HAAGA-HELIA
ammattikorkeakoulu

React Native –sovelluskehyksen ja AWS-pilvipalvelun avulla rakennettu sosiaalinen mobiilisovellus

Niko Salminen

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2018

HAAGA-HELIA
ammattikorkeakoulu

Tietojenkäsittelyn koulutusohjelma

Tiivistelmä

8.2.2018

Tekijä tai tekijät Niko Salminen	**Ryhmä tai aloitusvuosi** HETI13SIM
Opinnäytetyön nimi React Native –sovelluskehyksen ja AWS-pilvipalvelun avulla rakennettu sosiaalinen mobiilisovellus	**Sivu- ja liitesivumäärä** 21
Ohjaaja tai ohjaajat Niina Kinnunen	

Opinnäytetyön aihe on React Native –sovelluskehyksen ja AWS-pilvipalvelun avulla rakennettu sosiaalinen mobiilisovellus, jonka lähdekoodi on rakennettu JavaScript-kielellä ja on yhteinen tuettujen mobiilikäyttöjärjestelmien (iOS, Android) kesken. Aihe on tärkeä, koska käytetyt teknologiat kasvattavat jatkuvasti suosiotaan ja niiden yhteiskäytöstä ei ole saatavilla laadukasta ohjeistusta.

Työ on produktityyppinen ja sen lopputuotoksena on toimiva prototyyppi, jota voidaan käyttää sovelluksen liiketoimintaidean testaamiseen sekä jatkokehitykseen.

Työstä on rajattu ulos sovelluksen julkaisu, jatkokehitys ja kustannuslaskelmat.

Asiasanat
Mobiilikehitys, Android, iOS, Javascript, AWS

HAAGA-HELIA
University of Applied Sciences

Sisällys

1 Johdanto

Työssäni web-kehittäjänä olen päässyt usein hyödyntämään Facebookin kehittämää JavaScript-pohjaista React-käyttöliittymäkirjastoa. React on muodostunut alalla de facto -standardiksi ja sen yhteyteen on luotu valtava määrä kirjastoja jotka laajentavat sen käyttömahdollisuuksia käytännössä rajattomasti. Eräs kiinnostavimmista on Facebookin vuonna 2015 esittelemä React Native, jonka avulla natiivin mobiilisovelluksen käyttöliittymä ja logiikkakerros voidaan kirjoittaa JavaScriptillä ja React-komponenteilla.

Olen aiemmin kokeillut mobiilisovellusten kehittämistä Cordova- ja Ionic -alustalla ja todennut sen helpoksi mutta suorituskyvyltään ja käyttökokemukseltaan vaatimattomaksi ratkaisuksi. Kuultuani React Nativesta halusin päästä syventymään aiheeseen tarkemmin ja päädyinkin valitsemaan React Nativen opinnäytetyöni aiheeksi. Tavoitteenani on selvittää, miten React Native -sovelluksessa voidaan hyödyntää AWS-pilvipalveluita, joiden avulla on mahdollista rakentaa sovellus jonka tietovarasto toimii ilman dedikoiduja palvelimia täysin manageroidun DynamoDB-tietokannan sekä Cognito-autentikoinnin varassa. Lisäksi tavoitteenani on verrata React Native -kehitystä ja sillä luotua sovellusta Cordova-hybridisovellusten vastaaviin.

Sovelluksen aiheeksi valitsin ystäväni kehittämän idean sosiaalisesta keräilysovelluksesta, jossa avulla käyttäjät voivat luoda kuvakansioita sekä liittää kuviin erilaista metatietoa. Vastaavia sovelluksia on jo markkinoilla, mutta niiden toteutus on hyvin alkeellinen eikä mukana ole sosiaalista aspektia. Muista sosiaalisista kuvasovelluksista ratkaisu eroaa tarjoamalla mahdollisuuden liittää kuviin metadataa, jonka avulla voidaan mm. hakea palvelusta vastaavaa sisältöä.

Sovelluksesta on tarkoitus kehittää ensin MVP-versio käyttäjätestausta varten ja mikäli konsepti todetaan toimivaksi julkaistaan sovellus App Storessa sekä Google Play Storessa sekä jatketaan sen kehitystyötä saadun palautteen perusteella.

Opinnäytetyön keskeiset käsitteet ovat:

Mobiilikehitys	Sovellusten kehittäminen matkapuhelimiin ja tabletteihin
Android	Googlen mobiililaitteisiin kehittämä käyttöjärjestelmä

iOS	Applen iPhoneen ja iPadiin kehittämä käyttöjärjestelmä
JavaScript	Web- ja mobiilikehityksessä suosittu ohjelmointikieli
AWS	Amazon Web Services, Amazonin pilvipalvelukokonaisuus
Natiivisovellus	Mobiililaitteen pääasialliseksi tarkoitetulla kehityskielellä (esim. Java tai Swift) luotu sovellus

2 React Native yleisesti

React Native on Facebookin kehittämä avoimen lähdekoodin sovelluskehys, jonka avulla voidaan tuottaa ns. natiiveja mobiilisovelluksia käyttäen niin ikään Facebookin kehittämää erittäin suosittua JavaScript-kielellä toteutettua React-kirjastoa.

Reactin avulla on mahdollista rakentaa käyttöliittymiä deklaratiivisesti. Käyttöliittymä rakennetaan useasta sisäkkäisestä komponentista, jotka esitysvaiheessa päivitetään joko internetselaimen DOM-elementeiksi tai React Nativen tapauksessa mobiililaitteen natiivikomponenteiksi. (Facebook Inc. 2017a.)

