Tampereen ammattikorkeakoulu

Expo ja laajennuksen tuottaminen mobiilisovellukseen

Lauri Laiho

OPINNÄYTETYÖ
Toukokuu 2022

Tietojenkäsittely
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Ohjelmistotuotanto

LAIHO, LAURI:
Expo ja laajennuksen tuottaminen mobiilisovellukseen

Opinnäytetyö 32 sivua, joista liitteitä 0 sivua
Toukokuu 2022

Opinnäytetyön tarkoituksena oli kehittää toimeksiantajan verkosto- ja yhteisöpalvelun mobiilisovellukseen uusi laajennus. Laajennuksen tavoitteena on tarjota urheilijoille ja valmentajille ominaisuuksia helpottamaan aikataulujen ja harjoitusten luomista ja hallintaa sekä tilastojen seuraamista. Lisäksi työssä tutustutaan laajennuksiin mobiilisovelluksissa ja yleisiin asiakohtiin, joita on hyvä huomioida uutta laajennusta kehittäessä.

Dynaamisten moduulien tai laajennuksien toteuttaminen on etenkin iOS-järjestelmille vaivalloista, koska säännöt rajoittavat niiden latauksia. Näin ollen vaihtoehtoja laajennuksen toteuttamiseksi olivat sen integroiminen palvelun jo olemassa olevaan mobiilisovellukseen tai uuden ja erillisen sovelluksen kehittäminen. Konseptin kiteytyessä se päätettiin integroida palvelun sovellukseen ylläpidon helpottamiseksi, vaikkakin molemmissa vaihtoehdoissa oli hyviä puolia. React Native -pohjainen Expo-ohjelmistokehys mahdollisti helpon alustariippumattoman kehityksen, ja sen avulla laajennuksen kehittäminen alkoi ja eteni nopeasti ja yksinkertaisesti.

Työn keskeisenä tuotoksena syntyi urheiluteemaisen laajennuksen kehitysversio. Vaikka laajennus ei saavuttanut vielä julkaistavaa versioita, sen idea ja konsepti todettiin hyväksi ja toteutettavaksi. Kun kehitys eteni, tarvittavien ominaisuuksien määrä myös kasvoi jonkin verran yhteistyökumppaneilta saadun tiedon mukaisesti. Jatkokehitystä varten projektissa ratkaistiin monia ongelmakohtia ja mahdollisuudet uusien ja kehitettävien ominaisuuksien osalta ovat lupaavia.

Asiasanat: Expo, laajennus, React Native, mobiilisovellus

ABSTRACT

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

LAIHO, LAURI:
Expo and Creating a Plugin for a Mobile Application

Bachelor's thesis 32 pages, appendices 0 pages
May 2022

The purpose of this thesis was to develop a new plugin for the client's community management service's mobile application. The goal of the plugin is to offer new features for athletes and coaches to aid with the creation and management timetables, goals, and training sessions as well as monitoring statistics. In addition, the thesis introduces plugins in mobile applications and general issues that are important to consider when developing a new plugin.

Implementing dynamic modules or plugins is cumbersome, especially for iOS systems, due to terms of service restricting their upload. Hence the options for the plugin's implementation were either to integrate it to the already existing app or to develop it as a new and separate application. As the concept became clear, it was decided to do the integration to the existing app to ease its maintenance. React Native based Expo framework made cross-platform development possible and made it easy to start developing the plugin.

A developmental version of the sports-themed plugin was made as the main result of the thesis. Although no publishable version was reached yet, the concept of the plugin was found to be implementable. The number of features also increased as the development progressed, based on feedback received. Many of the problems were solved, and possibilities regarding future development, new features and systems are promising.

Key words: Expo, plugin, React Native, mobile app

SISÄLLYS

LYHENTEET JA TERMIT

Android Studio	Ohjelmointiympäristö, jolla voi kehittää natiiveja Android-sovelluksia
CSS	Cascading Style Sheets, tekniikka, jolla usein toteutetaan tyyliohjeita.
Frontend	Sovelluksen näkyvät osat, usein käyttöliittymä.
Hello World	Yksinkertainen viesti, jota käytetään usein tapana testata kehitysympäristön toimivuutta projektin alussa.
HTML	Hypertext Markup Language, merkintäkieli, tunnetaan tekniikkana, johon suurin osa web-sivuista perustuu.
Java	Ohjelmointikieli, jota voi käyttää Android-natiivisovellusten kehittämiseen.
JSX	JavaScript Syntax Extension, tekniikka Reactissa, joka helpottaa komponenttien renderöinnin syntaksia.
Mac	Applen kehittämä tietokonemerkki, joka käyttää omaa käyttöjärjestelmäänsä.
Ngrok	Tunnelointipalvelu, joka yhdistää etäverkostoja toisiinsa.
QR-koodi	Quick Response -koodi, kuviollinen koodi, joka sisältää tietoa.
SPA	Single-Page Application, yhdelle sivulle toteutettu web-sovellus.
Swift	Ohjelmointikieli, jota voi käyttää iOS-natiivisovellusten kehittämiseen.
TypeScript	Vahvasti tyypitetty muunnos JavaScriptistä.
XCode	Ohjelmointiympäristö, jolla voi kehittää natiiveja iOS-sovelluksia.

1 JOHDANTO

Älypuhelimet ovat nykyään tärkeä osa digitalisoitunutta maailmaa. Vuonna 2020 arviolta melkein neljä viidestä ihmisestä maailmanlaajuisesti omistaa älypuhelimen. (O'Dea, 2021) Ihmiset käyttävät älypuhelimiaan erilaisiin tarkoituksiin viestinnästä viihteeseen tai tiedon etsimisestä pankissa asioimiseen. Kaikki nämä toiminnot ovat erilaisten palvelujen kautta helposti käyttäjien saatavilla. Usein näillä palveluilla on myös omat mobiilisovellukset, jotka tarjoavat yrityksen palvelut helposti käyttäjille taskuun mahtuvalla tavalla. Kun kyseessä on lisäksi niinkin isot markkinat kuin neljä viidestä maailman ihmisistä, niin ei ole ihme, että mobiilialustojen ja -sovellusten suosio on räjähtänyt viime vuosikymmenen aikana.

Sovellusten tarkoituksena on toteuttaa tietty joukko toimintoja ja ominaisuuksia, joita käyttäjät voivat hyödyntää. Usein nämä ominaisuudet ovat melko tarkoin kohdennettu tietyn tavoitteen, tehtävän tai kokonaisuuden suorittamiseen, hallitsemiseen tai seuraamiseen. Alkuperäinen käyttökohde tai visio määrittelee pitkälti, millaisia toimintoja sovellus tarjoaa sen käyttötarkoituksen saavuttamiseksi. Esimerkiksi viestisovellus tarjoaa mahdollisuuden lähettää viestejä muille käyttäjille, mutta ei mahdollista kuvien muokkaamista. Koska kehitysresurssit ovat rajatut, on kannattavaa keskittyä piirteisiin, jotka edesauttavat ydintoiminnallisuuksien tuottamiseen ja vision toteutukseen.

Joskus sovelluksen käyttäjät tai kehittäjät huomaavat, että sovellusta tai sen toimintoja voisi hyödyntää eri tavalla kuin alun perin tarkoitettu. Vaikka potentiaalisia hyödyntämistapoja voi olla monia, kehittäjien ei kuitenkaan ole aina kannattavaa tai edes mahdollista käyttää aikaa ja resursseja näiden kehittämiseen tai tutkimiseen. Välillä uuden ja erilaisen käyttötavan huomataan kuitenkin olevan erittäin hyödyllinen ja laajentavan sovelluksen käyttötarkoitusta merkittävällä tavalla. Joissakin tapauksissa yksinkertaisten muutoksien tai lisäyksien teko mahdollistaa uuden käyttötarkoituksen. Lisäyksiä voi kuitenkin olla niin paljon, että uudistuksista muodostuu oma ja erillinen kokonaisuus, jolla on uusi käyttökohde tai kohderyhmä.

Mobiilisovellusten maailmassa uuden kokonaisuuden kehittämiselle on käytännössä kaksi vaihtoehtoa: erottaa kokonaisuus omaksi uudeksi sovellukseksi tai lisätä ominaisuudet ja toiminnot olemassa olevaan sovellukseen laajennuksena. Molemmat vaihtoehdot ovat varteenotettavia ja niissä on omat hyvät ja huonot puolensa kehityksen sekä käytön kannalta. Riippuen uuden kokonaisuuden laajuudesta tai tarvittavista teknologioista toinen vaihtoehdoista voi olla huomattavasti käytännöllisempi toteuttaa.

Opinnäytetyön tarkoituksena oli toteuttaa urheilijoille ja valmentajille kohdennettu laajennus toimeksiantajan verkostopalvelun mobiilisovellukseen hyödyntäen alustariippumattomia React Native- ja Expo-sovelluskehyksiä. Lisäksi työn tavoitteena on tutkia ja selvittää laajennuksiin liittyviä ongelmakohtia sekä ratkaisuja, joita on hyvä ottaa huomioon uutta laajennusta suunniteltaessa.

