ionic

Ionicin käyttöä suosittelisin toimeksiantajan tarpeisiin erityisesti käyttöliittymäprototyyppien tekemiseen. Ionic antaa parhaimmillaan kehittäjälle mahdollisuuden tuoda puhelimen toimintoja ja UI-elementtejä erittäin nopeasti mobiilisovellukseen. Valmiiksi tyylitellyt UI-elementit auttavat kehittämään sovelluksen nopeasti ja Sassin avulla voi koko sovelluksen värimaailman vaihtaa mieluisekseen. Yksinkertaisia toimintoja, kuten näkymien vaihtoja, on myös helppo lisätä, mutta lähes kaikki puhelimen toiminnot, kuten esimerkiksi kamera, vaativat erillisen kirjaston. Näiden lisäksi Ionicissä on ajateltu useita asioita, jotka ovat pieniä mutta silti tärkeitä asioita mobiilisovelluskehityksessä kuten esimerkiksi sovelluksen ikoneiden ja splash screenien generointi yhden kuvan avulla.

Heikkoudet nousevatkin Ionicissa esille, kun sovellukselta vaaditaan vaativampia toimintoja tai kovempaa suorituskykyä puhelimelta. Näissä tilanteissa astuu esiin heikkoudet WebView-komponentin sisällä pyörivästä mobiilisovelluksesta. Kuten Sonny Lazuardin artikkelissa (2015), jossa vertailtiin Ionicin ja React Nativen suorituskykyä samanlaisella sovelluksella, saattoi huomata, kuinka esimerkiksi kartan käyttäminen, tuo suuria eroja sovelluksen toimivuuteen. Erot suorituskyvyssä johtuvat Ionicin käyttämästä WebView-komponentista.

4.2.2 React Native

Toimeksiantajan näkökulmasta React Native erottuu ehdottomasti edukseen. Näkisin React Nativen olevan parempi valinta lähes kaikenlaisiin projekteihin. Vaikka Reactin

komponentit hidastavat kehitystä yksinkertaisen asioiden osalta, tuo komponentit nopeaa uudelleenkäytettävyyttä, sen jälkeen kun komponentti on kerran luotu. Sen lisäksi Hot Reloading -toiminto, hyvä suorituskyky ja natiivit UI-elementit tekevät React Nativesta melko hyvän valinnan mobiilisovelluskehitykseen pelkästään yhdelle alustalle kehitettäessä. Natiiville mobiilisovelluskehitykselle React Native silti häviää, jos sovellukselta vaaditaan tarkkaa optimointia suorituskyvyn näkökulmasta.

Heikkouksina React Nativessa nousee esille eroja mobiilialustojen toimivuudessa. React Native on alun perin kehitetty iOS-laitteita ajatellen. Android-tuki saapui React Nativeen vasta puoli vuotta myöhemmin. Alustojen eroja huomaa esimerkiksi navigaatiossa, jossa iOS-laitteille on mahdollista luoda natiivi navigaatiojärjestelmä, kun taas Android-laitteille tämä ei ole mahdollista. React Nativessa ongelmaksi nousee myös pitkällä aikavälillä sen kehitystahti. Toimintoja tulee jatkuvasti lisää ja joitain vanhoja toimintoja poistetaan käytöstä, jolloin sovelluksen ylläpitämiselle saattaa tulla hieman lisätyötä.

5 POHDINTA

Tämän opinnäytetyön tavoitteena oli vertailla Ionic ja React Native -ohjelmistokehyksiä. Vertailun tavoitteena oli tuottaa toimeksiantajalle tietoa ja perusteluita siitä, minkälaisiin tilanteisiin ohjelmistokehykset sopivat. Työ pääsi siihen asetettuun tavoitteeseen, löytämällä molemmista ohjelmistokehyksistä käyttötapauksia toimeksiantajan tulevia projekteja varten.