Ennen React Nativea yleisin käytetty tapa kehittää mobiilisovelluksia JavaScript-kielellä oli ns. hybridisovellusten luominen Cordova-kehystä käyttäen. Cordovan avulla luodut sovellukset koostuvat sovelluksen sisällä ajettavan web-selaimen avulla näytettävistä HTML-sivuista. React Native eroaa hybridisovelluksista käyttämällä HTML:n sijaan natiivikomponentteja, joita lisätään, poistetaan ja muokataan Reactin ja JavaScriptin avulla. (Facebook Inc. 2017b.)

Cordova-sovellusten merkittävimpänä rajoitteena niiden käyttöliittymä koostetaan HTML-elementeistä natiivikomponenttien sijaan. Tämän seurauksena mobiililaitteen oman ulkoasun ja käyttökokemuksen mukaileminen on huomattavan työlästä. Kirjastot kuten Ionic tarjoavat kokoelman valmiita natiivitoteutusta mukailevia komponentteja, mutta niitäkin käytettäessä on vaikeaa seurata suurien käyttöjärjestelmäpäivitysten kuten iOS 7 mukanaan tuomia merkittäviä muutoksia. Mikäli sovellusta ei pidetä ajan tasalla voi käyttöjärjestelmän päivityksen jälkeen voi sovelluksen ulkoasu olla edelleen vanhan järjestelmäversion mukainen. Natiivikomponentteja käytettäessä tätä ongelmaa ei ole, vaan sovelluksen käyttöliittymän ulkoasu ja tuntuma vastaavat aina järjestelmän omia. (Stein 2017.)

React Native -sovellukset koostuvat sekä natiivikoodista että JavaScript/JSX:stä. Luodut projektit sisältävät natiivisovellukset sekä iOS- että Android-laitteille. Näiden sovellusten tehtävä on alustaa halutut natiivirajapinnat ja tarjota ne sovelluksen sisällä ajettavan JavaScript-ohjelman käyttöön. JavaScript-koodi kirjoitetaan JSX-laajennettuna Ec-

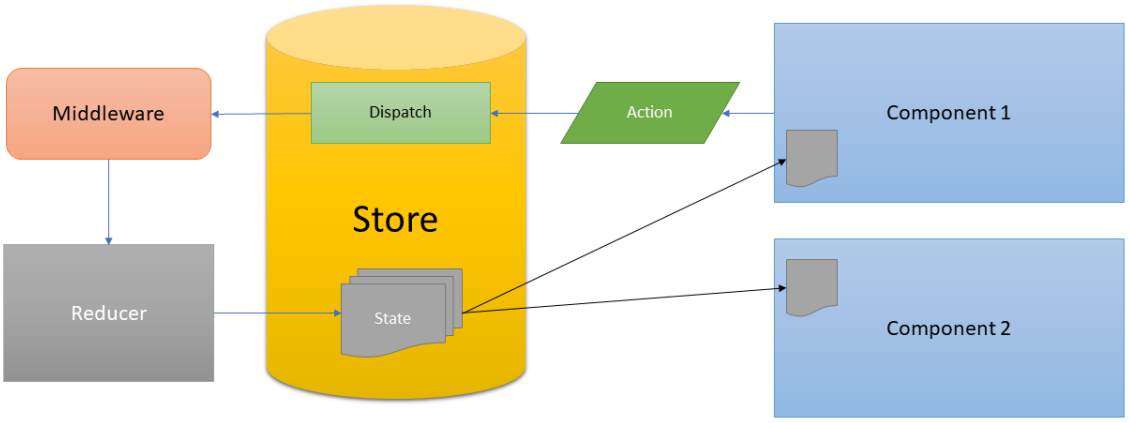
maScript2015 (aiemmalta nimeltään ES6) -koodina ja käännetään Babel-työkalulla tavalliseksi JavaScriptiksi sovelluksessa ajamista varten.

React Native –projektien aloittamisen avuksi on luotu Native Starter Kit (myöh. NSK) –työkalu, joka luo uuden React Native –projektin ja lisää sen käyttöön joukon yleisiä sen yhteydessä käytettyjä työkaluja ja kirjastoja. (GeekyAnts 2017.)

3 JSX-syntaksi ja Native Starter Kitin tarjoamat kirjastot

Facebook suosittelee React-sovellusten käyttöliittymäkerroksen toteuttamista JSX-syntaksilla. JSX vastaa muilta osin tavallista JavaScriptiä, mutta sallii XML:ää muistuttavan syntaksin upottamisen koodin sisään. (Facebook Inc. 2017c.)

Koska React keskittyy ainoastaan sovelluksen käyttöliittymäkerrokseen, käytetään sen yhteydessä erilaisia tietomallin ja sovelluksen tilan hallintaan kehitettyjä kirjastoja kuten Flux, Redux, Relay ja MobX. NSK lisää uusia projekteja luotaessa mukaan automaattisesti Redux-kirjaston. Redux eristää sovelluksen tilan yhteen keskitettyyn varastoon (store), jota voidaan muokata lähettämällä sille tapahtumia ja kuvaamalla, miten ne vaikuttavat sovelluksen tilaan. Tapahtumia kutsutaan Reduxissa nimellä Action ja ne ovat yksinkertaisia JavaScript-olioita jotka sisältävät tiedon tapahtuman tyypistä sekä mahdollisesti siihen liittyvää lisätietoa. Komponentit seuraavat varastosta saamaansa osaa sovelluksen tilasta ja lähettävät sen kautta Actioneita (kuva 1).