2 TAUSTA

2.1 Toimeksiantajan esittely

Opinnäytetyön toimeksiantaja on helsinkiläinen IT-alan yritys Yoso Oy. Yritys on perustettu vuonna 2005 ja keskittyy IT-konsultointiin ja ohjelmistokehitykseen. Vuosien varrella yritys on ehtinyt olla toteuttamassa monenlaisia projekteja niin sisäisesti kuin ulkoisestikin.

2.2 Projektin esittely

Opinnäytetyöhön liittyen toteutettiin laajennus toimeksiantajan Populon-verkostopalvelun mobiilisovellukseen sisäisenä kehitysprojektina. Populonissa käyttäjät voivat luoda yhteisöjä, järjestää tapahtumia, julkaista tiedotteita ja verkostoitua. Yhteisöt ovat pääosin yksityisiä, johon ryhmän ylläpitäjät voivat kutsua jäseniä sähköpostin välityksellä.

Populonin kaltainen suljettu verkostopalvelu toimi teoriassa varsin hyvin urheiluun keskittyville ryhmille ja joukkueille tapahtumineen ja tiedotteineen. Sellaisenaan monet ryhmien tarpeista, kuten tavoitteet tai harjoitusohjelmat, olivat kuitenkin kankeita tai mahdottomia toteuttaa perusversiossa. Tähän kohderyhmään saatiin myös asiakaskontakteilta palautetta, jonka perusteella päätettiin ryhtyä luonnostelemaan ja kehittämään laajennusta ratkaisemaan näitä puutteita.

Laajennuksen tavoite on tarjota urheilujoukkueille ja -ryhmille työkaluja toimintaan liittyvien asioiden ylläpitämiseen ja luomiseen. Harjoitusten, tavoitteiden ja aikataulujen hallinnointi todettiin tärkeimmiksi ominaisuuksiksi. Myös palautteen kerääminen ja antaminen valmentajille harjoitusten jälkeen osoittautui tarpeelliseksi. Palautteesta saatavan statistiikan todettiin erittäin hyödylliseksi niin urheilijoille kuin valmentajillekin edistymisen seurannan sekä valmennustoiminnan parantamisen kannalta.

3 TEKNOLOGIAT JA TYÖKALUT

3.1 Teknologiat

Projektiin tarvittavat teknologiat ja työkalut määräytyivät pääasiallisesti Populonin olemassa olevan mobiilisovelluksen mukaisesti. Mikäli laajennus olisi kehittynyt omaksi sovelluksekseen, teknologioiden osalta olisi ollut enemmän valinnanvaraa. Laajennus päätettiin kuitenkin integroida itse sovellukseen, joten kehitystä jatkettiin samoilla teknologioilla.

3.1.1 JavaScript

JavaScript on yksi maailman käytetyimmistä ohjelmointikielistä. Se on tarkoitettu ajettavaksi webbiselaimissa, ja on siihen tehtävään erittäin kykenevä. Noin 97 % koko maailman verkkosivuista käyttää JavaScriptiä. (W3Techs, n.d) JavaScriptissä on myös huonot puolensa, kuten heikko tyypitys, joka voi usein helposti johtaa vaikealukuiseen koodiin ja virheisiin ajon aikana. Puutteistaan huolimatta JavaScript on erittäin tuettu ja sille on kehitetty todella monia kehyksiä parantamaan sen toimintaa tai käyttämistä.

JavaScriptillä on pitkä historia, joka alkaa 90-luvun puolivälistä, kun aikansa merkittävä internet-alan toimija Netscape kehitti ja julkaisi sen. Alkuperäiseltä nimeltään LiveScript, JavaScriptin nykyinen nimi syntyi, kun Netscape muutti sen markkinointisyistä sisältämään toisen hyvin tunnetun ja käytetyn ohjelmointikielen, Javan, nimen. Näillä kahdella ohjelmointikielellä ei kuitenkaan ole paljoakaan yhteistä ja ne ovat hyvinkin erilaisia, nimistä huolimatta. Myöhemmin vuodesta 1997 lähtien JavaScriptiä on standardisoitu ECMAScript standardin mukaisesti. ECMAScript määrittelee monia syntaksiin liittyvien asioiden, kuten funktioiden toimintaa.

3.1.2 React

React on vuonna 2013 julkaistu avoimen lähdekoodin frontend-JavaScript-kirjasto, joka on tarkoitettu SPA-sovelluksien luomiseen. Reactin toinen keskeinen tarkoitus on helpottaa uudelleenkäytettävien käyttöliittymäkomponenttien suunnittelua ja luomista. Sen kehitti ja sitä ylläpitää Meta sekä kehittäjien yhteisö.

Reactin suosio kasvoi todella nopeaan ja se on nykyään yksi suosituimmista tavoista toteuttaa verkkosivusto tai applikaatio. Tähän vaikutti Metan, silloisen Facebookin, keskeinen rooli Reactin kehityksessä, lähdekoodin avoimuus sekä sen yksinkertaisuus ja uudelleenkäytettävyys. Lisäksi mahdollisuus toteuttaa koko verkkosivuston käyttöliittymä ja sen toiminta yhden tekniikan avulla on huomattavasti helpompaa kuin puhtaan JavaScriptin ja HTML:n kanssa työskentely.

Puhtaan JavaScriptin ja HTML:n sijasta suositellaan JSX:n käyttöä Reactin kanssa. JSX on laajennus JavaScriptille, joka yksinkertaistettuna helpottaa HTML elementtien luomista ja käsittelyä käyttöliittymäkomponenteissa. Suurimpana hyötynä on koodilausekkeiden sijoittelu HTML-elementteihin, mikä helpottaa dynaamisten komponenttien luomista huomattavasti.

3.1.3 React Native

Metan toimitusjohtaja Mark Zuckerbergin mukaan HTML5:n liiallinen panostaminen koitui kuitenkin virheeksi mobiilialustojen puolella. (Olanoff, 2012) Tämä johti Metan kehittämään yhtiön suhtautumista mobiiliin uuteen suuntaan. Vain kaksi vuotta Reactin julkaisun jälkeen Meta julkaisi uuden ohjelmistokehyksen, jolla toteuttaa mobiiliapplikaatioita. Uuden ratkaisun toivottiin yhdistävän Reactin yksinkertaisuuden sekä natiivien mobiilialustojen toiminnan.

Natiivialustat tarkoittavat laitteiden omia käyttöjärjestelmiä. Esimerkiksi Samsungin älypuhelimet käyttävät Android-käyttöjärjestelmää, kun taas Applen omat puhelimet käyttävät iOS:ää. Näillä järjestelmillä on omat funktiot, arkkitehtuurit, kirjastot ja toteutustavat. Ne eivät myöskään ole keskenään yhteensopivia. Androidille ja iOS:lle kehitetyt sovellukset, eli natiivisovellukset, eivät toimi keskenään ollenkaan. Näin ollen natiivisovellukset pitää kehittää erikseen eri järjestelmille

niiden omia kehyksiä, kieliä ja tapoja käyttäen. React Native kuitenkin ratkaisee tämän ongelman mahdollistamalla natiivisti tulkittavien komponenttien käytön. (React Native Docs, n.d)

React Native on rakennettu Reactin päälle ja pintapuolisesti ne ovat hyvinkin samanlaisia. Monet JSX:n periaatteet, kuten lausekkeiden lisääminen elementtien väliin sekä jaottelu yksinkertaisiin ja uudelleenkäytettäviin komponentteihin, ovat edelleen keskeisessä roolissa. HTML:n ja CSS:n sijaan käytetään pitkälti React Nativelle räätälöityjä ydinkomponentteja ja funktioita, jotka vastaavat natiivialustojen omia komponentteja. Esimerkiksi Reactissa web-applikaatioita rakentaessa elementit usein eritellään Div HTML -elementtien sisään, mikä helpottaa niiden asettelua käyttöliittymässä. React Nativessa vastaavasti käytetään komponenttia View, joka kääntyy vastaamaan Androidin ja iOS:n omia View-toteutuksia riippuen sovellusta käyttävästä laitteesta. Tarvittaessa React Native mahdollistaa myös natiivikoodin, esimerkiksi Javan tai Swiftin, toteutuksen myös, mikäli se on tarpeen kehityksen kannalta.

React Native on osoittautunut erittäin päteväksi vaihtoehdoksi toteuttaa mobiilisovelluksia. Helppous toteuttaa sovellus alustariippumattomasti sekä Androidille että iOS:lle oli erittäin merkittävä tekijä sen valinnassa projektiin toteutukseen. Lisäksi tarve osata vain yksi ohjelmointikieli kahden sijasta oli huomattava etu kehitykseen tarvittavien resurssien ja ajan kannalta. Palvelun web-applikaatio on myös toteutettu Reactilla, joten mobiilisovelluksen kehittäminen oli sujuvampaa ja yksinkertaisempaa toteuttaa.

