# Mobiilisovelluksen toteutus frisbeegolfin puttaamisharjoitteluun React Nativella

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Riihimäen kampus, Tieto- ja viestintätekniikka, insinööri (AMK)

Kevät, 2021

Vesa Pilvinen

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto -ja viestintätekniikka Tiivistelmä

HAMK, Riihimäki

Tekijä Vesa Pilvinen Vuosi 2020

Työn nimi Mobiilisovelluksen toteutus frisbeegolfin puttaamisharjoitteluun React Nativella

Ohjaajat Toni Laitinen

TIIVISTELMÄ

Tässä toiminnallisessa opinnäytetyössä kehitettiin mobiilisovellus Android-puhelimille frisbeegolfin puttausharjoittelua varten. Tavoitteena oli tuottaa sovellus, jolla pystyy tallentamaan puttausharjoittelun tuloksia ja keräämään dataa erilaisten puttien onnistumisen todennäköisyydestä. Sovelluksen on tarkoitus helpottaa käyttäjää löytämään omat heikkoudet ja vahvuudet puttaamisessa, jotta harjoittelua voisi suunnata oikein.

Projektin lopussa sovellus oli ladattavissa Google Play -palvelusta. Lisäksi kehityksen kannalta tavoitteena oli, että sovellus on helposti julkaistavissa myös Applen iOS-järjestelmälle, ja että sitä olisi helppoa jatkokehittää.

Opinnäytetyön teoreettisessa osuudessa käydään läpi sovelluksen halutut ominaisuudet sekä niiden tekniset tarpeet. Mobiilisovelluksen kehittämistä varten luodaan teoreettinen ja tekninen pohja sekä suunnitelma. Tämän jälkeen käydään vaiheittain läpi eri prosessit, joita tämän mobiilisovelluksen kehittämiseen tarvitaan. Työn pääpaino on kertoa yleisellä tasolla, mitä eri prosesseja mobiilisovelluksen kehittämiseen liittyy. Työn lopussa arvioidaan vielä kehitysprojektin onnistumista.

Avainsanat mobiili, sovellus, mobiilisovellus, kehitys, React Native, Android

Sivut 40 sivua

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology
HAMK, Riihimäki

**Abstract**

Author Vesa Pilvinen
Year 2020

Subject Developing a mobile application for disc golf putting practice

Supervisors Toni Laitinen

ABSTRACT

In this functional thesis, a mobile application for Android phones was developed to be used for frisbee golf putting practice. The goal was to produce an application that can keep record of the results of putting training sessions and collect data on the rate of success of different styles of putts. The app was designed to make it easier for the user to find their own strengths and weaknesses in putting so that the workout sessions can be directed correctly.

The application can be downloaded from Google Play at the end of the project. In addition, the goal was to make the application easily publishable to Apple's iOS system as well, and to make it easy to further develop.

In the theoretical part of the thesis, the desired features of the application and their technical needs are reviewed. A theoretical and technical basis and a plan are created for the development of the mobile application. This is followed by a step-by-step review of the various processes that are needed to develop the application. The focus of the work is to explain on a general level what different processes are involved in the development of a mobile application. Lastly, at the end of the thesis, the success of the development project is evaluated.

Keywords mobile, application, mobile application, development, React Native, Android
Pages 40 pages

## Sisällys

## Kuvat, taulukot ja kaavat

## Sanasto

| | |
|---|---|
| Android | Googlen kehittämä käyttöjärjestelmä, jota suurin osa puhelinvalmistajista käyttää puhelimissaan |
| API | Application Programming Interface. Palvelurajapinta, jota voidaan kutsua sovelluksesta. |
| Autentikaatio | Tarkoittaa tunnistautumista palveluun |
| CI/CD | Continuous Integration / Continuous Delivery tai Deployment (Jatkuva integraatio/Jatkuva kehitys). Tarkoittaa sovelluskehityksessä päivitysten ja muutosten automatisoitua testaamista ja automatisoitua julkaisuprosessia. |
| Commit | Commit on Git:ssä oleva ominaisuus, joka tarkoittaa yhtä pakettia muutoksista. Commit on yleensä kommentoitu, josta tunnistaa, mitä muutoksia mikäkin commit pitää sisällään. |
| Git | Suosittu versionhallinta järjestelmä. |
| Id | Id:llä tarkoitetaan jotain tunnusta. Yleensä joko jokin luku tai merkkijono. |
| iOS | Applen kehittämä käyttöjärjestelmä, jota käytetään Applen valmistamissa puhelimissa. |
| JavaScript-kirjasto | Liitettävä kokoelma valmiiksi luotuja JavaSript-toiminnallisuuksia. Kirjastot itsessäänkin saattaa hyödyntää myös muita JavaScript-kirjastoja. |
| Kehityskehys/Framework | Kehityskehys-sanaa käytetään kuvaamaan kokonaista kehitystyökalukokonaisuutta, joka yleensä sisältää tarkoin |

| | |
|---|---|
| | määritellyn subjektiivisen tavan työskennellä. JavaScriptin kohdalla kehityskehys koostuu yleensä useista eri kirjastoista. Se voi pitää sisällään myös erilaisia rakennustyökaluja ja esimerkiksi oman konsolisovelluksen. Esimerkkinä Ionic tai React Native ovat kehityskehyksiä |
| Lähdekoodi | Sovelluksen kaikki ohjelmoitu sisältö, logiikka ja toiminnallisuus. |
| NoSQL-tietokanta | SQL-tietokannalla viitataan yleensä relaatiotietokantoihin kuten MariaDB, PostgreSQL tai MySQL. Relaatiotietokannoissa tietotauluja voidaan linkata toisiinsa erilaisilla suhteilla. NoSQL-tietokannan tietomalli muistuttaa enemmän JSON-rakennetta. Esimerkkejä NoSQL-tietokannoista: SQLite, MongoDB ja Firebasen tietokannat. |
| Objekti | Objektilla viitataan tässä opinnäytetyössä JavaScriptissä ja JSON:ssa esiintyvään avain / arvopareihin perustuvaan tietotyyppiin. Yhdellä avaimella on aina jokin yksi arvo, joka voi olla useaa eri tyyppiä. Avaimen arvona voi olla myös toinen objekti. |
| UI | User Interface. Käyttöliittymä |

# 1 Johdanto

Frisbeegolf on peli, joka on melko samanlainen kuin golf. Yleensä kisassa pelataan rata, jossa on 18 väylää. Jokaisella väylällä on aloitusheittopaikka ja kori. Pelin tarkoitus on saada heitettyä kiekko mahdollisimman vähin heitoin koriin. (PDGA, 2021a). Frisbeegolfissa puttaamisella tarkoitetaan puttaustyylillä heitettyä heittoa, joka yritetään heittää koriin. Teknisesti ottaen puttaamisella tarkoitetaan alle kymmenen metrin päästä koria heitettyä heittoa. Kun putataan alle kymmenestä metristä, pätee puttaussäännöt, jolloin pelaajan tulee pystyä säilyttämään tasapaino heiton jälkeen ja tukijalan viereisen jalan tulee pysyä tukijalan takana tai tasalla koko heiton ajan ja sen jälkeen. (PDGA, 2021b). Tässä opinnäytetyössä puttaamisella tarkoitetaan nimenomaan puhekielessä esiintyvää tarkoitusperää, koska putteja on hyvä harjoitella myös kauempaa.

Tässä opinnäytetyössä kehitetään frisbeegolfin puttausharjoitteluun mobiilisovellus, joka on projektin lopussa ladattavissa Googlen Play-kaupasta. Sovelluksessa on tarkoitus pystyä keräämään harjoituskerroista dataa käyttäjälle, jolloin käyttäjälle voidaan graafisesti visualisoida, mitkä asiat putissa onnistuu ja mitkä eivät. Tarve sovellukselle nousee omasta kokemuksesta. Monissa muissa sovelluksissa on tiettyjä puutteita, jotka haluan korjata omaan puttausharjoittelurutiiniini. Erityisesti tulosten merkkaaminen tulisi olla mahdollisimman nopeaa ja intuitiivista puttausrutiinin ohella, joten käyttöliittymässä panostan erityisesti siihen. Pyrkimyksenä on, että sovellusta on mahdollisimman helppo jatkokehittää ja tarvittaessa mahdollista julkaista myös Applen laitteille. Opinnäytetyössä keskityn kertomaan mobiilisovelluksen kehittämisen erilaisista prosesseista valmiin Google Playstä ladattavan sovelluksen luomiseksi.

Opinnäytetyön teoriaosuudessa perehdytään sovelluksen haluttuihin ominaisuuksiin ja sitä kautta teknisiin tarpeisiin. Otan kantaa erilaisiin kehitysmalleihin ja tapoihin toteuttaa mobiilisovellus. Toteutusvaiheessa pureudutaan tarkemmin kehitysprosessin eri vaiheisiin.

# 2 Sovelluksen kuvaus ja käytetyt teknologiat

## 2.1 Kehitysteknologian valinta

Käymällä lävitse sovelluksen tarvittavat ominaisuudet ja mahdolliset tulevat ominaisuudet saadaan paras mahdollinen kuva siitä, mitä teknisiä tarpeita sovelluksella on. Pyrin tekemään sellaisia valintoja, ettei se aiheuta lisätöitä. Jos esimerkiksi valitsee liian muuttuvia tai uusia JavaScript-kirjastoja, voi sovellukseen herkästi tulla rikkovia päivityksiä ja muutoksia liian usein. Suosin hieman vanhempia ja suosittuja kirjastoja, joilla on jo mahdollisimman suuri joukko käyttäjiä ja hyvät dokumentaatiot.

Sovellus julkaistaan aluksi vain Android-laitteille, mutta kehitystyö on tarkoitus tehdä siten, että tulevaisuudessa sovellus on helposti kehitettävissä myös Applen laitteille. Androidin oma Java-pohjainen kehitysympäristö ei siis ole vaihtoehtona kovin houkuttava, koska siinä kehitetään ainoastaan Androidia varten. Kehitysalustaa valittaessa suurimpana kriteerinä on se, että koodin täytyy toimia sekä Android-puhelimille, että iOS-puhelimille ainakin lähes sellaisenaan.