Kuva 1. Tiedon kulku Reduxissa

Actionien käsittelyyn Reduxissa kirjoitetaan reducer-nimisiä olioita, jotka hallinnoivat kukin tiettyä osaa tilasta ja tekevät siihen muutoksia vastaanottamiensa actionien perusteella. Ennen reducereille saapumistaan actionit kulkevat vielä ns. middleware-kerroksen läpi, jossa niille on mahdollista tehdä lisäkäsittelyä esimerkiksi asynkronisten verkkopyyntöjen suorittamiseksi.

Koska sovelluksen logiikka koostuu JavaScriptistä eikä natiivikoodista, on sovellus tietyissä tilanteissa mahdollista päivittää luomatta uutta versiota sovelluskauppoihin. NSK asentaa kehitysympäristöön Microsoftin kehittämän CodePushin, jonka avulla päivitykset on mahdollista ajaa langattomasti ja automatisoidusti kaikkien käyttäjien mobiililaitteisiin. CodePushin kehitys itsenäisenä palveluna on lopetettu ja toiminnallisuus siirretty osaksi Microsoftin App Center -kehitysympäristöä. (Microsoft Corporation 2017.)

Sovelluksen sisäisen navigoinnin toteuttamiseksi NSK tarjoaa React Navigation -kirjaston, jonka avulla voidaan sovelluksen näkymät jakaa erilaisiin navigoitaviin kokonaisuuksiin. Kirjastossa on suora tuki Reduxille, mutta NSK:ssa navigaatiota käytetään oletusarvoisesti ilman sitä.

Sovelluksen kehityksessä esiintyvien regressioiden välttämiseksi on syytä pitää automatisoidut testit ajan tasalla. NSK:n mukana asentuu Jest, yksi suosituimmista React-testikirjastoista. Facebookin kehittämä Jest sisältää tarvittavat työkalut yksikkötestaukseen, mock-testiolioihin, testien kattavuuden mittaamiseen sekä muihin testaamisessa tarvittaviin toimintoihin. (Facebook Inc. 2017d.)

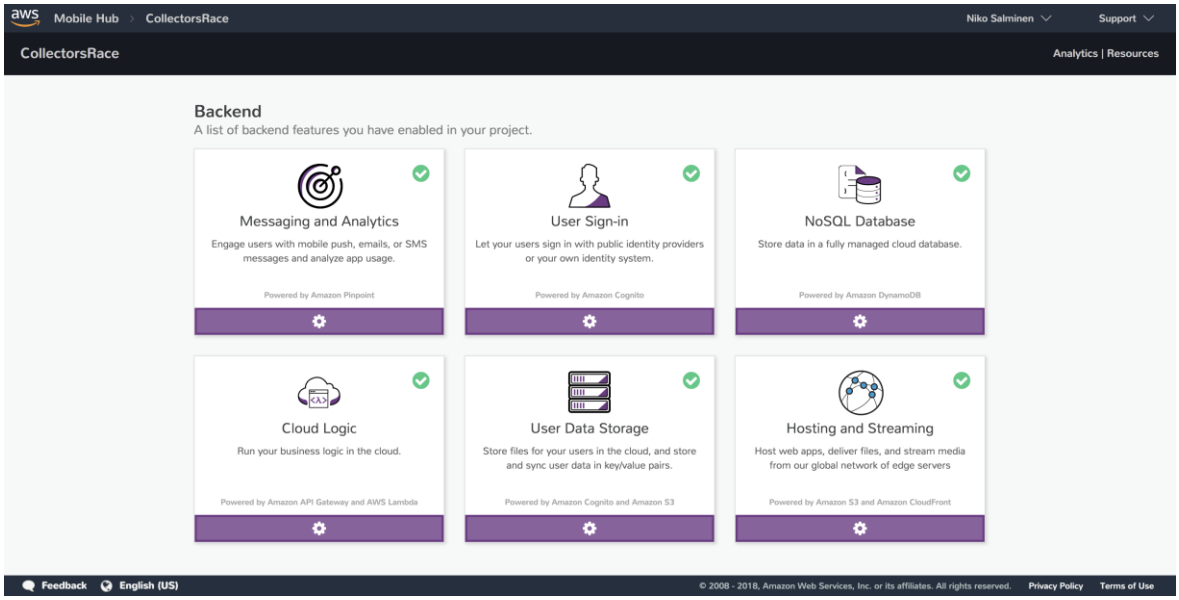
4 AWS sovelluksen tietovarastona

Amazonin AWS-palvelu sisältää useita erilaisia toiminnallisuuksia, joita voi ostaa ns. pilvipalveluina eli käytön mukaan veloitettuna. AWS sallii sovelluksen käyttäjien luoman sisällön jakamisen ja käyttämisen kustannustehokkaasti, skaalautuvasti ja maailmanlaajuisesti.

AWS:n tarjoamista palveluista sovelluksessa hyödynnetään DynamoDB-tietokantaa, S3-tiedostovarastoa, Cognito-autentikointityökalua, Lambda-funktioita ja CloudFront-välimuistipalvelua. Cognito-autentikointi suoritetaan sovelluksessa käyttäen Facebookia identiteettitarjoajana, jolloin käyttäjät voidaan todentaa Facebook-tunnuksien avulla.