3.2 Työkalut

Projektissa käytettiin myös muutamaa työkalua, jotka olivat enimmäkseen projektinhallintaan liittyviä. Nämä myös määräytyivät pitkälti perussovelluksessa käytettyjen työkalujen mukaan, sillä haluttiin pysyä samanlaisessa järjestelyissä kuin aikaisemmin.

3.2.1 Node.js

Node.js on JavaScript ympäristö, jonka tarkoituksena on ajaa JavaScript koodia webbiselaimen ulkopuolella. Sen pohjana toimii Googlen V8 JavaScript-moottori. Sen ensimmäinen versio julkaistiin vuonna 2009 ja sen kehitti Ryan Dahl. Nykyinen uusin versio on 18.

Projektissa Node.js:n päätehtävä oli projektin luominen ja eri ohjelmistokehysten asentaminen ja käyttäminen. Varsinaisesti itse projektiin Node.js:ää ei käytetty, vaan se oli alustava teknologia, jota tarvittiin muiden työkalujen ja teknologioiden käyttämiseen.

3.2.2 Npm

Node Package Manager eli npm on paketinhallintajärjestelmä Node.js:lle. Se ylläpitää suurinta JavaScript pakettien rekisteriä. Järjestelmän avulla kuka tahansa voi jakaa kehittämiään JavaScript paketteja tai ladata muiden kehittämiä paketteja omaan käyttöönsä. Paketteihin ja kirjastoihin liittyy omat lisenssit, joiden avulla kehittäjät voivat rajoittaa millaisiin projekteihin heidän pakettejaan voidaan käyttää. Julkisten kirjastojen jakaminen on ilmaista, mutta yksityisten pakettien jakamiseen tarvitsee Npm Organizations -tilin, johon sisältyy 7 dollarin kuukausimaksu käyttäjää kohden. (Npm Docs, n.d)

Npm:n kolme tärkeintä osaa ovat sen verkkosivut, rekisteri ja komentorivikäyttöliittymä eli CLI. Verkkosivuilla voi hallita ja organisoida omia julkaistuja paketteja sekä etsiä julkaistuja paketteja. Paketeilla on oma sivu, johon kehittäjät usein liittävät tiedot riippuvuuksista muihin paketteihin, versiohistorian sekä asennus ja -käyttöohjeet. Npm:n julkinen rekisteri on CouchDB-pohjainen tietokanta, joka pitää sisällään julkaistut paketit ja niiden metatiedot. Se on toteutettu CommonJS Package Registry -spesifikaation mukaisesti, joka määrittelee, miten paketit tunnistetaan nimen, version, ja URL-osoitteen mukaisesti. CLI on se työkalu, jota kehittäjät käyttävät julkaistujen pakettien ja moduuleiden hallinnoimiseen ja asentamiseen. Sitä ajetaan terminaalin kautta ja koostuu pitkälti komennoista, kuten npm install, joiden kanssa se on vuorovaikutuksessa rekisterin ja projektin kanssa.

Npm on vain yksi Node.js:n paketinhallintajärjestelmistä, ja se on myös oletusvaihtoehto, jonka asentamista suositellaan Node.js:n asennuksen yhteydessä. Vaihtoehtoina npm:lle ovat esimerkiksi Yarn sekä pnpm. Yarn pyrkii optimoimaan paketinhallintaprosessia ja mielletään tehokkaamaksi ratkaisuksi. Yarnin uudemmat versiot voivat olla moninkertaisesti nopeampia kuin npm riippuen toiminnosta sekä välimuistin käytöstä (Holmgren 2020). Pnpm toisaalta taas pyrkii helpottamaan tallennustilan käyttöä. Jokainen Node.js projekti sisältää node_modules -kansion, joka pitää sisällään projektiin liittyvät paketit, kirjastot sekä riippuvuudet. Kansio voi projektista riippuen olla hyvinkin suuri, ja ongelma vain kasvaa mitä useampia projekteja laitteella kehitetään. Pnpm säilyttää vain yhden kappaleen kustakin paketista, mutta niitä voi käyttää mistä tahansa projektista. Tämä säästää tallennustilaa, ja on erittäin hyödyllistä laitteilla, joissa tallenustila on vähissä tai projekteja on todella monta samanaikaisessa kehityksessä.

3.2.3 Git

Git on hajautettu avoimen lähdekoodin versionhallintajärjestelmä, jonka on alun perin kehittänyt Linus Torvalds Linux-käyttöjärjestelmän yhteydessä. Sen ensimmäinen versio julkaistiin huhtikuussa 2005. Nykyinen vakaa versio on 2.36.1. Se mahdollistaa muun muassa projektiin tehtyjen muutosten jakamisen, versioinnin, haaroittamisen ja yhdistämisen, jotka helpottavat usean kehittäjän samanaikaista työskentelyä. Vaikka Git on itsessään komentoriviltä käsin käytettävissä, sille on kehitetty monia eri sovelluksia, jotka mahdollistavat sen käyttämisen käyttöliittymän avulla.

Projektissa Git oli keskeinen versionhallinnan työkalu. Koska kyseessä oli olemassa oleva sovellus, oli tärkeää pitää laajennuksen kehitys omissa Git-haaroissaan, jotta muutokset eivät haittaisi perussovelluksen kehitystä. Samanaikaisesti perussovellukseen tehdyt muutokset eivät haitanneet laajennuksen kehitystä.

3.2.4 Visual Studio Code

Visual Studio Code on Microsoftin kehittämä avoimen lähdekoodin tekstinmuokkausohjelma, joka on pääasiallisesti tarkoitettu ohjelmointiin ja sopii loistavasti

eritoten JavaScriptillä kehittämiseen. Se on kevyt verrattuna moniin ohjelmointiympäristöihin, ja sisältää tuen Git-versionhallinnalle, tekstinkorostukselle ja automaattiselle koodintäydennykselle. Sille on myös saatavilla yhteisön tekemiä laajennuksia, jotka usein tuovat tuen uusille kirjastoille tai ohjelmistokielille.

Visual Studio Code on hyvä valinta Expolla kehittämiseen sen tarjoamien laajennusten ja JavaScriptin käytön ansiota. Tarvetta raskaammalle ohjelmointiympäristölle ei juurikaan ole sillä tarve ajaa sovellusta tai seurata konsolia tai terminaalia itse ohjelmistoympäristössä on pieni.

4 EXPO

4.1 Mikä on Expo?

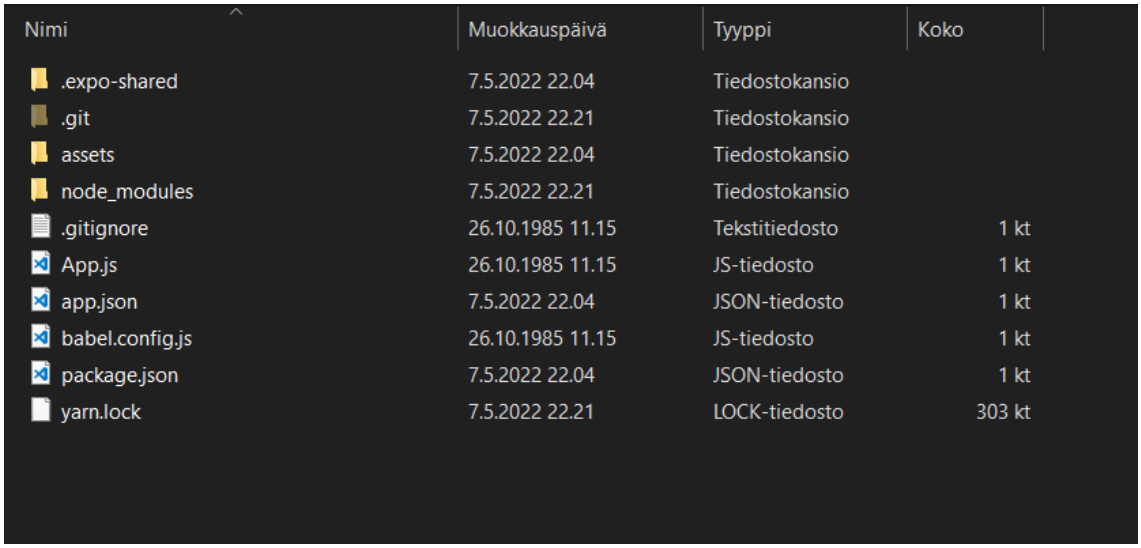
Expo on React Nativelle toteutettu ohjelmistokehys. Sen tarkoitus on parantaa ja suoraviivaistaa React Nativen kehityskokemusta tarjoamalla työkaluja kehittäjän käytettäväksi sekä yksinkertaistamalla pintapuoleista toimintaa. Expon toiminta perustuu kahteen työkaluun: Expo CLI:in ja Expo Go-sovellukseen.