### 2.1.1 Miksi React Native ja Expo

React Native on Facebookin kehittämä avoimen lähdekoodin työkalu mobiilisovellusten luomiseen. Se yhdistää natiivien mobiilisovellusten kehityksen ja Reactin, joka on yksi käytetyimpiä ja suosituimpia JavaScript-kirjastoista käyttöliittymien rakentamiseen web-sovelluksissa. React Nativessa puhelimien natiiveista UI-komponenteista on tehty React-komponentteja. Niitä voi käyttää samalla tavalla, kun React-komponentteja muutenkin, mutta taustalla ne käyttävät puhelimien natiiveja komponentteja. Se toimii siis tietynlaisena linkkinä natiivikomponenttien ja Reactin välillä. (React Native, 2020).

Koska React Native käyttää taustalla natiiveja komponentteja, on luotujen sovellusten suorituskyky lähes yhtä hyvä kuin natiiveilla sovelluksilla. React Nativella pitää kuitenkin olla tarkkana, mitä toiminnallisuuksia on käytettävissä laitekohtaisesti. Tätä varten on kuitenkin olemassa lisätyökaluja kuten Expo, jotka auttavat vielä yhtenäistämään iOS- ja Android-

kehitystä React Nativella. Tunnettuja esimerkkejä React Nativella kehitetyistä mobiilisovelluksista ovat esimerkiksi Instagram ja Facebook (React Native, 2021c).

Expo on kehityskehys, joka tarjoaa työkalut siihen, että voidaan rakentaa React Nativea hyödyntäen sovellus iOS:lle, Androidille ja Web-sivustolle. Expo pyrkii helpottamaan kehitystä niin, että yhdellä lähdekoodilla saadaan toimimaan koodi monelle alustalle. Koska minulla on entuudestaan React- ja web-kehitystaustaa ja React Nativea kirjoitetaan web-kehityksessä tutuilla tekniikoilla kuten HTML, JavaScript (React) ja CSS, ovat Expo ja React Native luonteva valinta kehityskehykseksi tähän projektiin.

### 2.1.2 Muut vaihtoehdot

Flutter on Googlen valmistama työkalu, jolla kehitetään sovelluksia useammalle alustalle. Taustalla Flutter-sovellukset hyödyntävät valmiiksi luotuja komponentteja, jotka käyttävät käännettynä laitteiden natiiveja ominaisuuksia. Flutter käyttää ohjelmointikielenään Googlen kehittämää Dart-kieltä, joka kääntyy joko suoraan ARM & x64 -suorittimen koodiksi, tai JavaScriptiksi web-sovellusta varten. (Flutter, n.d.-a). Aivan kuten React Nativellakin, Flutterilla suorituskyky on erinomainen, sillä se käyttää taustalla puhelimien natiiveja ominaisuuksia. Flutterin ainoana haittapuolena tässä projektissa on se, että minun olisi opeteltava uusi ohjelmointikieli. Esimerkiksi sovellus The New York Times on luotu käyttäen Flutteria (Flutter, n.d.-b).

Toisena potentiaalisena vaihtoehtona kehityskehykseksi olisi ollut Ionic. Se on Ionicin kehittämä usean alustan mobiilikehitykseen tarkoitettu kehitysympäristö. Kuten React Native ja Flutterkin, Ionic tarjoaa työkaluja ja palveluita useamman alustan sovellusten kehittämiseen käyttäen web-teknologioita kuten HTML5, CSS ja JavaScript (Ionic, 2021a). Ionicissa etuna on se, että sen kanssa yhdessä voidaan käyttää lähes mitä tahansa suosittua JavaScript-kirjastoa, kuten esimerkiksi Angular, React tai Vue. Puhelimessa Ionic-sovellukset käyttävät hybridiratkaisuna WebViewiä sekä joitakin puhelimen natiiviominaisuuksia. WebView tarkoittaa sitä, että sovellukset pyörivät käytännössä selaimen kaltaisessa ympäristössä eikä niinkään natiiveina sovelluksina (Ionic, 2021b). Tämä tarkoittaa sitä, että sovellusten suorituskyky ei pääse ihan samanlaisiin lukemiin kuin natiivi-, React Native-, tai Flutter-sovellukset, jotka käännetään lopulta natiiveiksi sovelluksiksi ilman WebViewiä.

Esimerkkejä tunnetuista sovelluksista, jotka on kehitetty Ionic-ympäristössä ovat esimerkiksi 86 400, Sworkit ja Instant Pot (Ionic, 2021c). Lopulta kehityskehyksen valinta tapahtui web-kehitystaustan vuoksi Ionicin ja React Nativen välillä. Haluan kuitenkin kehitystapojen lisäksi priorisoida myös suorituskykyä, joten React Native on sopivampi valinta.

## 2.2 Harjoituskertojen tallentaminen ja lukeminen käyttöliittymältä

Harjoituskertojen tulokset tulee pystyä tallentamaan, sillä tallennuksista muodostuva data luo koko sovelluksen tietopohjan ja tarkoituksen. Tästä datasta muodostetaan erilaisia kuvaajia ja päätelmiä siitä, mikä puttaamisen osa-alue sujuu ja mikä ei. React Native ja Expo tarjoavat valmiita ratkaisuja puhelimeen tallentamista varten. Koska haluan kuitenkin datan säilyvän käyttäjälle siinäkin tapauksessa, että käyttäjä esimerkiksi vaihtaa laitetta ja lataa sovelluksen uudelleen, tulee säilytyksen tapahtua ulkoisessa tietokannassa. Koska käyttäjiä on useita, tarvitaan tietokantaan oma käyttäjätunnus ja sitä kautta sovellukseen tunnistautuminen.

Näitä tarpeita pohtiessa vertasin Amazonin AWS-palveluita ja Googlen Firebase-palvelua. Molemmista löytyy kaikki oleelliset ominaisuudet tallentamiseen ja tunnistautumiseen. Ne ovat pienillä datamäärillä ilmaisia. Valitsin Googlen Firebasen lähinnä teknologioiden yhteyttämiseksi, sillä sovellus ladataan lopulta Google-tunnuksen avulla Google Play -palveluun. Lisäksi Expolla on oma implementaatio ja dokumentaatio Firebasen käyttöä varten, joten se varmasti ainakin toimii valittujen teknologioiden kanssa (Expo, n.d.-g).

### 2.2.1 Firebase, Realtime Database ja Authentication

Firebase on sovelluskehitykseen luotu työkalu Googlen pilvialustassa. Google suurikokoisena yrityksenä pystyy takaamaan lähes varmasti yhteyden jatkuvan toimivuuden. Firebase tarjoaa kaksi erilaista tiedon tallentamiseen tarkoitettua NoSQL-tietokantaratkaisua: Realtime Database ja Firestore Database. Dokumentaatio helpottaa valitsemaan sovellukseen sopivimman tietokantaratkaisun kysymällä kysymyksiä ja ehdottamalla vastausten perusteella sopivinta tietokantaa.

Dokumentaatiossa kysytään esimerkiksi tietokannan rooli, minkälaisia operaatioita tiedolle tarvitaan, minkälainen tiedon mallinnus halutaan, kuinka korkea saatavuusaste tarvitaan, tarvitaanko offline-ratkaisuja, jos käyttäjän internetyhteys on heikko ja kuinka monta eri tietokantainstanssia tarvitaan (Firebase, 2021). Tietokannan rooliksi valitsin ”enimmäkseen tiedon synkronointia yksinkertaisilla kyselyillä”, jolloin dokumentaatio ehdotti Realtime Databasea. Tiedon operaatioihin vastasin ”muutamia gigabittejä tai vähemmän dataa, joka muuttuu usein”, johon dokumentaatio suositteli jälleen Realtime Databasea. Saatavuudessa valitsin ”ainakin 99.95 prosentin saatavuus”. Tähän dokumentaation mukaan sopii kumpi tahansa tietokannoista. Offline-kyselyjen tarpeeseen vastasin ”Harvoin tai ei koskaan”, jolloin taas molemmat tietokantaratkaisut toimivat. Viimeiseen kysymykseen vastasin, että tarvitsen vain yhden tietokantainstanssin, johon edelleen molemmat tietokantaratkaisut sopivat. Testien perusteella päädyin valitsemaan Realtime Databasen

Authentication-palvelu tarjoaa työkalut autentikoitumiselle. Autentikoitumisella tarkoitetaan käyttäjän tunnistautumista palveluun. Jokaiselle käyttäjälle generoidaan oma uniikki ID, jonka pohjalta voidaan luoda tietokantamalli ja säännöt tietokantaan niin, että vain omaan dataan päästään käsiksi. Firebase tarjoaa useita eri kirjautumismetodeja kuten Google-tunnuksilla kirjautumisen, Facebook-tunnuksilla kirjautumisen, puhelinnumerolla kirjautumisen tai esimerkiksi sähköpostilla ja salasanalla kirjautumisen. Kehitän sovellukseen aluksi vain sähköpostilla ja salasanalla kirjautumisen, mutta lisään mahdollisesti myöhemmin ainakin Google-tunnuksilla kirjautumisen, sillä lähes kaikilla Android-puhelimien käyttäjillä on Googlen tunnus jo valmiiksi puhelimeen integroituna.

Firebaseen liittyen Google tarjoaa monia muitakin työkaluja, kuten esimerkiksi Cloud Functions -toiminnallisuuden, joka toimii yhdessä Firebasen kanssa. Sillä voi luoda taustapalvelimen kaltaista logiikkaa ja API-kutsuja esimerkiksi Firebasen tietokantoihin ja autentikoitumispalveluihin. Usein kehittäjät suosivat sitä, että sovelluksen logiikkaa piilotetaan taustapalveluihin, jotta käyttöliittymän puolelta ei pystytä tekemään haitallisia asioita tai pyritään piilottamaan sellaista logiikkaa, jota ei haluta käyttäjän pystyvän selvittämään. Cloud Functionsia käyttäen voidaan luoda tietynlainen välikäsi käyttöliittymän ja tietokannan välille. Omassa tapauksessani käytän vain sisäänkirjautumismetodia ja käyttäjän datan hakua ja tallennusta, eikä minulla ole tarvetta piilottaa niihin liittyvää logiikkaa, joten en ainakaan vielä ota käyttöön tätä ominaisuutta.