Koska AWS:n palveluista laskutetaan käyttöasteen mukaan, on niiden käyttö etenkin sovelluksen kehitysvaiheessa joko erittäin halpaa tai yleensä jopa ilmaista. Cognito-tunnistautuminen on ilmaista 50 tuhannen aktiivisen kuukausikäyttäjän rajaan asti, Lambda-funktioita voi ajaa kuukausittain miljoona kertaa veloituksetta ja DynamoDB tarjoaa 25 gigatavua ilmaista tietokantatilaa sekä kapasiteettia noin 200 miljoonan luku- ja kirjoituspyynnön kuukausittaiseen käsittelyyn. AWS S3 on pienellä käyttötasolla ilmainen vain ensimmäisen vuoden ajan, mutta tämänkin jälkeen tiedostojen tallentaminen maksaa vain $0.0245 gigatavua kohden kuukaudessa. (Amazon Web Services 2018.)

Amazonin pilvipalveluihin integroituminen on helpointa tehdä käyttämällä AWS Mobile Hub -palvelua (kuva 2), joka on mobiilisovelluksille optimoitu palvelukokonaisuus. Mobile Hub tarjoaa keskitetyn ja automatisoidun tavan luoda mobiilisovelluksen tarvitsemat palvelut ja ladata niiden käyttöön tarvittavat asetustiedostot. Palveluihin kytkeytyminen sovelluskoodista on tehtävä itse, mutta sen tekemiseen voi hyödyntää toista React Native -aloitusprojektia: Amazonin julkaisemaa AWS Mobile React Native Starter Appia. (Hall & Threlkeld 2017.)



Kuva 2. Kuvaruutukaappaus AWS Mobile Hub -palvelusta

AWS Mobile React Native Starter App (AMRNSA) sisältää osin päällekkäisiä toiminnallisuuksia NSK:n kanssa, mutta suurin osa tarvittavista ominaisuuksista on vain jommassakummassa (taulukko 1).

Taulukko 1. Native Starter Kitin ja AWS Mobile React Native Starter Appin kattamat ominaisuudet

Ominaisuus	**NSK**	**AMRNSA**
React Native	X	X
React Navigation	X	X
Jest	X	X
ESLint	X	X
Redux	X	
NativeBase	X	
CodePush	X	
EasyGrid	X	
Amazon Cognito		X
AWS Lambda		X
Amazon S3		X
Amazon DynamoDB		X

Kirjautumiseen sekä tietojen siirtämiseen mobiililaitteen ja AWS:n välillä tarvitaan asynkronisia verkkopyyntöjä. Asynkronisuus tarkoittaa, että sovellus voi suorittaa muita toimintoja sillä aikaa kun pyyntöä käsitellään. Reactia ja Reduxia käytettäessä yleinen tapa toteuttaa asynkroniset pyynnöt on tehdä niistä ns. thunkkeja, joka tarkoittaa myöhemmin suoritettavia funktioita. Tällöin asynkronisten kutsujen käsittelyyn tarvittava logiikka sisällytetään Redux-actioneihin, jolloin erillinen Redux Thunk -niminen mid-

dleware käsittelee ne ja lähettää käsitellyn datan perusteella lisää actioneita. (Abramov 2017.)

5 Applikaatio

Applikaation tavoite on tarjota sosiaalinen alusta erilaisten asioiden keräilyyn. Applikaatio on tärkeää tehdä, koska markkinoilla ei tällä hetkellä ole saatavilla vastaavan toiminnallisuuden tarjoavia sovelluksia. Keräilemiseen erikoistuneita sovelluksia on olemassa vain kourallinen, eivätkä ne tarjoa kuvan ja nimen lisäksi keräiltävien asioiden tarkempaa luettelointia esimerkiksi iän, värin tai muun perusteella. Lisäksi kaikista tarjolla olevista sovelluksista puuttuvat sosiaaliset toiminnot.

Produktin kohderyhmä ovat JavaScript-kieltä työssään käyttävät web- ja mobiilikehittäjät. Kohderyhmällä on ymmärrystä React-kirjaston sekä mobiilisovellusten kehittämisen perusteista.

Produktin tuotanto jakautuu kahteen vaiheeseen: suunnittelu- ja toteutusvaihe.

Suunnitteluvaiheessa kerätään lista niistä toiminnallisuuksista, jotka sovelluksen on toteutettava toimiakseen liikeidean testaamisen työkaluna. Toiminnallisuudet kerätään projektin backlogille toteutusvaihetta varten.

Toteutusvaiheessa sovelluksen alusta luodaan NSK-työkalun avulla, tarvittavat palvelut perustetaan AWS:ään ja projektin backlogin sisältämät toiminnalliset vaatimukset toteutetaan. Toteutuskielenä toimii JavaScript/JSX.

5.1 Suunnittelu

Toteutettava sovellus on sosiaalinen palvelu, jossa käyttäjät voivat luoda virtuaalisia kokoelmia omistamistaan tai näkemistään asioista. Keräiltävistä kohteista kerätään nimi, kuva sekä kokoelman tyypistä riippuen vaihteleva määrä muita tietoja, esimerkiksi auto- ja kerätessä merkki, malli sekä moottorin tyyppi.