4.1.1 Expo CLI

Expo CLI on komentorivikäyttöliittymä, jota voidaan ajaa terminaalista käsin. Sen asennukseen tarvitaan Node.js sekä paketinhallintajärjestelmä, joka tässä projektissa on npm. Expo CLI on saatavilla tavallisena npm-pakettina ja sen voi asentaa npm install -komennolla. Asennuksen voi varmistaa expo whoami -komennolla, joka kertoo Expo-käyttäjätunnuksesi tai antaa viestin, ettet ole kirjautunut sisään.

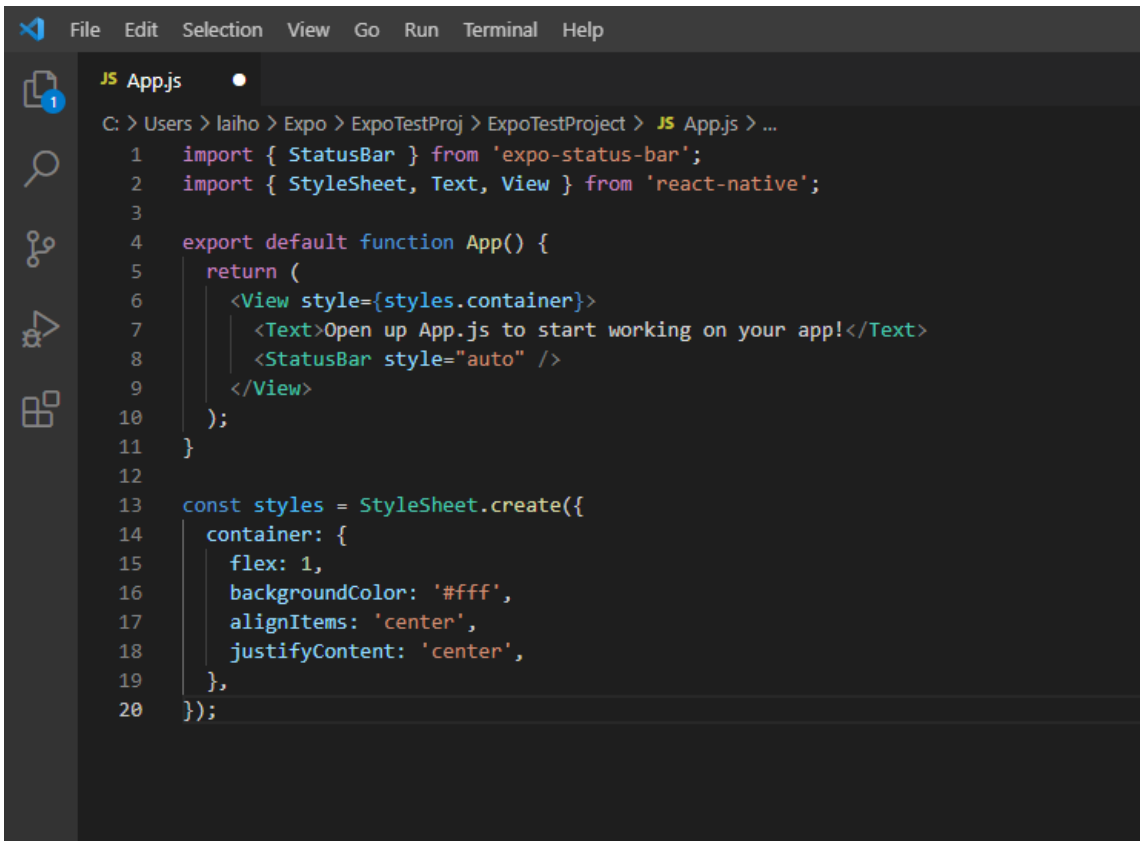
Kun Expo CLI on asennettu, uuden projektin luominen tapahtuu komennolla expo init, joka pyytää projektille nimeä ja luo projektikansion sen hetkiseen kansioon. Tämän jälkeen käyttäjä valitsee projektin työnkulun, ja halutun kielen. Expo tukee sekä perinteistä JavaScriptiä sekä TypeScriptiä, joka on vahvaa tyypitystä tukeva muunnos JavaScriptistä. Työnkulkuun voi valita ohjatun tai paljaan työnkulun. Paljas on lähellä tavallista React Native -kehitystä, jossa kehittäjien on hoidettava myös natiiveja Android- ja iOS -projekteja, mikä ei ole mahdollista toteuttaa pelkällä JavaScriptillä tai TypeScriptillä. Se kuitenkin tarjoaa mahdollisuuden silti käyttää osaa Expon tarjoamista palveluista. Ohjattu työnkulku toisaalta tarkoittaa vain valitulla kielellä ohjelmoimista, yhden projektin hallitsemista sekä mahdollisuutta täysin hyödyntää Expon palveluja ja toimintoja. Ohjatussa työnkulussa on kuitenkin huonot puolensa, kuten heikompi kontrolli eri alustojen toimintaan. Lisäksi kirjastot ja paketit saattavat tukea React Nativea, mutta eivät Expon ohjattua työnkulkua.

Työnkulun valinnan jälkeen, Expo luo uuden Hello World -tyylisen projektin, joka on valmiina käynnistettäväksi. Paljaan työnkulun valittaessa kansio sisältää Android ja iOS -projektit. Mikäli käyttäjä on myös asentanut Git:n, Expo luo projektista myös Git-arkiston versionhallintaa varten. Tämä säästää käyttäjältä vaivan tehdä se itse. Kuvassa 1 on esimerkki Expon luomasta aloitusprojektista.



KUVA 1. Expon luoman aloitusprojektin kansiorakenne ohjatulla työnkululla

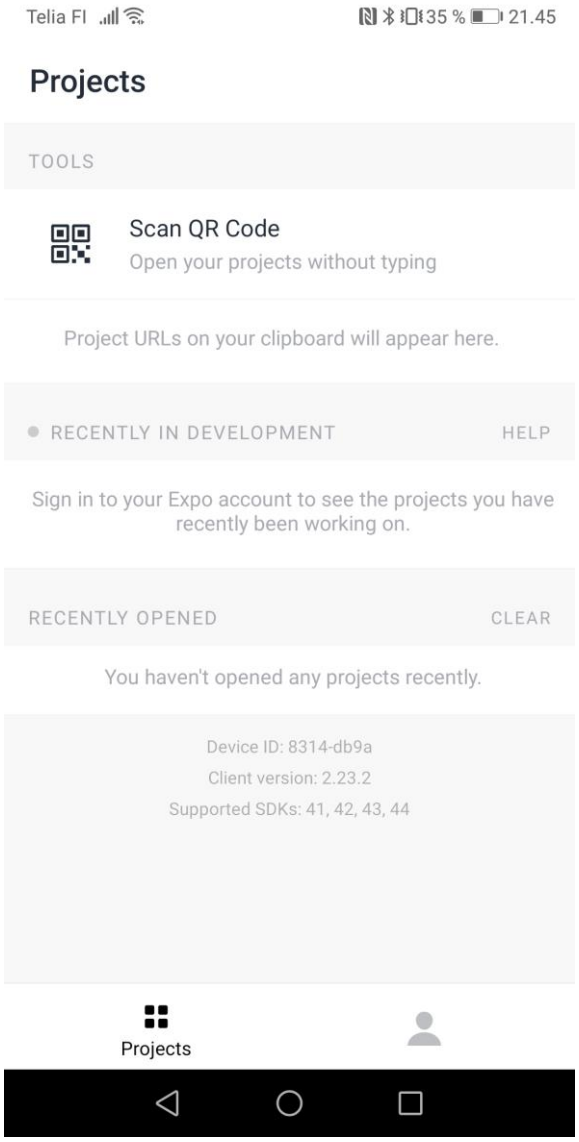
Lopullisesta aloitusprojektista löytyvät Git-arkiston tiedostojen lisäksi myös projektin Node-moduulit, jotka ovat omassa kansiossaan, joka on myös suljettu pois versionhallinnasta gitignore-tiedostolla. Myös riippuvuuksien hallintaan liittyvät tiedostot, kuten package.json, yarn.lock ja package-lock.json tiedostot, ovat valmiita käytettäväksi. Itse koodi löytyy App -nimisestä tiedostosta, josta käyttäjä voi lähteä kehittämään sovellustaan. Se on joko JavaScript- tai TypeScript-tiedosto riippuen aiemmista valinnoista. Kuvassa 2 on App.js-tiedoston yhden funktion Hello World -tyyppinen koodi. Tässä vaiheessa itse sovellus näyttää ainoastaan tekstin, joka kehottaa avaamaan App.js:n sovelluksen työstämiseksi.



KUVA 2. Expon aloitusprojektin App.js-tiedosto Visual Studio Codessa

4.1.2 Expo Go

Expo Go on mobiilisovellus Android ja iOS -käyttöjärjestelmille, ja on saatavilla App Store ja Google Play -kaupoista ilmaiseksi. Sovelluksen päävalikosta voi valita käynnistää viimeisimpiä projekteja, jos niiden kehitysympäristöt ovat päällä. QR-koodinlukijalla voi myös käynnistää Expo CLI:ssä käynnissä olevan projektin. Kuvassa 3 on esitetty, miltä Expo Go:n päävalikko näyttää.