Vertailtavista ohjelmistokehyksistä opin opinnäytetyön aikana paljon uutta tietoa. Ohjelmistokehyksien lisäksi opin paljon JavaScript-kirjastoista, joita Ionic ja React Native käyttävät. Etenkin React Nativen dokumentaatio ja opastus oli hyvin kirjoitettu. Dokumentaation selkeät esimerkit auttoivat ymmärtämään ja sisäistämään asioita nopeasti. Työn aikana haastavimpina asioina nousivat esiin erityisesti erinäiset ongelmat liittyen Ionicin ja React Nativen natiiveihin toimintoihin. Varsinkin React Nativen kolmannen osapuolen kirjastot toivat monesti erinäisiä ongelmia liittyen siihen, mitä React Nativen versiota kirjastot tukivat.

Web-kehityksessä uusia ohjelmistokehyksiä tulee jatkuvasti lisää ja niiden suosio muuttuu joka vuosi. Vaikka kehittäjän ei ole pakko tekniikoita valita suosioon perustuen, on hyvä ymmärtää mikä tekee uusista ohjelmistokehyksistä suosittuja ja miksi muut kehittäjät niitä käyttävät. Web-kehityksen mukana vaihtuvat trendit tulevat vaikuttamaan myös JavaScript-pohjaiseen alustariippumattomaan mobiilisovelluskehitykseen. Tulevaisuudessa tätä vertailua ja tämän työn aikana opittuja tietoja onkin hyvä käyttää pohjana ja hyödyntää vertaillessa uusia ohjelmistokehyksiä.

LÄHTEET

Owens, J. 2016. The state of JavaScript, Mobile Frameworks. Luettu 22.10.2016. http://stateofjs.com/2016/mobile/

Smartphone OS market share. 2016. IDC Research, Inc. Luettu 20.10.2016. http://www.idc.com/prodserv/smartphone-os-market-share.jsp

Wilken, J. 2015. Ionic in Action: Hybrid Mobile Apps with Ionic and AngularJS. Yhdysvallat: Manning Publications.

Introduction to the Camera API. 2016. Mozilla Foundation. Luettu 01.12.2016. https://developer.mozilla.org/en-US/docs/Mozilla/B2G_OS/API/Camera_API/Introduction

Saleh, H. 2014. JavaScript Mobile Application Development. Yhdistynyt kuningaskunta: Packt Publishing.

Panhale, M. 2016. Beginning Hybrid Mobile Application Development. Yhdysvallat: Apress.

Wargo, J. 2014. Apache Cordova API Cookbook. Yhdysvallat: Addison-Wesley Professional.

Ionic Documentation, CLI. 2016. Drifty Co. Luettu 09.11.2016. http://ionicframework.com/docs/cli/

Ionic Cloud. 2016. Drifty Co. Luettu 10.11.2016. https://ionic.io/cloud

AngularJS Developer Guide, Dependency Injection. 2016. Google Inc. Luettu 19.11.2016. https://docs.angularjs.org/guide/di

Tarasiewicz, P. & Böhm, R. 2014. AngularJS. Saksa: Brainy Software Inc.

Phan, H. 2015. Ionic Cookbook. Yhdistynyt kuningaskunta: Packt Publishing.

iOS Human Interface Guidelines. 2016. Apple Inc. Luettu 29.10.2016. https://developer.apple.com/ios/human-interface-guidelines/

Android Design Principles. 2016. Google Inc. Luettu 29.10.2016. https://developer.android.com/design/get-started/principles.html

Android Studio User Guide. 2016. Google Inc. Luettu 10.11.2016. https://developer.android.com/studio/run/index.html

de Sousa Antonia, C. 2015. Pro React. Yhdysvallat: Apress.

Feldman, R., Bagnardi, F., Højberg, S. & Hall, J. 2016. Developing a React Edge, Second Edition. Yhdysvallat: Bleeding Edge Press.

Eisenman, B. Learning React Native. 2015. Yhdysvallat: O'Reilly Media, Inc.

React Native Docs, NavigatorIOS. 2016. Facebook Inc. Luettu 13.11.2016. https://facebook.github.io/react-native/docs/navigatorios.html

React Native Docs, Native Modules. 2016. Facebook Inc. Luettu 03.11.2016. https://facebook.github.io/react-native/docs/native-modules-ios.html

Lazuardi, S. 2015. Ionic Framework vs React Native. Luettu 15.11.2016. https://medium.com/react-id/ionic-framework-hybrid-app-vs-react-native-4facdd93f690#.l000xxslj