Erottuakseen muista saatavilla olevista keräilysovelluksista sovellukseen toteutetaan sosiaalinen ulottuvuus tarjoamalla käyttäjille mahdollisuus seurata toisia käyttäjiä ja heidän kokoelmiaan sekä ”tykätä” ja kommentoida kokoelmia ja niiden sisältämiä kohteita.

Mainitut vaatimukset sisältävä MVP (Minimum Viable Product) -toteutus voidaan toteuttaa taulukossa 2 esitellyillä käyttötapauksilla.

Taulukko 2. Sovellukset MVP-vaiheen käyttötapaukset

Koodi	**Käyttötapaus**
UC-1	Käyttäjä voi rekisteröityä palveluun sosiaalisen median tunnuksilla
UC-2	Käyttäjä voi luoda uuden kokoelman
UC-3	Käyttäjä voi lisätä uuden kohteen kokoelmaan
UC-4	Käyttäjä voi muokata kokoelmansa tietoja
UC-5	Käyttäjä voi muokata kokoelman kohteita
UC-6	Käyttäjä voi poistaa kokoelmansa
UC-7	Käyttäjä voi poistaa kohteen kokoelmastaan
UC-8	Käyttäjä voi selata kokoelmiaan
UC-9	Käyttäjä voi selata kokoelmiensa sisältöä
UC-10	Käyttäjä voi listata palveluun rekisteröityneet FB-ystävänsä
UC-11	Käyttäjä voi katsoa toisen käyttäjän profiilia
UC-12	Käyttäjä voi seurata toisia käyttäjiä
UC-13	Käyttäjä voi tykätä toisten käyttäjiä kokoelmista ja kohteista
UC-14	Käyttäjä voi kommentoida kokoelmia ja kohteita
UC-15	Käyttäjä saa ilmoituksen saamistaan kommenteista ja tykkäyksistä

5.2 Toteutus

5.2.1 Tarvittavien ohjelmistojen ja kirjastojen asennus

Sovelluksen luomisen apuna käytin GeekyAnts -yhtiön kehittämää Native Starter Kit (NSK) -työkalua. NSK on komentorivityökalu, joka luo valmiin React Native -sovellusrungon kehitystyön aloittamisen nopeuttamiseksi.

Koska React on JavaScript-pohjainen alusta, vaatii sillä kehittäminen Node.js -ajoympäristön. Kehitystyössä käytettyyn macOS -ympäristöön sen asentaminen tehtiin käyttämällä suosittua Homebrew-asennustyökalua. Node.js:n yhteydessä asennetaan automaattisesti myös pakettienhallintatyökalu NPM.

Node.js -asennuksen jälkeen käytin NPM:ää Native Starter Kitin asentamiseen. Asennuksen jälkeen Native Starter Kit voitiin ajaa, jolloin se loi uuden sovellusprojektin ja latasi sekä asensi kaikki siihen tarvittavat työkalut ja kirjastot.

Kun sovellusprojekti oli luotu, asensin Applen XCode-kehitysympäristö joka tarvitaan iOS-sovellusten kääntämiseen ja testaamiseen iPhone- ja iPad -laitteilla sekä simulaattorilla. XCode ladataan Applen App Store -sovelluskaupasta. Asentamiseen oikeaan laitteeseen tarvitaan maksullinen kehittäjälisenssi, mutta tämän tuotoksen toimivuuden tarkistamiseen ja kehittämiseen riittää simulaattori.

Android-laitteisiin asentaminen vaatii Android Studio -kehitysympäristön, jonka latasin ja asensin Androidin internetsivuilta. Lisäksi tarvittiin SDK Manager-sovellus, jonka avulla voidaan ladata jokaiselle Android-käyttöjärjestelmäversiolle kohdennetun kehityspaketin (SDK, Software Development Kit). Android-laitteisiin voi asentaa omia sovelluksia USB-johdon avulla ilman kehittäjälisenssiä, joten pystyin käyttämään omaa Android-puhelintani sovelluksen testaamiseen oikealla laitteella.

5.2.2 Applikaation tekninen toteutus

Sovelluksessa tarvittavan Facebook-autentikoinnin voi toteuttaa joko JavaScript-koodilla tai hyödyntämällä laitteen natiivitoteutusta. Natiivitoteutus on vaihtoehdoista parempi, koska se tarjoaa käyttäjälle saumattomamman käyttökokemuksen. Natiivitoteutus on saavutettavissa Facebookin oman React Native FBSDK-kirjaston avulla. Sen hyödyntäminen vaatii muutoksia sekä Android Studio -että XCode-projekteihin, mutta tarvittavat muutokset oli kuvattu projektin GitHub-sivulla hyvin. Muutosten jälkeen FBSDK:n kautta autentikointi tuotti tunnistautumistokenin AWS Cognito -palvelun kautta käytettäväksi.

AWS:n starter-projekti ei ole rakennettu Reduxin päälle, joten en voinut kopioida siinä käytettyä tunnistautumislogiikkaa sellaisenaan. Seurasin esimerkkikoodissa käytettyä rakennetta ja muutin sen Redux-kirjaston kanssa yhteensopivaksi hajauttamalla erilliset omat omiksi Redux-actioneikseen jotka ajetaan asynkronisesti hyödyntäen Redux Thunk -middlewarea.