Firebasen kaltaisesta sovelluskehityksestä, jossa vastuu taustapalveluiden infrastruktuurista on pilvipalvelussa, käytetään termiä serverless (Cloudflare, 2021).

### 2.2.2 Erilaisten kuvaajien piirtäminen näytölle

Koko sovelluksen idea on, että käyttäjälle voidaan tuoda tallennettu data visuaalisesti näkyviin ja suodatettavaksi. Reactille ja React Nativelle on käytettävissä laaja valikoima erilaisia svg-kirjastoja, jotka piirtävät kuvaajia datan perusteella svg-grafiikkaa hyödyntäen. Sovellukseen haluan ainakin viivadiagrammin sekä jonkinlaisen prosenttikuvaajan, joten kirjastosta olisi hyvä löytyä sellaiset.

React-native-svg on avoimen lähdekoodin kirjasto, joka käyttää sisäisesti erittäin suositun d3-kirjaston ominaisuuksia tuomaan kuvaajien piirustustyökalut helposti käytettäviksi React Native -komponenteiksi (React-native-svg-charts, n.d.). Tästä kirjastosta löytyy edellä mainitut kuvaajatyypit, joten se on luonteva valinta kuvaajien piirtämistä varten.

### 2.2.3 Tarve navigaatiolle

Käyttäjällä tulee olla mahdollisuus tarkastella aiemmin tallennettujen harjoituskertojen tuloksia monella eri tasolla (lista harjoituskerroista, yksittäisen harjoituskerran yleinen katsaus ja jokainen putti erikseen). Lisäksi tulee pystyä tarkastelemaan omaa profiilisivua, josta voi muokata omia asetuksia. Pitää myös pystyä merkkaamaan tuloksia puttaamisen aikana. Sovellus sisältää siis paljon erilaisia sivuja ja näytettäviä asioita eli tarvitaan yksinkertaista logiikkaa siitä, milloin mitäkin näytetään käyttäjälle ruudulla. Kaiken logiikan hoitaminen pelkällä JavaScript-koodilla on mahdollista, mutta silloin kaikki laitekohtaisten ominaisuuksien ja konventioiden hyödyntäminen on erittäin työlästä. Siksi käytän tähän Expon virallisesti suosittelemaa React Navigation -kirjastoa (Expo, n.d.-e). Se käyttää taustalla alustan natiiveja ominaisuuksia huomioiden esimerkiksi Android puhelimissa olevan ”takaisin”-painikkeen toiminnallisuuden.

Navigaatiota voidaan hyödyntää myös jäsentelemään tunnistautumisprosessia. Sovellus erotellaan tunnistautumattomaan puoleen ja tunnistautuneeseen puoleen kuten React Navigation -kirjaston dokumentaatiossa kuvataan (React Navigation, n.d.).

Tunnistautumattomalla puolella on vain muutamia eri navigaatiosivuja kuten sisäänkirjautumisen sivu ja käyttäjän luomiseen tarkoitettu sivu. Koko muu käytettävä sovellus sijaitsee tunnistautuneella puolella.

## 2.3 Sovelluksen lokalisointi

Koska haluan sovelluksen mahdollisesti olevan käytettävissä kansainvälisesti, niin sovelluksen teksteille olisi hyvä olla helppo lisätä uusia käännöksiä. Lisäksi esimerkiksi päivämäärät ja yksiköt olisi hyvä näyttää lokalisoidusti käyttäjän laitteen lokalisaatioasetuksen tai sovelluksen asetuksen perusteella. Sovelluksen asetuksista tehtävä kieli- ja yksikkövalinta jää jatkokehitettäväksi ominaisuudeksi, eikä siis sisälly tämän opinnäytetyön prosesseihin.

Käännöskirjastoksi valitsen erittäin yleisen ja suositun i18n kirjaston, jota käytän yhdessä Expon Localization -paketin kanssa, kuten dokumentaatiossa ohjeistetaan (Expo, n.d.-b). Käännöskirjasto mahdollistaa sen, että riippuen laitteen tai sovelluksen valitusta kielestä sovellus lataa eri tiedoston, josta kaikki sovelluksen tekstit ladataan. Jokainen kielitiedosto on rakenteeltaan samanlainen JSON-tiedosto. Tiedosto sisältää avain-arvo-pareja, jossa arvo on tietty käännös. Koodissa käännöksiin viitataan aina avaimen kautta. Kaikkien käännöstiedostojen avaimet ovat siis identtiset, mutta arvot eroavat. Ensisijaisesti ei ole välttämätöntä, että sovelluksessa on useampi kielitiedosto. Oleellista on, että sovelluksen tekstien on tultava käännöstiedoston kautta, jotta käännöksiä on helppo lisätä. Työssäni toteutan sovelluksen ensisijaisesti englanninkielisenä, jotta se olisi heti mahdollisimman laajasti käytettävissä.

## 2.4 Projektinhallintaan liittyvät valinnat

### 2.4.1 Versionhallinta

Sovelluksen koodin kirjoittaminen tapahtuu lokaalisti omalla tietokoneella. Siihen liittyy kuitenkin useita eri riskejä. Mitä jos useiden päivien työn jälkeen tietokone hajoaa? Mitä jos teen virheellisen toiminnallisuuden ja haluan palata edelliseen versioon? Tallennanko koko sovelluksen aina uuteen versiohakemistoon, jotta edelliseen voi palata ongelmatilanteessa ja

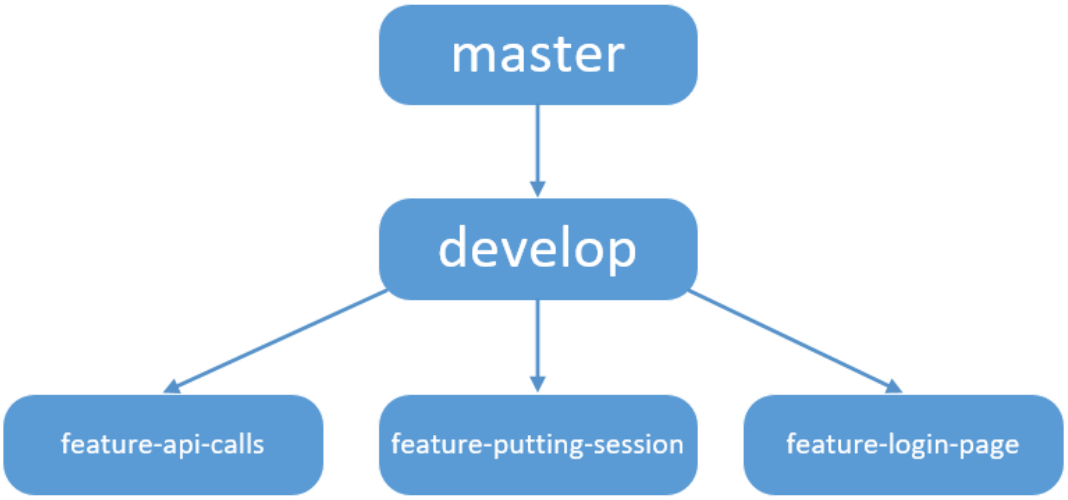
vienkö nämä kaikki erikseen myös verkkolevylle? Näiden ongelmien ratkomisessa on hyötyä versionhallintajärjestelmästä. Eräitä suosittuja versionhallintajärjestelmiä ovat esimerkiksi SVN ja Git.

Versionhallintaan hyödynnän erittäin suosittua Git:iä. Kuten muutkin versionhallintajärjestelmät, myös Git mahdollistaa sen, että versiohistoriassa voi palata mihin tahansa aiempaan versioon koodista. Tämä tuo turvaa ongelmatilanteessa, jossa tehdään uusi sovelluksen rikkova päivitys. Silloin voidaan palata vanhaan toimivaan versioon ja työstää päivitystä erillään uudelleen. Versionhallinta Git:ssä noudattaa puurakennetta (Kuva 1). Projektin päähaaraa tai -versiota eli juurta kutsutaan usein master-haaraksi. Master-haarassa on usein tuotannon tasossa oleva lähdekoodi, josta voidaan jakautua esimerkiksi kehitykselle omaan haaraan. Haaroja voi yhdistellä halunsa mukaan merge-ominaisuudella, jolloin Git-järjestelmä osaa lisätä tai muuttaa vanhaan koodiin uudet, tai muutetut koodirivit. Tämä haaroihin jakaminen mahdollistaa esimerkiksi myös sen, että sovelluksen koodista voidaan julkaista useampaan haaraan eri versio. Esimerkiksi testipalvelinversio, tuotantopalvelinversio ja kehittäjän versio voivat sijaita omissa erillisissä haaroissaan.

Tämä ei vielä ratkaise ongelmaa siitä, että projektia työstetään lokaalisti omalta tietokoneelta ja tiedostot voivat tuhoutua tietokoneen rikkoutuessa. Git:iin on rakennettu ominaisuus olla yhteydessä etäpalvelimella sijaitsevaan Git-projektiin eli Git-repositoryyn. Useimmiten Git-projektien hallinnointiin käytetäänkin pilviratkaisua kuten GitLab tai GitHub, jolloin voidaan turvata, että lähdekoodi sijaitsee keskitetysti palvelimella. Merge-toiminnallisuuden kanssa tämä helpottaa lisäksi useamman ohjelmoijan työtä siten, että jokainen voi työstää samaa keskitettyä projektia samanaikaisesti, koska Git osaa tunnistaa muutokset jokaiselta tekijältä erikseen ja yhdistää muutokset oikein. Sijoitan Git-projektin GitLab-palvelimelle omaan repositoryyn, jonka nimeän sovelluksen nimen mukaan DGPP (Disc Golf Putting Practice). Sinne voidaan määritellä myös automatisointia, joka on kytkettynä versionhallintaan. Voidaan esimerkiksi suorittaa koodille haluttuja komentoja, kuten testien ajaminen jokaisen tietyn haarapäivityksen myötä. GitLabin valitsen siksi, että sinne on ilmaista luoda privaatti projekti, kun taas GitHubissa sellaisen luomiseen tarvitaan maksullinen tunnus.

Noudatan työskennellessä kuvan 1 mukaista haarautumista. Master-haarasta on otettu kopio develop-haaraan. Develop-haarasta otetaan jokaista ominaisuuden kehittämistä varten kopio omaan nimettyyn haaraansa. Kun ominaisuus on kehitetty valmiiksi, voi sen haaran yhdistää takaisin develop-haaraan. Kun sovellus on testattu ja todettu toimivaksi develop-haarassa, yhdistetään develop-haara takaisin master-haaraan eli tuotantoon.