KUVA 3. Expo Go -sovelluksen päävalikko

Profiili -välilehdeltä käyttäjä voi kirjautua sisään omalle Expo kehittäjätilille. Tiliä tai sisään kirjautumista ei vaadita kehityksen aloittamiseen, mutta tietyt ominaisuudet liittyen projektinhallintaan vaativat tilin. Kirjauduttua sisään voi myös tarkastella muita tilille asetettuja projekteja.

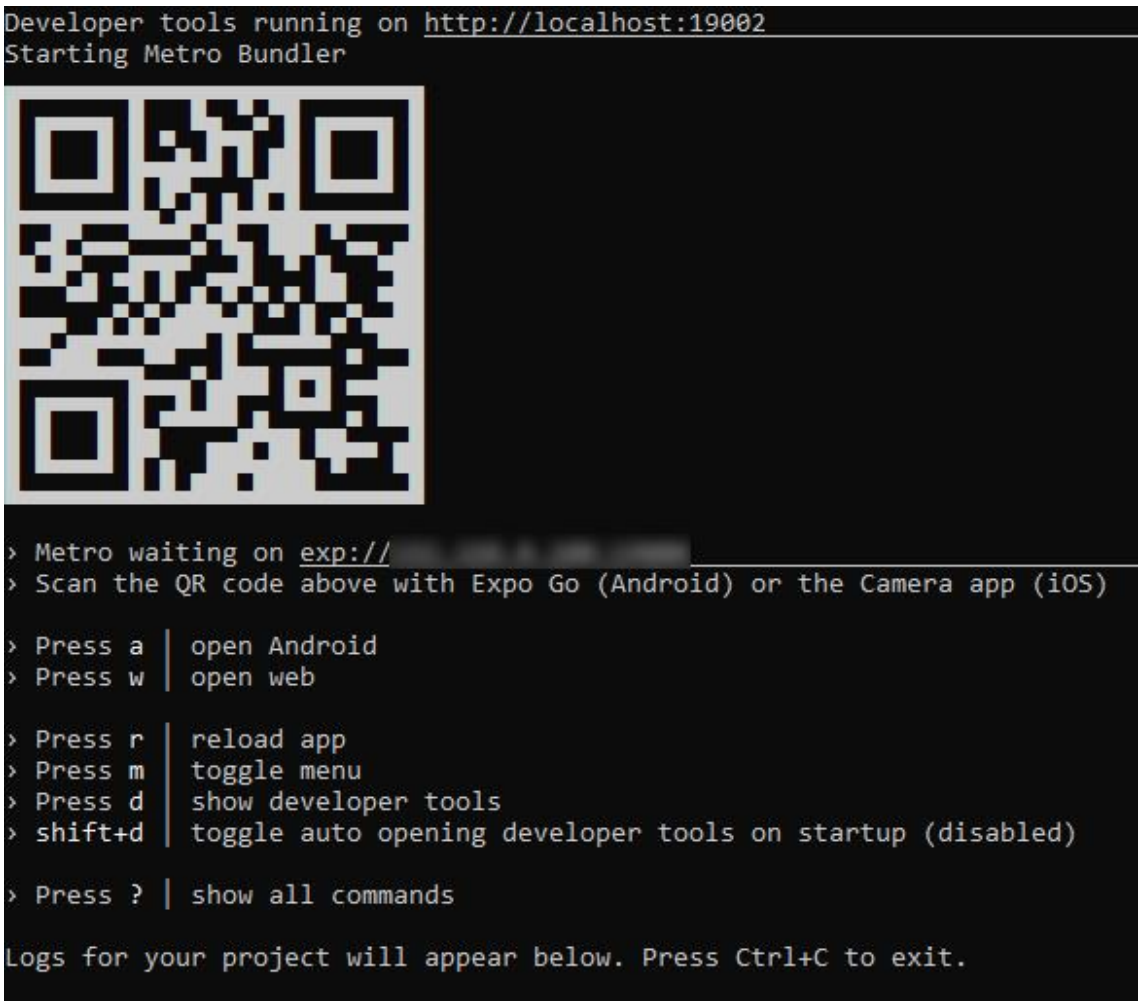
4.2 Kehittäminen Expolla

Expo on React Nativen ohjelmistokehys, joten kehittäminen Expolla seuraa pitkälti samaa kaavaa kuin tyypillinen React Native kehitys. Toisin kuin React Nativella, Expolla ei tarvitse tehdä mitään natiiviprojektien kanssa, vaan koko kehityksen voi toteuttaa JavaScriptillä.

4.2.1 Projektin ajaminen

Expo projekti käynnistetään komennolla expo start, joka käynnistää kehitysympäristön. Ympäristö toimii paikallisella palvelimella, joka hoitaa kommunikaation projektin ja Expo Go -sovelluksen kanssa. Expo tukee myös tunnelointimahdollisuutta hyödyntämällä Ngrok-tunnelipalvelua. Vaikka se hieman hidastaa toimintaa ja kommunikaatiota laitteen ja kehitysympäristön välillä, se mahdollistaa muiden kuin paikallisessa verkossa olevien käyttäjien ottavan yhteyden palvelimeen. Ympäristön tärkein osa on JavaScript paketointityökalu Metro, joka on myös Facebookin kehittämä. Metro automaattisesti kokoaa projektin JavaScript koodin ja resurssit niin kutsuttuihin bundleihin sekä tarvittaessa muuntaa niitä laitteille paremmin ymmärrettävään muotoon.

Avatakseen projektin mobiililaitteella Expo luo QR-koodin kehitysympäristön käynnistämisen yhteydessä. Kehittäjät voivat skannata tämän koodin Expo Go sovelluksella, mikä luo yhteyden laitteen ja palvelimen välille. Yhteyden muodostettua, Metro luo bundlen ja lähettää sen laitteelle, joka alkaa ajamaan sovellusta kehitystilassa. Kuvassa 4 esitetään Expon valmiina olevaa kehitysympäristöä ja QR-koodia, jolla projektin voi käynnistää.

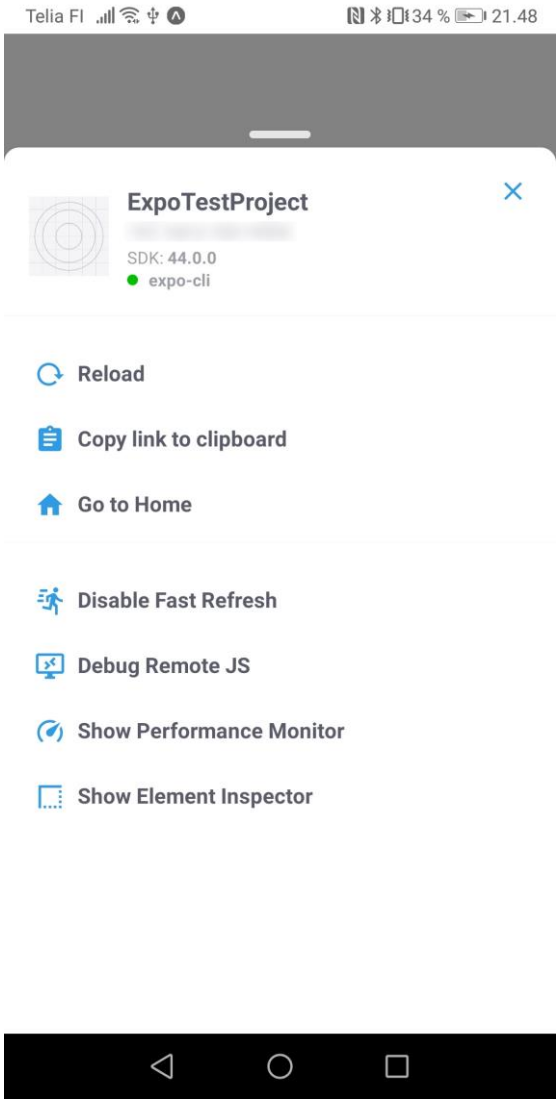


KUVA 4. Expo CLI ja kehitysympäristö valmiina käyttöön

4.2.2 Kehittäjän ominaisuudet

Expo tarjoaa kehittäjille joukon ominaisuuksia ja työkaluja helpottamaan sovelluksen kehittämistä. Useat näistä toimii samalla tavalla kuin vastaavat toiminnot Reactissa ja React Nativessa, kuten Fast Refresh eli nopea uudelleenlataus, joka päivittää sovelluksen koodiin tehdyt uusimmat muutokset. Kehittäjävalikon saa auki heilauttamalla laitetta tai heilautusta vastaavalla komennolla tai näppäinyhdistelmällä. Kuvassa 5 Expon kehittäjävalikko, josta voi käyttää kehittäjille tarkoitettuja työkaluja.

Valikosta voi myös tarkastella sovelluksen tehokkuutta ja muistinkäyttöä. Etenkin suuremmissa projekteissa tai sovelluksissa, optimointi on tärkeä osa ja mahdollisuus seurata muistin käyttöä tai kuvataajuutta helpottaa merkittävästi optimoitavien näkymien tai toimintojen löytämistä.