Kirjauduin AWS:n Mobile Hub -palveluun ja kytkin päälle tarvitsemani palvelut. Kuvasin sovelluksen tarvitseman tietorakenteen DynamoDB-palveluun ja listasin sovelluk-

sen tarvitsemat API-kutsut API Gateway-palveluun. Kytkin myös päälle Cognito-palvelun ja valitsin että sovellukseen kirjaudutaan Facebook-tunnuksilla. Loin Facebookiin sovelluksen jonka kautta tunnistautuminen tapahtuu.

Seuraavaksi loin AWS:n esimerkkikoodin perusteella backend-koodin Node.js:n Express-kirjastoa hyödyntäen. Expressiin kirjoitetaan middleware-nimisiä funktioita joilla sovellus kuuntelee tiettyjä http-metodeja (get, post, put, delete, jne) tietyissä osoitteessa. Expressin middlewaret eivät liity Reduxin middlewareihin millään tavalla. AWS:n starter kitin mukana tulevalla työkalulla sain paketoitua backend-koodin zip-paketiksi, jonka pystyi lähettämään suoraan AWS:n Lambda-palveluun ajettavaksi. AWS Mobile Hub tarjosi Lambda-paketin lähettämisen jälkeen asetustiedoston, jonka lisäämällä React Native -projektiini sain Mobile Hubista päälle kytkemäni AWS:n palvelut käyttöön sovelluksesta käsin.

Sovelluksen eri näkymät koostin erillisistä komponenteista, jotka sisälsivät Native Base-kirjaston tarjoamia käyttöliittymäkomponentteja. Sovelluksen toiminnallisuus tapahtuu käyttöliittymäelementtien lähettämien Redux-actionien kautta. Sovelluksen navigoinnin toteutin käyttämällä NSK:n mukana tullutta StackNavigator -toiminnallisuutta, jossa uuteen näkymään siirtyminen tapahtuu mobiililaitteiden natiivisovelluksista tutulla animoidulla korttisiirtymällä. Sen lisäksi lisäsin käyttökokemuksen sujuvoittamiseksi sovellukseen DrawerNavigator -komponentin, joka on vasemmasta reunasta esiin liukuva navigaatio ja tarjoaa linkkejä suoraan sovelluksen yleisimpiin toiminnallisuuksiin.

Lisäsin sovellukseen mahdollisuuden lisätä kuvia joko kameralta tai mobiililaitteen tallennetuista kuvista. Molempien lisääminen vaatii muutoksia sovelluksen natiivikoodiin sekä uusien kirjastojen tuonnin projektiin laitteen natiiviominaisuuksien hyödyntämiseksi. iOS-laitteilla kehitystyötä hidasti se, että puhelinsimulaattori ei tarjoa minkäänlaista kameratoiminnallisuutta edes emuloituna. Sen sijaan etukäteen tallennettujen kuvien valitsemiselle simulaattorissa on täysi tuki. Projektin myöhemmässä vaiheessa löysin React Native Image Picker -kirjaston, joka tarjoaa yksinkertaisen tavan valita kuvia puhelimen albumeista tai kamerasta hyödyntäen natiiveja käyttöliittymäelementtejä sekä iOS:lla että Androidilla.

NSK:n esimerkkitoteutuksessa sisäänkirjautuminen on sijoitettu omaksi sivukseen navigaatioon. Tämä valinta ei ole mielestäni perusteltu, koska kirjautumissivulle on tällöin mahdollista palata sovelluksen tai puhelimen back-napin avulla, jolloin käyttäjä saattaa päätyä jo kirjautuneena kirjautumissivulle jolla voi olla virheellistä tai harhaanjohtavaa sisältöä. Lisäksi kirjautumisen vanhentumisen jälkeen on mahdollista päästä lukemaan kirjautumisen vaativaa tietoa. Poistin kirjautumissivun navigaatiosta ja loin sen tilalle uuden LoginGate-nimisen komponentin. Komponentille annetaan parametreiksi kaksi komponenttia, joista yksi näytetään kirjautuneille ja toinen kirjautumattomille käyttäjille. Komponentti seuraa kirjautumisen tilaa Reduxin storen kautta ja siirtää käyttäjän välittömästi näkymästä toiseen tilan muuttuessa.

Kuvien tallentamiseen käytetyn AWS S3 -palvelun käyttöönotto oli yksi projektin haastavimmista vaiheista. Palvelu ei suostunut hyväksymään sille parametreina annettuja tunnistautumistietoja, vaan se kelpuutti ainoastaan oman AWS-namespacensa kautta tehdyn kirjautumisen tiedot. Olin käyttänyt kirjautumisessa React Native -optimoituja versioita Cognito-kirjastosta, jotka eivät tallentaneet tunnistautumistietoja samaan paikkaan mistä S3 niitä yritti etsiä. Ongelma korjaantui käyttämällä Cognitosta geneeristä JavaScript-kirjastoa React Native -version sijaan.

React Native Image Picker palautti viittaukset lähetettäviin kuvatiedostoihin `file://` -alkuisina osoitteina, joita AWS:n esimerkkisovelluksen käyttämä React Native Fetch Blob -kirjasto ei jostain syystä osannut lukea. Muutin sovelluksen käyttämään toista vastaavaa kirjastoa nimeltä React Native FS, jolta kuvien lukeminen onnistui kamerasta ja kuvagalleriasta niin iPhone-simulaattorilla kuin oikealla Android-laitteellakin.