Kuva 1. Versionhallintajärjestelmän haarojen puurakenne.




### 2.4.2 Ohjelmakoodin tarkastaminen ja testaaminen hallinnoidusti

Yksi sovelluskehityksen tärkeistä osa-alueista on lähdekoodin testaaminen. Testaamisella tarkoitetaan ohjelmakoodin osille, kuten esimerkiksi funktioille tai komponenteille kirjoitettuja testejä erilaisilla syötteillä. Testit itsessään kirjoitetaan myös yleensä samalla ohjelmointikielellä, kun testattava ohjelmakoodi. Tässä tapauksessa kielenä on TypeScript. Testit voidaan suorittaa esimerkiksi aina ennen ohjelmakoodin kääntämistä sovellukseksi. On olemassa erilaisia kohdennettuja testejä, joita voidaan kirjoittaa. Tähän sovellukseen kirjoitan yksikkötestejä ja snapshot-testejä. Yksikkötestaamisella tarkoitetaan sitä, että kohdennetaan testaaminen tiettyyn pieneen yksikköön, joka voi olla funktio tai komponentti (React Native, 2021b). Esimerkiksi laskennallinen funktio on hyvä yksikkötestauksen kohde – palauttaako funktio halutunlaisen arvon annetulla testisyötteellä. Näissä voidaan tarjota erilaisia ääritapauksia syötteen arvoihin testatakseen, että ohjelmakoodi toimii myös niissä tapauksissa. Esimerkiksi puttaamisen osumaprosentin laskemisessa tuloksen tulee kaikissa tapauksissa olla välillä 0–100. Yksikkötestien piiriin kuuluu myös esimerkiksi nappien toiminnallisuuksien testaaminen.

Snapshot-testaamisella testataan, että komponentin tulostus pysyy yhtenäisenä kerrasta toiseen. Testi tulostaa komponentin ja kaappaa siitä snapshot-kuvan, jota verrataan edellisen snapshot-testin tallennettuun snapshot-kuvaan. Snapshot-kuva on ”.snap”-päätteinen tiedosto, joka sisältää Reactin luoman virtuaalisen DOM-puun (Document Object Model). Testi menee läpi, mikäli tiedostojen sisällöt ovat keskenään samanlaiset. Tämä tarkoittaa sitä, että Snapshot-kuvat tulee päivittää aina, kun komponentteihin tulee muutoksia, jotka aiheuttavat komponentin tulostuvan eri tavalla. Tällä pyritään estämään ei-haluttujen komponenttimuutosten läpi pääseminen. (React Native, 2021a).

### 2.4.3 Koodinlaaduntarkastuspalvelu

Käytän projektissa koodin tarkistamiseen Sonarcloudin tuottamaa laaduntarkistuspalvelua. Sonarcloud on yhteydessä suoraan versionhallintaprojektiin GitLab:ssa, ja se tutkii projektikoodin lävitse, ja etsii ohjelmakoodista yleisiä rakenteellisia ongelmia ja raportoi niistä kehittäjälle. Lisäksi Sonarcloud tarkastaa koodista esimerkiksi testien kattavuusprosentin. Pyrin seuraamaan Sonarin tuottamaa laaturaporttia, ja pitämään kaikki mittarit hyvällä tasolla (A-taso raportissa).

## 2.5 Sovelluksen tilan hallinta Reduxilla

Sovellus koostuu komponenteista, jotka ovat puurakenteessa, kuten esimerkiksi versionhallinnan haarat (Kuva 1). Komponenteilla on tietoja, jotka vaikuttavat siihen, mitä ne tulostavat ruudulle. Kun puhutaan sovelluksen tai komponentin tilasta, puhutaan tietojen arvosta kullakin sovelluksen käyttämisen hetkellä. Reactissa normaalisti välitetään tietoja puuhierarkiassa alaspäin komponentilta seuraavalle. Jos jotakin tietoa tarvitaan useammassa komponentissa, tietojen välittäminen komponentilta toiseen puurakenteessa saattaa olla kuitenkin melko haastavaa, kun sovelluksen komponenttien hierarkia on tarpeeksi monimutkainen. Siksi monesti käytetään jotain yleistä varastoa tiedoille, josta komponentit voivat seurata ja päivittää tietoja. Tähän käyttöön on kehitetty useita eri työkaluja, joista valitsin käyttöön Redux-kirjaston.

Reduxissa ajatuksena on luoda yleinen varasto tiedolle, joka strukturoidaan halutusti, esimerkiksi ominaisuuksien mukaan. Komponentit voivat kuunnella varastosta tiettyä

haluttua osuutta (reducer), ja poimia muuttuvia tietoja. Kun tiedot yleisessä varastossa päivittyvät, kaikki tietoa kuuntelevat komponentit sovelluksessa tulostuvat tarpeen mukaan uudelleen uusilla tiedoilla. Redux-varastoa on mahdollista käskyttää komennoilla (action). Kun actionia kutsutaan, reducer kuulee sen ja suorittaa tilan päivityksen kyseessä olevaan tieto-osioon halutulla tavalla. (Redux, n.d.-b).

Reduxin käyttöön hyödynnän Redux Toolkit-kirjastoa, joka helpottaa Reduxin kanssa työskentelyä erilaisilla työkaluilla, ja jossa määritellään yleisiä käytäntöjä Reduxin käyttöön (Redux Toolkit, n.d.).

## 2.6 Ohjelmoinnin aikainen projektin hallinta ja lähdekoodiprojektin rakenne

### 2.6.1 TypeScript, TSLint ja Prettier

Käytän ohjelmointiin JavaScriptin sijaan TypeScriptiä, joka on JavaScriptin päälle rakennettu laajennus. TypeScript tuo JavaScriptiin useista muista kielistä tutun tyypityksen. Eli muuttujalle voidaan määritellä esimerkiksi, onko kyseessä merkkijono, numero tai esimerkiksi tietyn muotoinen objekti. Lopuksi kun koodi rakennetaan, se käännetään normaaliksi JavaScriptiksi. Vahvempi tyypitys tekee tuotantoon menevästä koodista vähemmän virhealtista, koska tieto mahdollisista tyyppivirheistä saadaan jo koodia käännettäessä JavaScriptiksi. Laskentafunktiota ei voida kutsua parametrilla, joka on merkki jono, jos parametrin tyyppi on funktiolle määritelty numeroksi. TypeScriptin rinnalla voidaan käyttää myös ohjelmoinnin aikaista sääntötarkistusta. Tähän hyödynnän TSLint-työkalua. Se tarkistaa jatkuvasti määriteltyjen sääntöjen mukaisesti, onko koodi haluttujen sääntöjen mukaista ja oikein tyypitettyä. Esimerkiksi voidaan antaa virheviesti koodieditorissa, jos koodi on jollain tapaa väärin jäsenneltyä, vaikka se olisikin muutoin toimivaa.

Koodin jäsentelyssä voidaan hyödyntää myös erilaisia formatointityökaluja. Päädyin valitsemaan tähän Visual Studio Codeen asennettavan Prettier-lisäosan. Sille voidaan määritellä erilaisia formatointiasetuksia ja kun formatointi suoritetaan, se muokkaa koodin asetusten mukaiseksi. Esimerkkinä JavaScriptissä ei välttämättä tarvitse jokaisen rivin loppuun puolipilkkua, joten se saattaa kehittäjällä unohtua joiltain riveiltä ja joiltain ei. Prettieriä voidaan käskeä lisäämään puolipilkku jokaisen ohjelmarivin loppuun tiedostoa

formatoitaessa. Formatoinnin voi halutessaan suorittaa automaattisesti esimerkiksi aina kun tehdään rivinvaihto tai kun tallennetaan tiedosto. Prettier käyttää ensisijaisesti omaa konfiguraatiotiedostoa projektihakemistossa ja jos ei sitä ole niin se käyttää lisäosan asetuksista löytyviä määrityksiä (Prettier, n.d.). Konfiguraatiotiedostossa on se etu, että jos kehittäjiä olisi useampia, muiden kehittäjien ei tarvitsisi tehdä asetuksia erikseen vaan asetukset pysyisivät kaikille samana. Kaikille kehittäjille pitää tässä tapauksessa ohjeistaa käytettäväksi editoriksi jotain Prettierin tukemista editoreista, ja että Prettier-lisäosa tulee olla asennettuna kyseiseen editoriin. Näin voidaan edelleen varmistaa sitä, että koodi on helposti luettavaa ja tyyliltänsä yhtenevää kaikilta osin, vaikka ohjelmoija itse muotoilisi ohjelmakoodin alustavasti hieman eri tavalla. Projektin kehittämisen ohjeistuksen voi lisätä esimerkiksi sovellusprojektin juurihakemistossa sijaitsevaan ”README.md” nimiseen tiedostoon, mikä on eräänlainen ohjesivu, joka näytetään GitLab-projektin etusivulla (GitLab, n.d.-a).

### 2.6.2 Julkaisuprosessi ja sen automatisointi

Aina kun uutta päivitystä luodaan projektiin, täytyy julkaisua varten suorittaa useita aikaa vieviä toimenpiteitä kuten yksikkö- ja snapshot-testien sekä Sonar-laatutarkistuksen suorittaminen. Sovelluksen lähdekoodista rakennetaan Androidiin asennettava paketti, josta sovellus voidaan asentaa puhelimeen. Rakennus tapahtuu Expon komennon avulla Expo-palvelimella ja siitä edelleen Google Play-kauppaan vienti tapahtuu Expon komennolla. Lisäksi ennen näitä vaiheita tulee aina päivittää sekä sovelluksen näkyvä versionumero että Android-versionumero ”app.json”-tiedostoon. Näiden työvaiheiden suorittaminen jokaisella pienimmälläkin päivityksellä vie melko paljon aikaa etenkin Expon rakennusprosessien vuoksi. Näiden prosessien automatisointiin hyödynnän GitLabista löytyviä CI/CD-työkaluja (Continuous Integration / Continuous Deployment). GitLabiin voidaan projektin juurihakemistossa sijaitsevaan ”.gitlab-ci.yml”-tiedostoon määritellä, että kun tietty Git-haara päivittyy, GitLab ajaa eräänlaisen komentojen suorittajan (GitLab Runner agent) halutuilla ominaisuuksilla ja suorittaa siellä halututtuja komentoja (GitLab, n.d.-b).