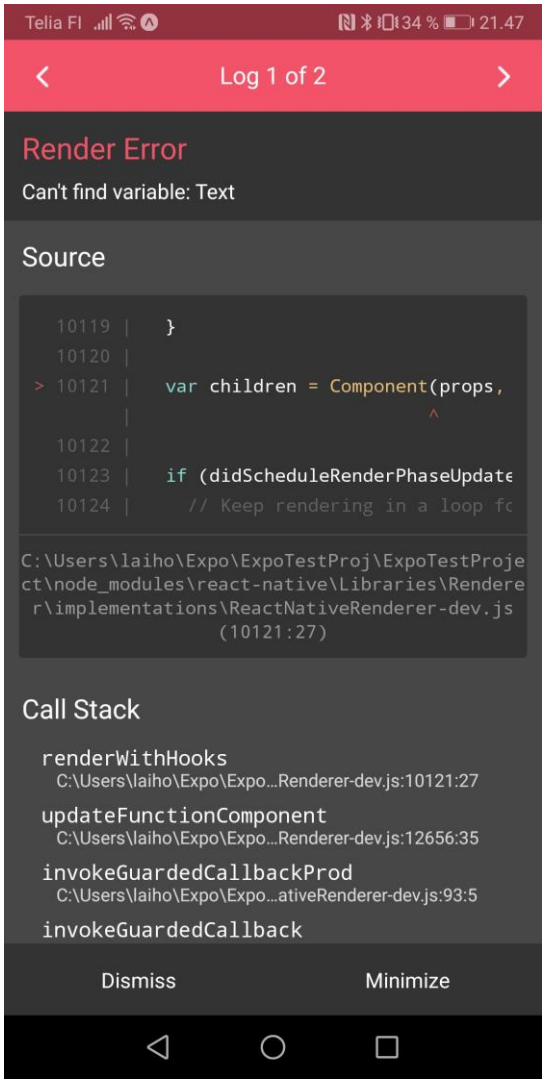
KUVA 5. Kehittäjävalikko Expossa

4.2.3 Virheidenhallinta

Expon virheidenhallinta on hyvin samankaltainen React Nativen virheidenhallinnan kanssa. Jos sovelluksessa tapahtuu kriittinen virhe kehityksen aikana, Expo antaa virheilmoituksen, joka pyrkii kertomaan, mikä meni vikaan ja missä komponentissa. Kuvassa 6 on Expon virheilmoitusnäkymä, joka on tapahtunut, koska koodista ei löydy muuttujaa Text. Jos virheitä tai varoituksia on useampia, niitä voi selata ja tarkastella erikseen. Uudelleen lataaminen tai muutosten tekeminen ja tallentaminen, jos Fast Refresh on päällä, tekee uuden paketin ja sulkee virheilmoituksen automaattisesti.

Tietyissä tapauksissa Expon virheidenhallinta ei paljoa helpota virheen löytämisessä. Vaikka se usein antaa vähintäänkin hyvän suunnan tai idean siitä mikä

meni vikaan, joskus itse virhe tai sen lähde eivät ole Expon mukaan missään mitä kehittäjä itse on työstänyt. Etenkin uusille kehittäjille tämä voi olla hämmentävää. Usein virheen tai sen lähteen poikkeavat tiedot johtuvat Expon taustalla käytettävistä kirjastoista tai komponenteista, joissa virhe teknisesti tapahtuu, vaikka kehittäjä itse ei olisikaan tehnyt niihin mitään muutoksia. Kuvan 6 esimerkissä, App.js tehdyt muutokset aiheuttavat virheen React Nativen renderöintikirjastossa. Usein kuitenkin vähintään toisesta kategoriasta löytyy vihje virheen alkuperästä.



KUVA 6. Expon virheilmoitusnäkymä

4.2.4 Koontiversion tekeminen

Koontiversion, eli buildin tekeminen Expolla tapahtuu CLI:n kautta antamalla komento expo build:android tai expo build:ios riippuen alustasta, jolle build halutaan

tehdä. Expo rakentaa buildit omilla palvelimillaan, joilta voi seurata buildin tilaa tai komennolla expo build:status. Kun build on valmis, sen voi ladata, jolloin se toimii kuin muutkin alustakohtaiset buildit. Sovelluksen julkaisunkauppapaikoille voi tapahtua manuaalisesti alustakohtaisten käytäntöjen mukaisesti tai käyttämällä Expon omia työkaluja.

4.3 Eroavaisuudet React Nativesta

React Native ja Expo vastaavat toisiaan pitkälti, sillä Expo on kuitenkin kehys React Nativelle. Vaikka Expo tuo helpotuksia kehitystyöhön, se menettää manuaalista kontrollia mitä React Nativella on mahdollista hyödyntää. React Native ei myöskään voi käyttää Expon tarjoamia työkaluja, vaan sen täytyy käyttää omia, joihin usein liittyy natiiviprojektien konfiguroimista.

Koska React Nativella kehitetään Androidille ja iOS:lle omat projektit, kehittäjillä täytyy olla mahdollisuus käyttää niiden edellyttämiä ohjelmointiympäristöjä, Android Studiota sekä XCodea. XCode toimii ainoastaan Applen Mac-tietokoneilla, joten kehittäjillä täytyy olla pääsy myös sellaiseen. Tämä voi olla este React Nativella kehittämiselle, etenkin kun kyse on harrastelijakehittäjistä tai pienistä yrityksistä. Expo on kuitenkin käytettävissä, koska sen käyttämiseen ei tarvita kuin tekstinmuokkausohjelma ja sopiva mobiililaite.

4.4 Miksi Expo tai miksi ei?

Vaikka Expo on tehty helpottamaan kehittäjien elämää React Nativella kehittäessä, se ei ole täydellinen. Riippuen projektin tai kehittäjien tarpeista, Expo ei välttämättä ole otollinen valinta. Se soveltuu parhaiten nopeaan kehitykseen tai suhteellisen kevyen sovelluksen tekemiseen, jonka tarpeet Expon kirjastot voivat täyttää.

Merkittävin tekijä on usein kontrollin heikentyminen projektin tai sen käytössä olevien kirjastojen riippuvuuksien suhteen. Expo voi hyödyntää ainoastaan sen omia tai nimenomaan sitä tukevia kirjastoja. Tämä jättää niin natiivikirjastoja kuin muiden kehittäjien jakamia täysin kehittäjien ulottumattomiin. Riippuen sovelluksen

tarpeista tai suunnitelluista ominaisuuksista, Expo ei välttämättä voi toteuttaa sitä. (Expo Docs, n.d)

Hyvänä puolena kuitenkin on, että Exposta voi tarvittaessa siirtyä puhtaaseen React Nativeen, jolloin ei tarvitse huolehtia Expon tuomista rajoituksista. Tämä tapahtuu expo eject -komennolla, joka muuntaa projektin tyypilliseksi React Native projektiksi, joka sisältää omat projektit Androidille ja iOS:lle.

5 LAAJENNUS MOBIILISOVELLUKSEEN

5.1 Laajennukset yleisesti

Laajennus on usein erilainen ja irrallinen asia yksinkertaisesta uuden ominaisuuden lisäämisestä, vaikka laajennukset voivat olla yksinkertaisia tai yksittäisiä toimintoja, etenkin, jos kehittäjä ei ole virallinen vaan esimerkiksi yhteisön jäsen. Tyypillisesti laajennukset lisäävät yhtäaikaisesti joukon uusia toiminnallisuuksia, jotka keskittyvät tietyn tarkoituksen tai tarpeen täyttämiseen. Esimerkiksi laajennus, joka lisää valokuvien ottamisen usein lisää myös otettujen kuvien tallentamisen sekä katsomisen.

Kehittäjät usein haluavat käyttäjien kokevan laajennuksen merkittäväksi, ja sovelluksen tai palvelun käyttöä parantavaksi. Sovelluksen luonteesta ja toteutustavasta riippuen laajennukset saattavat olla suurempia tai pienempiä. Laajennus ei myöskään ole välttämättömyys vaan se usein on tarkoitettu parantamaan tiettyä osaa tai käyttötarkoitusta sovelluksessa. Jos jokin ominaisuus vaikuttaa välttämättömältä sovelluksen alkuperäisen tavoitteen tai tarkoituksen kannalta, se on kannattavampaa toteuttaa tavallisena päivityksenä, joka on saatavilla kaikille käyttäjille. Näin sovelluksen alkuperäinen tarkoitus tai tehtävä ei kärsi.

5.2 Ongelmat mobiililla

Mobiilialustoilla laajennuksien suurin ongelma on dynaamisen jakamisen hankala toteutus. Androidille on olemassa toiminnallisuus osittaa sovellus tai sen toiminnot eri moduuleihin, jotka ovat dynaamisesti ladattavissa ja asennettavissa käyttäjän pyynnöstä.