Nykyisten puhelinten kamerat tuottavat pikselimäärältään niin suuria kuvia, että niiden kokoa oli pakko pienentää - 6 megatavua on aivan liian suuri tiedostokoko mobiiliyhteyden yli lähetettäväksi. Kuvien pienentämiseen käytin React Native Image Resizer -kirjastoa. Rajasin sovelluksen kautta lähetettävien kuvien maksimikooksi 2048 pikseliä pidemmältä sivultaan. Koko valikoitui iPad-tablettien näytön resoluution mukaan, koska halusin säästää mahdollisuuden näyttää kuvia tabletillakin koko ruudun kokoisena. Lisäksi kuvista luodaan listauksissa näytettäväksi thumbnail-versio 256 x 256 pikselin kokoisena sekä heikommalla pakkauslaadulla.

Sovellusta käytettäessä käyttäjälle esitetään useassa näkymässä lukuisia kuvia. Niiden lataaminen joka kerta verkosta hidastaisi sovelluksen käyttöä, tuhlaisi laitteen tiedonsiirtokaistaa ja aiheuttaisi turhia kustannuksia AWS:n S3-latauksista sekä tiedonsiirrosta. Näiden ongelmien kiertämiseksi tarvitsin ratkaisun, jolla kuvat voi tallentaa sovelluksen välimuistiin ja ladata ne sieltä verkkoyhteyden sijaan. React Nativessa itsessään on ominaisuus kuvien välimuistiin pakottamiseksi, mutta kirjoitushetkellä ominaisuus on tuettu ainoastaan iOS-laitteilla. Lisäsin projektiin kirjaston nimeltä React Native Image Cache, joka hoitaa kuvien tallentamisen ja noutamisen välimuistista automaattisesti.

5.3 Tuotos

Projektin lopputuloksena syntyi sovellus sekä asetustiedostot joiden avulla sovelluksen vaatimat taustapalvelut saa luotua AWS-palveluun. Sovellus toteuttaa sille määritellyt MVP-tavoitteet lukuunottamatta käyttäjien keskenäistä vuorovaikutusta (käyttötapaukset UC-10 – UC-15), jonka toteutus jätettiin myöhempiin iteraatioihin sen vaatiman työn suuruudesta johtuen.

Sovellus toimii sekä iOS- että Android-laitteilla ja tuntuu sekä näyttää molemmissa natiivilta sovellukselta.

Sovelluksella on mahdollista kirjautua sisään Facebook-tunnuksilla. Mikäli käyttäjä on aiemmin luonut kokoelmia sovellukseen toisella laitteella synkronoituvat kokoelmat kirjautumisen yhteydessä myös uuteen laitteeseen. Kirjautumista ei tarvitse tehdä uudestaan jatkettaessa sovelluksen käyttöä myöhemmin.

Sovelluksen avulla voidaan luoda uusia kokoelmia sekä muokata tai poistaa olemassa olevia. Muutokset päivitetään AWS DynamoDB-tietokantaan.

Kokoelmiin on mahdollista lisätä kuvia sekä antaa näille kokoelman asetuksissa säädetty määrä metatietoa. MVP-vaiheessa ainoa tuettu tieto on kuvan nimi, mutta tulevilla kehityskierroksilla tullaan lisäämään kokoelmiin mahdollisuus luoda vapaasti käyttäjän

haluama määrä tietokenttiä. Kuvat lähetetään AWS S3 -tallennuspalveluun ja niiden tiedot DynamoDB-tietokantaan.

6 Pohdinta

Tavoitteenani oli selvittää, miten React Native -sovelluksessa voidaan hyödyntää AWS-pilvipalveluita. Havaintojeni perusteella hyödyntäminen on nykyisillä työvälineillä ja esimerkeillä helppoa, suurimmaksi osaksi projektin aikana julkaistun AWS Mobile React Native Starter Appin ansiosta. Mikäli React Native -sovelluksen halutaan hyödyntävän myös Reduxia tai muita vastaavia kirjastoja, täytyy AWS-palveluiden käyttötapaan tehdä muutoksia haluttujen toimintatapojen tukemiseksi. Vaikka Amazonin kehittämää esimerkkiprojektia ei käytettäisi, on mobiilisovelluksissa yleisimmin käytetyt palvelut yhteen kokoava AWS Mobile Hub yksinäänkin huomattavasti kehitystyötä nopeuttava lisäpalvelu.

Toinen tavoitteeni oli verrata React Native -kehitystä ja sillä luotua sovellusta Cordova-hybridisovellusten vastaaviin. Verrattaessa React Native -sovellusta Cordovalla toteutettuun hybridisovellukseen on havaittavissa selkeä ero sovelluksen suorituskyvyssä ja käyttökokemuksessa ensin mainitun hyväksi, etenkin yritettäessä mukailla käyttöjärjestelmän standardia ulkoasua. Sovelluksen toiminnallisuuden jakaminen loogisiin kokonaisuuksiin onnistuu myös mielestäni huomattavasti helpommin React Native -alustalla kuin suosituimman Cordova-pohjaisen kehitysympäristö Ionicin käyttämässä Angularissa.

Tavoitteiden saavuttaminen onnistui mielestäni hyvin, sillä opinnäytetyön aikana sain hyvin kattavan käsityksen käytettävissä olevista palveluista ja niiden tämänhetkisestä tilasta. Myös kehitystyöstä ja sen helppoudesta verrattuna Ionic-kehitysalustaan sain kerättyä paljon tietoa.