Tuodessa uutta päivitystä versionhallinnan master-haaraan nostan versionumeron ”app.json”-tiedostoon manuaalisesti kehitystyön ohella. Lisäksi päivitän muuttuneiden komponenttien snapshot-testit ja testaan, että kaikki testit menevät omalla tietokoneella

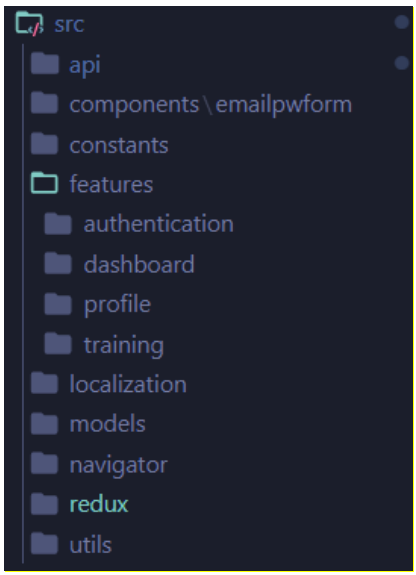
ajettuna läpi. Sen jälkeen käytän Git:n merge-toiminnallisuutta yhdistämään uuden ominaisuuden haaran kehityshaaraan. Kehityshaaran päivittyessä GitLab suorittaa snapshot-testit ja yksikkötestit vielä automatisoidusti. Kun testit ovat menneet onnistuneesti läpi, yhdistän kehityshaaran master-haaraan, josta automatisoidusti tehdään koodille vielä Sonarin laatutarkistus ja mikäli se menee läpi ilman virheitä, käytetään Expon komentoja automatisoidusti rakentamaan sovelluspaketti ja viemään se Google Play -palveluun. Näin minun ei tarvitse odotella rakennusprosessia, eikä muistaa komentoja – muistan vain nostaa versiota ja yhdistän muutokset Git-haaroista toiseen ja loput julkaisuprosessista tapahtuu automaattisesti.

## 2.7 Hakemistorakenne

### 2.7.1 Skaalautuva hakemistorakenne

Hakemistorakennetta pohdittaessa on hyvä ottaa huomioon esimerkiksi skaalautuvuus. Tässä projektissa noudatan soveltaen Redux-kirjaston suosittelemaa feature-rakennetta (Redux, n.d.-a). Tämä tarkoittaa sitä, että hakemistot on jaoteltu sovelluksen ominaisuuksien perusteella features-alihakemistoon, kuten kuvassa 2 on esitetty. Se tuo skaalautuvuutta projektiin pitäen rakenteen kuitenkin selkeänä.

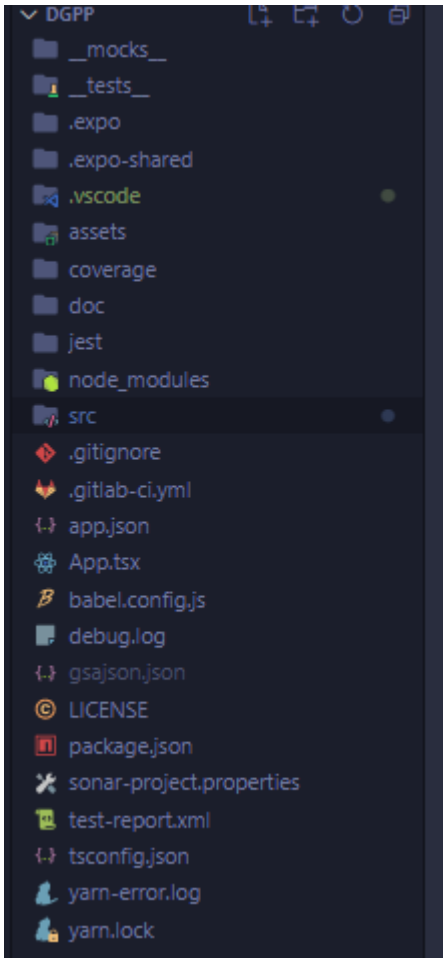
Kuva 2. Hakemiston feature-rakenne.



### 2.7.2 Oleellisimmat hakemistot ja niiden sisällöt

Juurihakemisto sijaitsee projektin juuressa ja sisältää siten koko sovellusprojektin kaikki tiedostot ja kansiot (Kuva 3). Oleellisia tiedostoja juurihakemistossa ovat ”package.json” ja ”app.json” ja ”App.tsx”. Siellä on sen lisäksi myös TypeScriptin ja esimerkiksi GitLabin CI:tä ja Sonaria varten tarvittavia erilaisia asetustiedostoja. Juurihakemisto sisältää myöskin ”src”-hakemiston, jonka alta löytyy lukuun ottamatta ”App.tsx”-tiedoston komponenttia kaikki sovelluksen käyttämät ohjelmakoodit ja React-komponentit.

Kuva 3. Juurihakemisto.




Tiedosto ”package.json” pitää sisällään määrityksen tarvituista kirjastoista ja paketeista, jotka paketinhallintasovellus Yarn tai Node Package Manager (NPM) lataa tiedostoon määritellyillä versioilla. Sinne määritellään myös erilaisia ajoskriptejä ja testien asetuksia. ”App.tsx” sisältää sovelluksen juurikomponentin. React noudattaa HTML:n tavoin puurakennetta, ja ”App.tsx” on puun runko, josta muut komponentit haarautuvat.

Tiedostossa ”app.json” on yleisiä sovelluksen rakentamiseen tarvittavia määrityksiä kuten splash-kuva (kuva joka näytetään kun sovellus käynnistetään ja ladataan tarvittavia tietoja), sovelluksen ikoni, yleinen käyttäjälle näytettävä versionumero, Android-versionumero, jotta eri versiot osataan erotella teknisesti ja esimerkiksi sovelluksen nimi (Expo, n.d.-a).

Kansio ”api” pitää sisällään rajapintakutsuihin liittyvät asiat. Siellä on Firebase-asetustiedoston lisäksi erilaisia funktioita, joilla voidaan suoraan kutsua sovelluksen Firebase-palveluita ja palauttaa tietoja esimerkiksi Realtime Databasesta halutussa formaatissa. Myös tunnistautumista varten tarvitaan kutsu Firebase-palveluihin.

Kansioon ”components” luodaan yleiskäyttöisiä komponentteja, joita useampi ominaisuus tarvitsee tai saattaa tarvita. Esimerkiksi jos haluan kustomoida oman nappikomponentin, jota käytetään useammassa paikassa, tämä on oikea hakemisto sille. Hakemistoon ”features” luodaan sovelluksen ominaisuuksille omat kansiot. Jokainen näistä pitää sisällään kyseisen ominaisuuden omat tarvittavat komponentit, joita ei tarvita muualla sovelluksessa. Navigaattorin logiikka menee myös omaan ”navigator”-hakemistoonsa. (Kuva 2.)

Käytän lokalisaatioon i18n-js-kirjastoa. Sen asetukset, ja kielitiedostot menevät ”localization”-kansion alle, joka löytyy ”src”-hakemistosta. Objektien ja datan mallinnukset eli tyypitykset sijaitsevat ”models”-hakemiston alla (Kuva 2). Esimerkiksi tietokannasta saapuvaa dataa voidaan mallintaa ja tyypittää valmiiksi käyttöliittymän puolelle. Lisäksi esimerkiksi putille ja harjoituskerralle tarvitaan omat tietotyypit. Putti sisältää tietoja kuten etäisyys, puttausasento, tuulen nopeus ja tuulen suunta.

## 2.8 Työnjaottelu tehtäviin ja työn hallinnointi

Käytin työn hallinnoinnin helpottamiseen Trello-taulua. Trello on Atlassianin kehittämä tehtävien hallintaan tarkoitettu työkalu. Tehtäviä hallinnoidaan eri sarakkeiden välillä siirrettävillä tehtäväkorteilla. Kortteja voi siirrellä sarakkeiden välillä tehtävän etenemisen mukaan. Sarakkeina voisi olla esimerkiksi ”backlog”, ”in progress” ja ”complete”. Jokaiselle kortille on myös määriteltävissä määräaika ja hälytykset määräajan puitteissa ja niihin voi

käydä kommentoimassa tehtävään liittyvistä asioista. Tähän kehitysprojektiin käytän Trello-taulua jäsentelemään työvaiheita, jotta keskittyminen osakokonaisuuksiin kerrallaan olisi helpompaa.

### 2.8.1 Testilaitteet

Sovelluksen testaamiseen kehityksen aikana käytän emuloitua Android-ympäristöä. Tämä ympäristö on Android Studion mukana tuleva lisäominaisuus, joka helpottaa sovellusten testaamista suoraan kehitysympäristön rinnalla.

Lisäksi minulla on fyysisenä puhelimena Applen iPhone 11, jolla testaan myös, että sovellus toimii iOS-käyttöjärjestelmässä kehityksen aikana ainakin välttävästi.

### 2.8.2 Kehitysympäristö

Kehitysympäristönä käytän tietokonettani, jossa käyttöjärjestelmänä on Windows 10. Ohjelmakoodi kirjoitetaan käyttäen Visual Studio Code -tekstieditoria. Lisäksi pitää olla asennettuna työkalut kuten NodeJS, Expo, Android studio. Jatkokehitystä varten saatan tehdä myös virtuaaliympäristön, jossa on Linux ympäristöön valmiiksi asennettuna tarvitut kehityssovellukset ja Git-projekti, jolloin kuka tahansa voisi ladata virtuaaliympäristön ja aloittaa kehityksen asentamatta kaikkia tarvittavia sovelluksia omalle kotikoneelleen. Opinnäytetyötä varten työstän projektia kuitenkin vain suoraan kotikoneelta ilman virtuaaliympäristöä.