IOS:llä vastaava järjestelmä on myös olemassa, mutta se ei tue suoritettavan koodin sisällyttämistä moduuleihin. Käytännössä moduulit voivat sisältää erilaisia resursseja, kuten dataa, kuvia tai muita visuaalisia elementtejä. Tämä johtuu Applen mobiilikauppapaikka App Storen sovellusten hyväksymisen ohjeistuksesta. Ohjeistuksen mukaan, App Storeen julkaistavien sovellusten tulisi olla kokonaisia paketteja, eivätkä ne saisi ladata, asentaa tai suorittaa koodia, joka muuttaa sovelluksen toimintaa tai tuo uusia ominaisuuksia. (Apple, n.d) Ohjeistuksessa on

poikkeus opetustarkoituksellisille sovelluksille, mutta useimmat sovellukset eivät tähän kategoriaan kuulu, mukaan lukien Populon.

Applen ohjeistus luo tilanteen, jossa laajennukset ovat järkevintä toteuttaa dynaamisina moduuleina, mutta se tarkoittaisi iOS:n tuesta luopumista tai oman toteutuksen suunnittelua iOS:lle. Liiketoiminnan kannalta iOS -tuesta luopuminen on kuitenkin merkittävä päätös, sillä iOS:n osuus mobiilialustojen markkinoista on noin neljännes maailmanlaajuisesti. (Laricchia, 2022)

5.3 Ratkaisut ja toteutus

Koska yleisesti kannattavin tapa toteuttaa laajennus mobiilisovellukselle on iOS:n sovellusohjeistuksen takia hankala, laajennuksen toteuttamiseksi jää kaksi muuta vaihtoehtoa. Ne eivät ole yhtä helposti skaalattavia tai ylläpidettäviä kuin modulaariset laajennukset, mutta mahdollistavat kuitenkin uusien toiminnallisuuksien tarjoamisen niitä haluaville käyttäjille.

Ensimmäinen vaihtoehdoista on laajennuksen erottaminen omaksi sovellukseksi. Koska uuden sovelluksen kehittämisessä on omat haasteensa ja tarvittavat järjestelmät, tähän vaihtoehtoon sisältyy enemmän perustan rakentamista, ennen kuin edes itse laajennuksen kehitys voi alkaa. Tämä toteutus kuitenkin tarjoaa enemmän joustavuutta projektin rakentamiseen ja teknologiavalintoihin.

Toinen vaihtoehto on laajennuksen integroiminen perussovellukseen. Tämä on usein helpompi toteuttaa, sillä monet kriittiset järjestelmät, kuten kirjautuminen ja tunnistautuminen ovat jo olemassa. Tällöin aika ja resurssit menevät suoraan itse laajennuksen ominaisuuksien kehittämiseen eikä tarvitse huolehtia uuden sovelluksen perustan rakentamisesta.

5.4 Laajennusten kipukohdat

Toteutustavasta riippumatta laajennuksiin liittyy seikkoja, jotka saattavat vaatia erilaista lähestymistapaa kuin perussovellus. Nämä niin sanotut kipukohdat ovat keskeisiä myös yleisesti ohjelmistokehityksessä. Vaikka ne ovat varsin tavallisia

ohjelmiston tai sovelluksen suunnitteluvaiheessa käsiteltäviä seikkoja, laajennusta tehdessä ne usein pitää ottaa huomioon uudelleen. Etenkin laajemmat laajennukset, jotka lisäävät enemmän kuin vain muutaman ominaisuuden, saattavat vaatia hyvin erilaisia lähestymistapoja niiden toteutukseen. Toteutustavasta riippuen laajennus saattaa myös muuttaa perussovelluksen vaatimuksia.

5.4.1 Lähdekoodi ja sen ylläpito

Lähdekoodi on keskeinen osa sovelluksen tai järjestelmän toteutusta, ja pitää kokonaisuuden sisällään käytetyn ohjelmointikielen muodossa. Koska koko sovellus on rakennettu sen varaan, kaikki sen toiminta myös riippuu siitä. Kun jo olemassa olevaan sovellukseen luodaan laajennus, se vääjäämättä tarkoittaa lisää lähdekoodin kehittämistä tai huoltoa.

Mikäli uusi laajennus päätetään erottaa omaksi sovelluksekseen, se tarkoittaa uuden projektin ja näin ollen lähdekoodin luomista. Alkuperäinen lähdekoodi säilyy ennallaan ja vaatii saman verran kehitystä ja ylläpitoa kuin aikaisemminkin. Mutta uusi lähdekoodi toisaalta vaatii huomattavasti huomiota, sillä se pitää kehittää alusta asti ja ylläpitää niin kuin alkuperäinenkin. Vaihtoehtona on toteuttaa uusi sovellus hyvin samalla tavalla ja teknologioilla kuin perussovellus, jolloin monia järjestelmiä, komponentteja tai näkymiä voidaan nopeasti ottaa käyttöön toteuttamalla ne samalla tavalla kuin alkuperäisessäkin. Tämä vaihtoehto ei ole kuitenkaan aina mahdollinen. Riippuen omaksi sovelluksekseen erotetun laajennuksen tarpeista ja toiminnoista, vanha toteutus tai sen käyttämät teknologiat eivät välttämättä palvele uusia vaatimuksia tai käyttökohteita. Lisäksi kaksi samanlaista lähdekoodia, jotka ovat pitkälti jakaneet toteutuksiaan, tekevät ylläpitämisestä raskaampaa. Jos keskenään jaetusta koodista löytyy virheitä, ne pitää korjata molemmista projekteista, mikä vaatii enemmän resursseja ylläpitää.

Jos laajennus integroidaan perussovellukseen, se pitkälti pitää lähdekoodin tarpeet ja sen ylläpidon ennallaan. Tässä tilanteessa laajennus on vain lisäominaisuuksia, joihin tietyillä käyttäjillä on pääsy, joten vaikka muutoksia ja lisäyksiä koodiin tarvitsee tehdä, ylläpito ei paljoakaan muutu. Lisäykset koodiin on myös hyvä pitää erillään sen uudelleenkäyttömahdollisuuksien ulkopuolella. Ehtojen syöttäminen koodin ja niiden avulla komponenttien käyttäytymisen ohjaaminen ei

ole yleisesti kovin tehokas tai luettava vaihtoehto. Se myös paisuttaa koodia ja vaikeuttaa ylläpitoa. Laajennuksen integroiminen voi olla kuitenkin melko kankeaa, sillä se myös sitoo laajennuksen käyttämään pitkälti samoja teknologioita, toteutustapoja ja infrastruktuuria kuin perussovellus. Riippuen laajennuksen tarpeista ja tavoitteista, se voi aiheuttaa ongelmia laajennuksen halutun toiminnan toteutuksessa. Käytössä olevilla tekniikoilla voi olla hankalaa toteuttaa haluttuja toimintoja.

Toimeksiantajan projektin tapauksessa laajennus päätettiin integroida perussovellukseen. Integroiminen todettiin helpommaksi lähestymistavaksi projektin alkuvaiheissa, kun suunnitelmia laajennuksen lopullisista yksityiskohdista vielä hiottiin. Perussovellukseen integroiminen antoi laajennukselle nopean tavan muodostua ja tuoda esiin suunnitelman osa-alueita, jotka olivat jääneet huomiotta suunnitteluvaiheessa. Lisäksi halutut toiminnallisuudet katsottiin mahdollisiksi toteuttaa käyttäen Expoa.

5.4.2 Järjestelmien ja komponenttien uudelleenkäyttö

Ohjelmoinnissa luokkien, järjestelmien ja komponenttien uudelleenkäyttö on yksi alan keskeisimmistä käytännöistä, jota olisi hyvä hyödyntää mahdollisimman paljon. Koodin kehittäminen uudelleenkäytettäväksi säästää aikaa, kun kehittäjien ei tarvitse kirjoittaa samaa asiaa useampaan kertaan, sekä tekee koodista luettavampaa. Lisäksi se myös usein säästää tilaa, joka on mobiililla tärkeää, sillä käyttäjät saattavat käyttää maksullista mobiilidataa ladatessaan sovellusta tai sen päivityksiä. Google Play:lla ja App Storella on myös rajoitukset sovellusten koosta.

Kun kyseessä on jo valmiiksi olemassa oleva järjestelmä tai sovellus, johon laajennus integroidaan, osa komponenteista voi olla jo valmiiksi uudelleenkäytettävissä riippuen niiden alkuperäisestä toteutuksesta. Hyvin valmistetussa koodissa suurin osa olisi valmiiksi geneeristä ja uudelleenkäytettävää, mutta geneerisyyttä ei välttämättä ole tarvittu ottaa huomioon sovelluksen alkuperäisen tarkoituksen suhteen. Tämän takia saatetaan joutua tekemään muutoksia lähdekoodin ja komponentteihin uutta laajennusta integroitaessa perussovellukseen paremman uudelleenkäytettävyyden saavuttamiseksi. Parempi uudelleenkäytettävyys usein

myös helpottaa lähdekoodin ylläpitämistä. Mikäli sovellukseen on myöhemmin suunnitelmissa integroida lisää laajennuksia, uudelleenkäytettävyyteen panostaminen on entistä tärkeämpää.