Web-kehittäjänä arvostan myös React Native -kehityksen helppoutta verrattuna natiivikehitykseen Javalla tai Swiftillä. Uusia opeteltavia asioita on huomattavasti vähemmän ja opittuja asioita voi hyödyntää pääosin molemmilla mobiilialustoilla.

AWS:n hyödyntäminen mobiilisovelluksen taustajärjestelmänä vaikuttaa erittäin toimivalta ratkaisulta. AWS Mobile Hub -palvelu yksinkertaistaa tarvittavien palvelujen luo-

mista ja tarjoaa myös mahdollisuuden kopioida ympäristö helposti muille alueille ilman tarvetta opiskella monimutkaisen CloudFormation -palvelun käyttöä.

React Nativen uutuus ja JavaScript-maailman nopeat ekosysteemimuutokset näkyvät kehitystyössä käytettyjen kirjastojen ja työkalujen nopeana vanhentumisena. Koko projektin perustana käytetty Native Starter Kit ehti vanhentua projektin aikana kehittäjän lopetettua sen aktiivisen ylläpitämisen uudemman React Native Seed -työkalun otettua sen paikan. Lisäksi Microsoftin CodePush itsenäisenä palveluna lakkautettiin projektin aikana ja siirrettiin osaksi App Center -kokonaisuutta, johon oli onneksi helppoa päivittää. Aloitusprojektia ei ole järkevää vaihtaa sovelluskehityksen päästyä vauhtiin, mutta tulevissa projekteissa kannattaa käyttää ajanmukaisimpia tarjolla olevia työkaluja.

Aihetta olisi mahdollista tutkia lisää vertailemalla Native Starter Kitin seuraajaksi luotua React Native Seediä muihin tarjolla oleviin aloitusprojekteihin ja kehitysalustoihin kuten Xamarin ja uusin Ionic. Lisäksi muissa pilvipalveluissa kuten Microsoft Azure, IBM Cloud ja Google Cloud olisi mahdollisuuksia tehdä vertailevaa tutkimusta niiden käyttökelpoisuudesta vastaavaan tarkoitukseen sekä integroitavuudesta kehitysalustaan.

Opinnäytetyön teko oli paikoin työlästä johtuen valmiiden ratkaisujen ja ohjeiden vähäisestä tarjonnasta. Suurin osa opinnäytetyön tekoon käytetystä ajasta kului siihen kuuluvien osien ja kirjastojen integroinnista ja mukauttamisesta. Osa vastaan tulleista ongelmista pysäytti työn etenemisen välillä pitkäksikin aikaa, koska ratkaisuja juuri vastaavaan tilanteeseen ei ollut löytyä. Useimmiten apu löytyi sovelluksessa käytettyjen kirjastojen GitHub-sivujen bugiraporteista ja niihin liittyvistä keskusteluista.

Opinnäytetyötä tehdessä opin käyttämään React Native -työkaluja paljon aiempaa tehokkaammin. Muita tärkeitä uusia asioita joita opin projektin aikana olivat mm. AWS Mobile Hub -palvelun olemassaolo, Cognito-palvelun käyttö ja natiivikirjastojen lisääminen React Native -projektiin. Myös AWS:n DynamoDB ja S3 -palvelut tulivat aiempaa tutummaksi.

Ehkä tärkein opittu asia projektin aikana oli, että uusia työkaluja käytettäessä kannattaa varautua niissä tapahtuviin suuriinkin muutoksiin. Mitä vähemmän integraatioita pro-

jektiin joudutaan tekemään, sitä todennäköisemmin selvitään pienillä muutoksilla mikäli jokin käytetyistä kirjastoista tai palveluista joudutaan vaihtamaan projektin aikana toiseen.

Lähteet

Abramov 2017. Redux Thunk. Luettavissa: https://github.com/gaearon/redux-thunk. Luettu: 8.12.2017.

Amazon Web Services 2018. AWS Free Tier. Luettavissa: https://aws.amazon.com/free. Luettu: 2.2.2018.

Facebook Inc. 2017a. React – A JavaScript library for building user interfaces. Luettavissa: https://reactjs.org/. Luettu: 6.11.2017.

Facebook Inc. 2017b. React Native - A framework for building native apps using React. Luettavissa: https://facebook.github.io/react-native/. Luettu: 6.11.2017.

Facebook Inc. 2017c. Introducing JSX. Luettavissa: https://reactjs.org/docs/introducing-jsx.html. Luettu: 8.12.2017.

Facebook Inc. 2017d. Jest – Delightful JavaScript testing. Luettavissa: https://facebook.github.io/jest/. Luettu: 10.12.2017.

GeekyAnts 2017. Native Starter Kit. Luettavissa: https://github.com/start-react/native-starter-kit. Luettu: 6.11.2017.

Hall, M. & Threlkeld, R. Announcing: React Native Starter Project with One-Click AWS Deployment and Serverless Infrastructure. Luettavissa: https://aws.amazon.com/blogs/mobile/announcing-react-native-starter-project-with-one-click-aws-deployment-and-serverless-infrastructure/. Luettu: 6.11.2017.

Microsoft Corporation 2017. Microsoft Code Push. Luettavissa: https://github.com/Microsoft/code-push. Luettu: 8.12.2017.

Stein 2017. The Duel: React Native vs. Cordova. Luettavissa: https://www.toptal.com/mobile/comparing-react-native-to-cordova. Luettu: 9.12.2017.