# 3 Kehitysvaihe

## 3.1 Kehitystyön aloitus ja projektin runko

### 3.1.1 Työn alustus

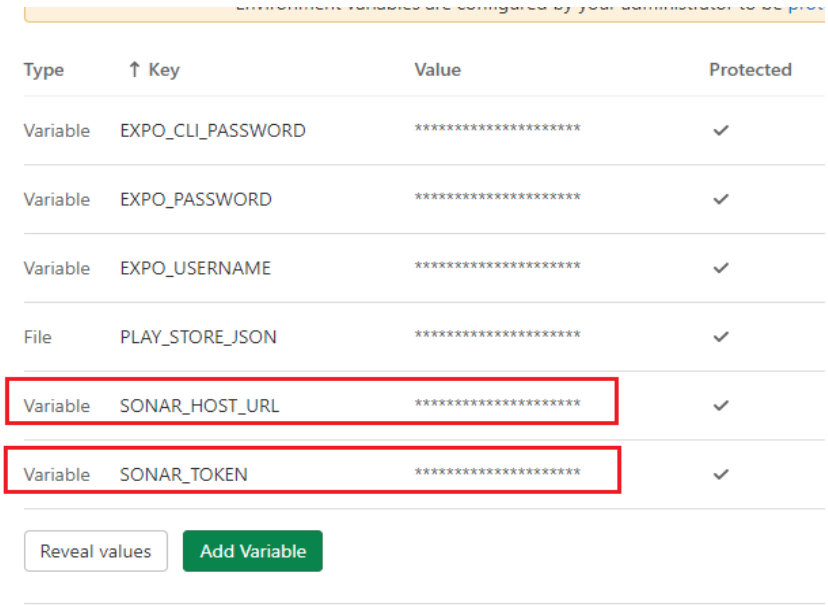
Aloitin työn asentamalla ensin kaikki tarvittavat kehitystyökalut. Näitä ovat Visual Studio Code, Expo, Git ja NodeJS. Asennus kaikkien muiden paitsi NodeJS:n kohdalla tapahtui kyseisten palvelujen virallisilta verkkosivuilta ladattavista tiedostoista, eikä näissä tullut

ongelmia. Joillain kirjastoilla voi olla tietyn Node-version kanssa ongelmia, joten haluan, että sen versiota on helppo muuttaa myöhemmin. Siksi käytin Noden asentamiseen avoimen lähdekoodin konsolisovellusta nvm-windows. Sovellus mahdollistaa useamman Node version asentamisen ja versiosta toiseen vaihdettaessa tarvitaan vain komento ”nvm use <versio>”. Nvm:llä Noden asentaminen tapahtuu komennolla ”nvm install <versio>”, ja asennettujen versioiden listaus komennolla ”nvm list” (Butler, n.d.). Aloitin työn tekemisen uusimmalla LTS node-versiolla. LTS-versio (Long Term Support) on vakaa versio johon Node:n kehitystiimi tarjoaa tukea pitkällä aikavälillä.

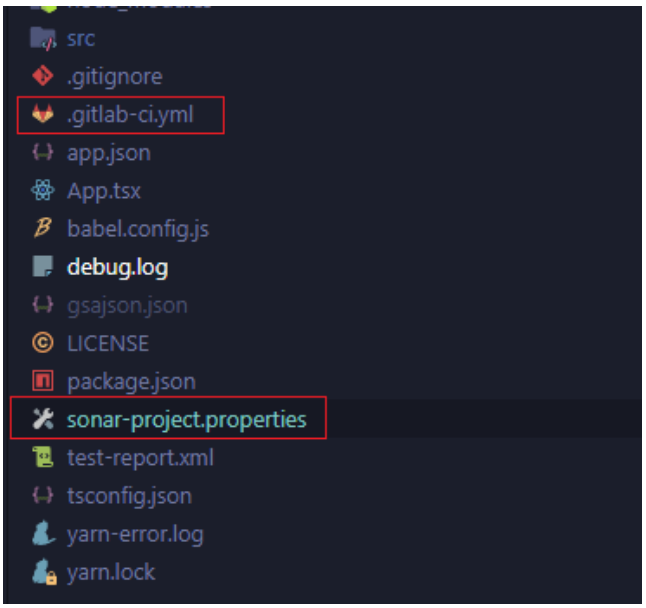
### 3.1.2 Git- ja GitLab-projektien alustaminen ja SonarCloud palvelun asetukset

Normaalisti uusi Git-projekti aloitetaan Git:n omalla ”git init” -komennolla. Loin Git-projektin kuitenkin GitLab palvelun kautta ja kloonasin kyseisen projektin eli repositoryn omalle koneelle. Seuraavaksi alustin repositoryn hakemistoon Expo-projektin Expon alustustyökalun avulla. Se tarjoaa valmiita malleja kehitykselle, mutta koska minulla on jo kokemusta React Nativesta ja Exposta, valitsin tyhjän projektialustuksen, jossa ei ole tehty valmista runkoa. Tässä työvaiheessa loin myös SonarCloud-tunnukset ja yhdistin palvelun GitLab-repositoryyn. SonarCloudin ja GitLabin omien dokumentaatioiden ohjeistamana tämä onnistui melko helposti. Tätä varten loin GitLabiin CI-konfiguraatiotiedoston YAML-kielellä. GitLabiin piti myös asettaa token-avain ja Sonar-palvelun verkko-osoite salattuina muuttujina (Kuva 4), joita SonarCloudiin yhdistäessä hyödynnettiin. Lisäksi projektin juureen piti luoda ”sonar-project.properties”-tiedosto jossa Sonar-palvelun asetukset sijaitsivat (Kuva 5).

Kuva 4. Salatut muuttujat Sonar-palvelua varten.




Kuva 5. Tarvitut tiedostot sonar-palvelua varten.



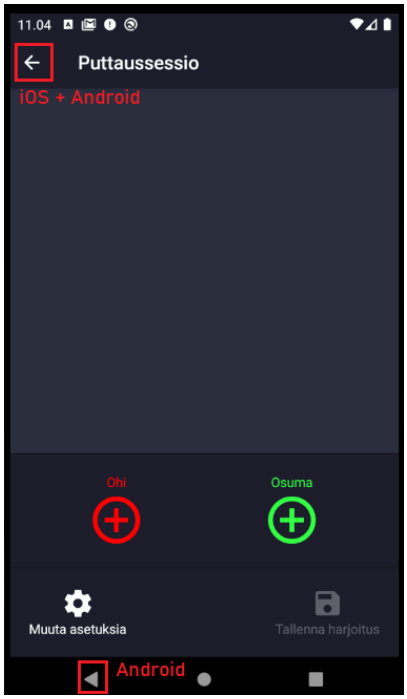

### 3.1.3 Navigaatio

Navigoinnin ja sivujen luomiseen käytin React Navigation -kirjastoa. Navigaattorin konfigurointi ja komponentit sijoitin lähdehakemiston alla sijaitsevaan ”navigator”-kansioon. Koska sovellukseen pitää kirjautua sisään loin kaksi päänavigaation tasoa: autentikaatio ja

sovellus. Autentikointipuoli pitää sisällään käyttäjän luomisen ja mahdollisuuden kirjautumiseen. Jos käyttäjä ei ole kirjautunut sisään Firebasen avulla, pidemmälle sovellukseen ei pääse käsiksi. Kaikki sovelluksen muu käytettävä logiikka sijoittuu sovelluksen puolelle. Tähän löytyi erittäin kattava dokumentaatio React Navigation -kirjaston omilta verkkosivuilta (React Navigation, n.d.).

Autentikaation puolella loin perus ”Stack Navigator” -mallin. Stack-navigointi nimensä mukaisesti pinoaa uudet sivut toistensa päälle. Uusilta sivuilta pääsee edelliselle sivulle Android-puhelimilla natiivista ”takaisin”-painikkeesta ja iOS:ssa visuaalisen nuolen kautta, joka sijaitsee ruudun vasemmassa yläkulmassa (Kuva 6). Stack-navigointi sopii tilanteisiin, joissa uusi sivu avataan usein esimerkiksi napista sovelluksessa. Sille ei ole yleensä erillistä omaa vakiopaikkaa käyttöliittymässä kuten esimerkiksi tab-navigoinnissa. Tab-navigaatiossa ruudun alaosassa on koko ajan näkyvissä visuaalinen navigaatiopalkki (Kuva 7). Autentikaation puolella kahdesta eri napista (”Kirjaudu sisään” ja ”Luo uusi käyttäjä”) avataan saman näköinen, mutta hieman eri tekstit sisältävä lomake, jossa käyttäjä syöttää sähköpostin ja salasanan joko uutta tunnusta varten, tai kirjautuakseen sisään.

Kuva 6. Laitekohtaisia navigaation eroja.



Kuva 7. Tab-navigaatiopalkki.


Loin Sovelluksen puolelle sisäkkäisen tab-stack-rakenteen (Kuva 8), jossa hyödynnetään sekä stack-navigaatiota, että tab-navigaatiota. Alapalkissa sijaitsee pääsivulla jatkuvasti navigointipalkki, jolla voi selata sovelluksen pääsivuja. Näitä ovat: Dashboard (yleiskatsaus puttausprosentteihin), Harjoitussessiot (suodatettava lista harjoitussessioista), Profiili (sisältää käyttäjän nimen ja uloskirjaus napin ja jatkossa muitakin asetuksia). Stack-navigaatio on lisätty tämän pohjalle koska kun avataan esimerkiksi ”Harjoitussessiot”-sivun listalta yksittäisen harjoituskerran tiedot, se avataan omalle sivulle, jolloin haluan, että alanavigaatio ei ole näkyvissä. Toinen sivu, jossa haluan tämän kaltaisen logiikan on itse puttaussessio. Haluan, että puttaussessio näkyy omalla ruudullaan, ja siitä ei voi poistua muuta kuin lopettamalla session. Tämän voi muuttaa navigaation hierarkian muuttamalla toisin päin, mikäli halutaankin, että voidaan poistua session aikana selaamaan muuta sovellusta.