Jos laajennuksesta tehdään erillinen sovellus, uudelleenkäytettävyys on edelleen tärkeää sen tuomien hyötyjen myötä, mutta ei yhtä kriittistä. Omalla sovelluksella on epätodennäköistä, että tilan rajoitukset tulisivat vastaan. Lisäksi komponentit palvelevat vain yhden kokonaisuuden tarpeita, eivätkä kahden.

Populonin perussovellukseen integroitaessa muutoksia koodin uudelleenkäytettävyyden parantamiseksi täytyi tehdä varsin vähän. Muutamaa komponenttia muutettiin geneerisemmiksi, jotta ne soveltuivat myös laajennuksen tarpeisiin. Tämä säästi aikaa ja kyseisiä komponentteja on nyt helppo käyttää sekä perussovelluksessa että laajennuksessa.

5.4.3 Identiteetti

Identiteetti tarkoittaa sovelluksen, palvelun, tuotemerkin, tai tässä tapauksessa, laajennuksen yleistä tunnetta tai persoonallisuutta. Markkinoinnin suhteen identiteettiin liittyy monia komplekseja tekijöitä, kuten markkinoinnin kohdentaminen ja positiivisen imagon rakentaminen. Sovellustasolla identiteetti tulee usein ilmi logojen, värimaailmojen tai johdonmukaisen suunnittelun ja toiminnan kautta.
Identiteetin rakentaminen on huomattavasti vapaampaa, kun laajennus on toteutettu omana sovelluksenaan. Sen voi yhä tietenkin liittää osaksi alkuperäisen sovelluksen tuotemerkkiä, mutta sen ei tarvitse seurata perussovelluksen identiteettiä. Äärimmäisessä tapauksessa laajennus voi ajan myötä kehittyä jopa täysin erilliseksi tuotemerkiksi.

Jos laajennus on toteutettu osana perussovellusta, identiteetin luominen on usein sidottu perussovelluksen identiteettiin, sillä käyttäjän on pakko käyttää perussovellusta. Erilliset identiteetit voisivat olla erittäin hämmentäviä käyttäjille, jotka joutuisivat käyttämään haluamaansa sovellusta täysin erilaisen sovelluksen sisällä. Tästä johtuen integroidun laajennuksen on hyvä seurata perussovelluksen identiteettiä. Laajennuksella on kuitenkin jonkin verran liikkumavaraa identiteettinsä

suhteen. Se voi hyödyntää omaa logoa, värejä tai tarjota hieman erilaisen käyttäjäkokemuksen, kunhan se ei poikkea liian paljoa alkuperäisestä.

Koska projektin laajennus integroitiin perussovellukseen, identiteetin säilyttäminen melko samanlaisena oli tärkeää. Laajennukselle haluttiin oma logo, joka saatiin helposti suunniteltua käyttäen perussovelluksen logoa osana sitä. Logon lisäksi käyttäjäkokemusta ja käyttöliittymää käytettiin erottamaan laajennusta perussovelluksesta, jotta käyttäjille olisi silmäyksellä selvää kumpi on käytössä, mutta molemmat säilyvät pitkälti samanlaisina.

5.4.4 Käyttäjäkokemus

Käyttäjän kokemuksia ja tunteita palvelun tai tuotteen käytöstä kutsutaan käyttäjäkokemukseksi. Vaikka käsitteeseen usein liitetään vain konkreettiset asiat, kuten käyttöliittymä tai sen ulkonäkö, käyttäjäkokemukseen kuuluu monia muitakin osa-alueita. Tuotteen tai merkin imago ja suunnittelukin voidaan katsoa osaksi kokonaisuutta, joka voi alkaa jo ennen kuin käyttäjä on edes saanut tuotteen käytettäväkseen. (Interaction Design, n-d).

Etenkin käyttöliittymien ja visuaalisen puolen asiat ovat usein frontend-kehittäjiä lähellä. Yksi tärkeä aspekti käyttäjäkokemuksessa on elementtien ja toimintojen sisäinen johdonmukaisuus. Tähän kuuluvat muun muassa värit, näkymät, asettelut ja muut visuaaliset elementit. Sisäinen johdonmukaisuus helpottaa käyttäjien kokemusta vähentämällä tarvetta ajatella sekä luomalla johdonmukaisia toimintatapoja, jotka pysyvät samanlaisia näkymästä toiseen. (Schlatter & Levinson, 3–7)

Laajennuksen ja perussovelluksen käyttäjäkokemukset jakavat tiettyjä aspekteja, koska ne ovat integroitu toisiinsa. Vaikka tietyt aspektit, kuten käyttöliittymä ja värit, eroavat laajennuksen ja perussovelluksen välillä hieman, ne myös jakavat toisia. Etenkin sovelluksen asentamiseen, sisäänkirjautumiseen ja rekisteröintiin liittyvät prosessit ja käytännöt ovat samat. Sovelluksen sisällä ilmenevät lievät erot helpottavat tunnistamaan laajennuksen ja perussovelluksen välillä sekä tuovat identiteettejään esille näkyvämmin.

6 POHDINTA

Lopputuotoksen osalta opinnäytetyö onnistui varsin hyvin. Vaikka tuotos jäi kehitysversioksi, se kuitenkin loi projektille hyvän pohjan, josta jatkaa kehitystä. Se myös osoitti, että projekti on mahdollista toteuttaa osana perussovellusta eivätkä perussovelluksen teknologiavalinnat olleet este laajennuksen kehitykselle.

Laajennuksen kehitys jatkuu ja lähitulevaisuudessa se olisi tarkoitus saada asiakaskumppaneiden testattavaksi kaikkine ydinominaisuuksineen. Pitemmällä tähtäimellä laajennus saatetaan myös vielä erottaa omaksi sovelluksekseen riippuen sen ja perussovelluksen kehityksen kulusta ja tarpeista. Laajennus on pitkälti toteutettu niin, ettei erottaminen olisi mahdottoman suuri este, mikäli tähän ratkaisuun tulevaisuudessa päädytään. Vanhojen komponenttien muuttamista geneerisimmiksi ja uusien komponenttien uudelleenkäytön edistämistä on hyvä jatkaa tulevaisuudessakin.

Vaikka laajennuksen toteutus ei onnistunut dynaamisella tavalla, pääasiassa iOS:n ohjeistuksen takia, se on kuitenkin varsin toimiva. Tulevaisuudessa mobiilialustat, etenkin iOS, toivottavasti siirtyvät enemmän kohti helposti ladattavia ja asennettavia lisäosia kohti, jotta sekä käyttäjien että kehittäjien kokemukset parantuisivat.

Expo itsessään osoittautui päteväksi ohjelmistokehykseksi mobiilisovellusten kehittämiseen. Se toimii erittäin hyvin etenkin, jos sovellukseen tarvittavat ominaisuudet ovat käytettävissä. Aktiivinen kehitys ja yhteisö tuo paljon tukea ja apua kehittäjille.

LÄHTEET

Apple. 2022. App Store Review Guidelines. Viitattu 26.4.2022
https://developer.apple.com/app-store/review/guidelines/

Expo Docs. n.d. Limitations. Viitattu 20.5.2022
https://docs.expo.dev/introduction/why-not-expo

Holmgren, J. 2020. Yarn 1 vs Yarn 2 vs NPM. Viitattu 25.4.2022
https://shift.infinite.red/yarn-1-vs-yarn-2-vs-npm-a69ccf0229cd

Interaction Design. n.d. User Experience (UX) Design. Viitattu 21.5.2022
https://www.interaction-design.org/literature/topics/ux-design

Laricchia, F. 2022 Market share of mobile operating systems worldwide 2012-2022. Viitattu 15.4.2022
https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/

Npm Docs. n.d. About npm. Viitattu 20.5.2022
https://docs.npmjs.com/about-npm

O'Dea, S. 2021. Global smartphone penetration rate as share of population from 2016 to 2020. Viitattu 16.4.2022
https://www.statista.com/statistics/203734/global-smartphone-penetration-per-capita-since-2005/

Olanoff, D. 2012. Mark Zuckerberg: Our Biggest Mistake Was Betting Too Much On HTML5. Viitattu 21.5.2022
https://techcrunch.com/2012/09/11/mark-zuckerberg-our-biggest-mistake-with-mobile-was-betting-too-much-on-html5

React Native Docs. n.d. Core Components and Native Components. Viitattu 20.5.2022
https://reactnative.dev/docs/intro-react-native-components

Schlatter, T. & Levinson, D. 2013. Visual Usability. Principles and Practices for Designing Digital Applications. E-kirja. Massachusetts: Elsevier Inc. 3–7
https://ebookcentral.proquest.com/lib/tampere/reader.action?docID=1158375

W3Techs. 2022. Usage statistics of JavaScript as client-side programming language on websites. Viitattu 24.4.2022
https://w3techs.com/technologies/details/cp-javascript/