Kuva 8. Navigaattoreiden ja sivujen hierarkia sovelluspuolella.

```
Stack navigaatio:
  Stack sivu:
    - Tab navigaatio:
      - Dashboard-sivu
      - Harjoituslistasivu
      - Profiilisivu
  Stack sivu:
    - Harjoituskerran sivu (puttisessio)
    - Harjoiutuskerran tietojen sivu
    - Harjoituskerran puttilistan sivu
    - Putin tietojen sivu
    - Profiilin editointi sivu
```

## 3.2 Firebase ja dataskeema

### 3.2.1 Tunnukset ja konfigurointi mobiilisovellukseen

Loin Googlen Firebase-palvelun verkkosivujen kautta uuden Firebase-sovelluksen. Kirjautumismetodiksi kävin asettamassa ”sähköposti ja salana” -metodin. Lisäksi loin Firebaseen sovellukselle ”Realtime Database” -tietokannan. Kirjautumista varten kävin asettamassa sähköpostilla ja salasanalla kirjautumisen päälle. Otin sivuilta ylös asetukset, jotka vaaditaan mobiilisovelluksesta Firebaseen yhdistämistä varten. Näitä tietoja ovat API-avain, autentikoinnin verkko-osoite, tietokannan verkko-osoite ja projekti-id. Normaalisti tämänkaltaiset tiedot piilotetaan ympäristömuuttujien ja versionhallinnan salattujen muuttujien avulla, mutta Firebasen omaan dokumentaatioon nojaten nämä tiedot voidaan jättää näkyville ja tietokanta tulee turvata sääntöjen avulla. Sijoitin nämä lähdehakemiston alla sijaitsevaan ”api”-hakemistoon, ”firebase.ts”-tiedostoon (Kuva 9). Loin tähän tiedostoon myös alustusskriptin, jonka voi sisällyttää muihin tiedostoihin missä Firebase-instanssia tarvitaan (Kuva 10).

Kuva 9. Firebasen asetukset asetustiedostossa.

```
import firebase from 'firebase'

const firebaseConfig = {
  apiKey: "AIzaSyD0ozdgEJMdDp6hxxGGNOS1r4pzyI8goEI",
  authDomain: "dgpp-3f275.firebaseapp.com",
  databaseURL: "https://dgpp-3f275.firebaseio.com/",
  projectId: "dgpp-3f275",
}
```

Kuva 10. Firebasen alustusskripti.

```
export const initFirebase = () => {
  try {
    firebase.initializeApp(firebaseConfig);
  } catch (err) {
    if (!/already exists/.test(err.message)) {
      console.error('Firebase initialization error', err.stack);
    }
  }
  const fb = firebase;
  return fb;
};

export const fb = initFirebase();
```

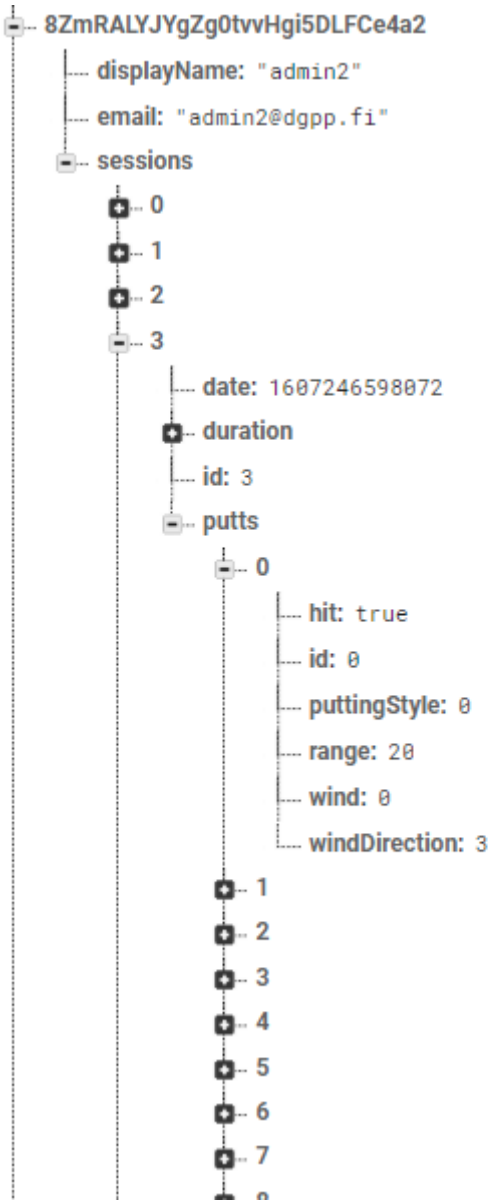
### 3.2.2 Dataskeeman muoto ja tietokannan luku- ja kirjoituisoikeuksien määritys

Tietokannan luku- ja kirjoitusoikeudet ovat oletuksena sellaiset, että kunhan käyttäjällä on Firebasen konfigurointiin tarvittavat tiedot, voi kuka tahansa lukea ja kirjoittaa tietokantaan. Aluksi testasin Firebase-yhteyden toimivuutta ilman kirjautumismahdollisuutta tallentamalla jotain yksinkertaista dataa tietokantaan sovelluksen kautta.

Data tallentuu JSON muodossa, joka voidaan sovelluksessa parsia JavaScript-objektiksi. Kuten kuvasta 11 nähdään, ylimmällä tasolla avaimina ovat käyttäjien uniikit tunnukset (uid). Jokaisen uid:n arvona on keskenään samanmuotoinen datapaketti, joka pitää sisällään nimen, sähköpostiosoitteen ja listan sessioista. Jokainen sessio pitää sisällään päivämäärän, joka tallennetaan UTC:n numeromuodossa, keston (tunnit, minuutit ja sekunnit), id:n sekä listan puteista. Jokainen putti pitää sisällään id:n, puttaustyylin, etäisyyden,

tuulennopeuden, tuulen suunnan sekä tiedon siitä, osuiko putti. Tietokantaan piti konfiguroida roolipohjaiset oikeudet lukemiselle tai kirjoittamiselle. Realtime Database tunnistaa kirjautuessa palveluun käyttäjän id:n jonka perusteella käyttäjällä on oikeus pyytää tietokannasta omat tiedot.

Kuva 11. Kuva käyttäjän datamallista.




Dataskeemaa varten tein valmiiksi lähdehakemiston alle ”models”-kansion, johon loin käyttäjän datamallille oman tyypin ”UserDataSchema” (Kuva 12). Tämä auttaa muodostamaan datan oikeaan muotoon, ennen kun esimerkiksi kutsutaan tallennusta datalle. Tämän jälkeen kävin asettamassa tietokannan sääntöihin ehdon, että avaimen

”users” alla olevan objektin, joka on nimetty käyttäjän uid:n mukaan, luku- ja kirjoitusoikeudet ovat voimassa, jos kyseinen id on sama, kuin tunnistautumissession antama uniikki id (Kuva 13). Näin varmistetaan, että käyttäjä pääsee käsiksi vain omaan dataansa.

Kuva 12. Käyttäjän datamallin tyypitys.

```
export interface UserDataSchema {
  sessions: TrainingSessionModel[];
  displayName: string;
  email: string;
}
```

Kuva 13. Tietokannan oikeuksien säännöt.

```
{
  "rules": {
    "users": {
      "$uid": {
        ".read": "$uid === auth.uid",
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

## 3.3 API-funktiot

Seuraavana työstin Firebase-palvelua kutsuvat funktiot (API-funktiot). Funktiot ovat lähdehakemistossa sijaitsevan ”api”-hakemiston alla. API (Application Programming Interface) tarkoittaa kutsuttavaa rajapintaa, joka tässä tapauksessa on Firebasen rajapinta. Oleellisia funktioita aluksi ovat: Sisään ja ulos kirjautuminen, käyttäjän luominen, session tallennus ja datan lukeminen tietokannasta.

### 3.3.1 Tallentaminen ja lukeminen tietokannasta

Tyypitin funktiot tarkasti niin, että datan muoto pysyy Firebasessa aina samanlaisena. Tietojen hakemiseen tarkoitettu ”readUserData”-funktio palauttaa joko ”UserDataSchema” muotoisen objektin, tai ei mitään (Kuva 14). Tallennusfunktiossa tyypitettiin syötettävä tieto

listaksi ”TrainingSessionModel”-tyyppisiä objekteja (Kuva 15). Tein tallennus funktion niin, että kaikki harjoituskerrat tallennetaan jokaisella kerralla, kun yksikin uusi harjoituskerta halutaan tallentaa. Harjoituskertojen id-numeroa pidetään juoksevana päivämäärän perusteella.

Kuva 14. Lukufunktio tyypitettynä.

```
export const readUserData = async (userId: string): Promise<UserDataSchema | null> => {
  try {
    const snap: UserDataSchema = await (
      await fb
        .database()
        .ref('users/' + userId)
        .once('value')
    ).val();

    return snap ? snap : null;
  } catch (error) {
    console.log(error);
  }

  return null;
```

Kuva 15. Tallennusfunktio tyypitettynä.

```
export const saveSessions = (userId: string, sessions: TrainingSessionModel[]) => {
  fb.database()
    .ref('users/' + userId + '/sessions/')
    .set(sessions);
};
```

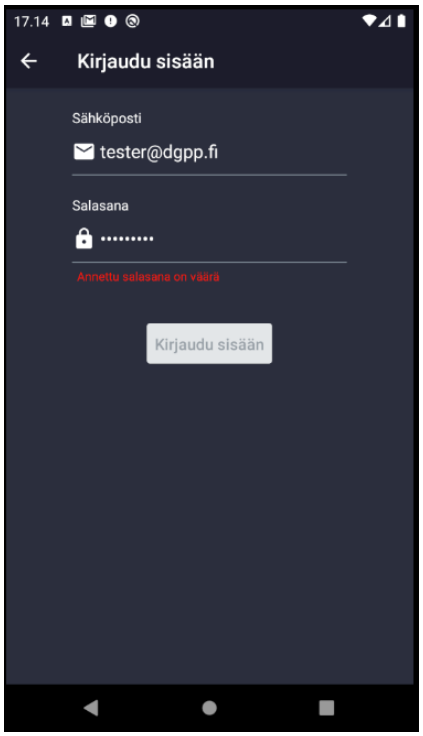
### 3.3.2 Autentikaation funktiot

Nämä funktiot ovat myös pakollisia API-funktioita, jotka täytyy saada tehdyksi, ennen kuin sovellusta voi julkaista loppukäyttäjille. Sisään kirjautuessa yritetään ensin hakea puhelimen muistista tallennettuja käyttäjätunnuksia ja koitetaan palauttaa sessio tai kirjautua tallennetuilla tiedoilla uudelleen. Mikäli niitä ei löydetä, käyttäjä ohjataan valitsemaan, haluaako hän kirjautua sisään vai luoda uuden käyttäjän. Jos tunnukset löytyvät, kirjaudutaan sisään automaattisesti. Tallennettaessa tunnuksia puhelimen muistiin käytän salattua tallennusmenetelmää, jonka Expo tarjoaa SecureStore API:n kautta (Expo, n.d.-f).

Autentikointifunktioissa tarkistin myös Firebasen dokumentaatiosta minkälaisia mahdollisia virheviestejä palautuu virheellisistä kirjautumisyrityksistä. Sillä tavalla voin poimia

virheviestit ja tuoda ne käyttäjälle jossakin muodossa näkyviin. Jos esimerkiksi käyttäjä antaa väärän salasanan, virheviesti näkyy salasana kentän alapuolella punaisella (Kuva 16).

Kuva 16. Käyttäjä antoi väärän salasanan.




## 3.4 Sovelluksen sivujen ja niiden sisältämien komponenttien luominen

Seuraavaksi kehitin sovellukseen tarpeiden mukaisen navigaatio- ja sivurakenteen. Sivujen kehittämisen yhteydessä loin sivuille tarkoitettuja komponentteja tarpeen mukaan. Käyn tässä luvussa lävitse lyhyesti mitä kukin sivu pitää sisällään.

### 3.4.1 Harjoituslistan sivun komponentit

Listasivulla on tarkoitus tarkastella listaa jo tallennetuista harjoituskerroista. Sieltä pääsee siis käsiksi dataan, jota on sovelluksen avulla tallennettu. Listalta voi valita päivämäärän perusteella harjoituskertoja ja tarkastella niiden tietoja tarkemmin. Kun valitsee harjoituskerran, siitä näytetään koko harjoituskerran yleiset tiedot ja jos haluaa, voi tarkastella vielä listaa puteista ja tarkastella jokaisen putin yksityiskohtia tarkemmin. Tämä

siis koostuu useasta näytettävästä sivusta kuten harjoituskertojen listan sivu, harjoituskerran yksityiskohtien sivu, puttilistan sivu ja sivu putin yksityiskohdista.

Tässä hyödynnän navigaatiossa jo aiemmin mainitsemaa sisäkkäistä rakennetta, jossa on sekä stack-navigaatio että tab-navigaatio (Kuva 8, s21). Tämän voi tehdä joko niin, että alanavigaatio häviää, jos tarkastellaan yksityiskohtia, tai niin, että alanavigaatio pysyy koko ajan päällä. Valitsin niin, että yksityiskohtien sivuilla alanavigaatio häviää, kunnes käyttäjä navigoi takaisin pääsivulle. Tässä tapauksessa päänavigaattorina toimii stack-navigator. Toisessa tapauksessa pitäisi päänavigaationa olla tab-navigaatio.

Lista saa datansa käyttäjälle Firebasesta ladatusta datasta, joka ladataan aina sovellukseen kirjautuessa sekä tallentaessa uutta dataa. Data ei voi muuttua sovelluksen käytön aikana muuten kuin tallennettaessa, joten muuta tarvetta hakea dataa uudelleen ei sovelluksen käytön aikana tule. Kun data on saatu ladattua tietokannasta, asetan sen Reduxin varastoon talteen, jolloin se jää koko sovelluksen tasolle hyödynnettäväksi siellä missä sitä tarvitaan. Reduxin varastoa kuunteleva komponentti päivittyy automaattisesti, mikäli varastossa oleva data muuttuu, joten tallennettaessa dataa ja haettaessa uudet tiedot Reduxiin, lista päivittyy aina automaattisesti vastaamaan sitä.

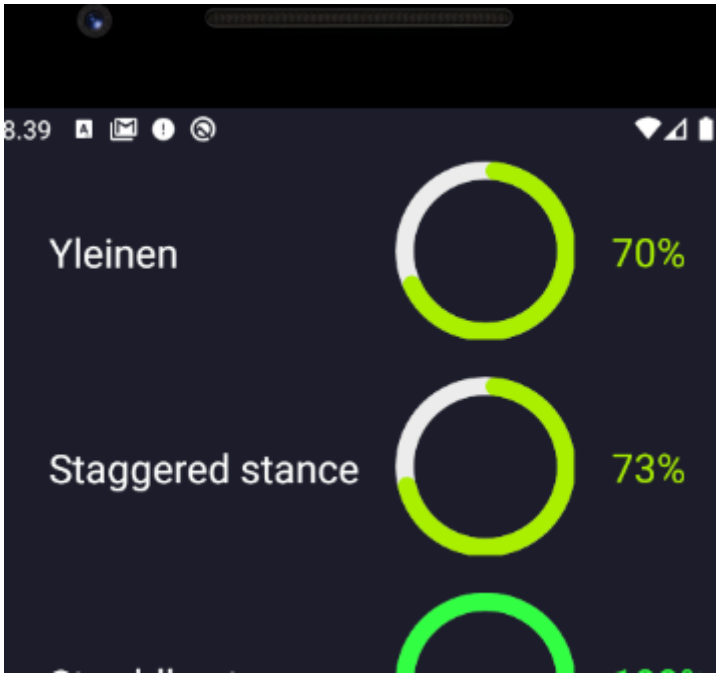
### 3.4.2 Puttisessiosivun eli harjoituskerran komponentit

Puttisessio koostuu puttien lisäämiseen ja listaukseen tarkoitetusta sivusta sekä putin asetusten muuttamiseen tarkoitetusta sivu. Kuten harjoituskertojen listasivulla, puttisessiossakin haluan, että alanavigaatiopalkki häviää, jotta käyttöliittymässä jää enemmän tilaa puttauksen aikana tarvittaviin asioihin, joten navigaatioiden hierarkia pysyy samana. Alas navigaation tilalle sijoitin kaksi ikonia. Kuten kuvasta 6 (s.19) huomataan, vasemmalle tuli putin asetuksia varten ratas-kuvake ja oikealle session päättämistä ja tietojen tallentamista varten disketti-kuvake. Toistaiseksi lisäsin ikoneille myös selitetekstit, mutta selitetekstit voisi mahdollisesti joko laittaa helppokäyttöisyysasetuksen taakse tai kokonaan pois, jotta käyttöliittymä olisi siistimpi.

### 3.4.3 Dashboard-sivun komponentit

Dashboard sivulla on tarkoitus olla näkyvillä erilaisia kuvaajia, jotka on luotu käyttäjän datan perusteella. Esimerkiksi osumaprosentteja erilaisilla asetuksilla. Jatkossa tarkoitus on myös mahdollisesti suodattaa minkä perusteella graafeja näytetään. Ensimmäistä sovellusversiota varten näytetään vain jokaisen puttiasennon osumaprosentti prosenttiympyränä (Kuva 17). Kuvaajien piirtämiseen käytän kirjastoa react-native-svg. Dashboard-sivun tarvitsee laskea prosentteja käyttäjän datan perusteella, joten tässä vaiheessa loin myös erillisen tiedoston laskentafunktioille.

Kuva 17. Prosenttiympyrä Dashboard-sivulla.




## 3.5 Testien kirjoittaminen

Aloitin kirjoittamalla snapshot-testit, koska snapshot-testit ovat hyvin lyhyitä ja yksinkertaisia kirjoittaa (Kuva 18). Snapshot-testit varmistavat, että komponentin tulostus pysyy eheänä. Snapshot-testien jälkeen kirjoitin yksikkötestejä muutamille valikoiduille asioille, kuten esimerkiksi prosenttiympyrän laskentafunktiolle (Kuva 19).

Kuva 18. Esimerkki snapshot-testistä.

```
__tests__ > App.test.tsx > ...
1   import React from 'react';
2   import App from '../App';
3   import { render, waitFor } from '@testing-library/react-native';
4
5   describe('<App />', () => {
6     it('matches snapshot', async () => {
7       const tree = render(<App />).toJSON();
8
9       await waitFor(() => {
10        expect(tree).toMatchSnapshot();
11      });
12    });
13  });
```

Kuva 19. Esimerkki yksikkötestistä.

```
describe('calculateOverallPercentage', () => {
  it('returns a number between 0 - 1', () => {
    const res = calculateOverallPercentage(mockPutts);
    const betweenZeroAndHundred = res >= 0 && res <= 1;
    expect(betweenZeroAndHundred).toBeTruthy();
  });
});
```

## 3.6 Julkaisu ja sen automatisoinnin määrittäminen

Testasin sovellusta Expo-sovelluksen välityksellä Applen iPhone 11 -puhelimella sekä Xiaomi Mi 9 -merkkisellä Android-puhelimella ja totesin sen toimivaksi halutuilta osin. Sovellus sisältää nyt kaikki tarvittavat ominaisuudet ensimmäistä julkaisua varten.

### 3.6.1 Continuous Integration

Seuraavaksi tein konfiguraatiot sille, että Git-projektin develop-haaraa päivittäessä versionhallinta suorittaa testit automatisoidusti. Tämä tapahtui kuvan 20 mukaisella konfiguraatiolla tiedostossa ”.gitlab-ci.yml”. Jatkokehityksessä tämä on hyvä laaduntarkistus uudelle koodille. Automatiikan tulisi ajaa snapshot-testit ja yksikkötestit. Kuvan 20 komento ”yarn test” suorittaa ”package.json”-tiedoston määritelmän mukaisesti komennon ”jest --

coverage”. Tämä komento suorittaa yksikkötestit ja snapshot-testit. Testien kattavuusraportin kerääminen tehdään kuitenkin edelleen manuaalisesti samalla komennolla ennen koodin vientiä versionhallintaan koska siitä muodostuu uusia tiedostoja projektihakemistoon ja Sonar käyttää raporttitiedostoja omassa analytiikassaan. Sonarin laatutarkastuksen määrittelin master-haaran päivityksen yhteyteen (Kuva 21).

Kuva 20. Testien automatisoitu ajaminen develop-haaran päivittyessä.

```
jest-tests:
  image: node:latest
  stage: test
  before_script:
    - yarn install --cache-folder .yarn
  cache:
    key: ${CI_COMMIT_REF_SLUG}
    paths:
      - .jest
      - .yarn
  only:
    - develop
  script:
    - yarn test
```

Kuva 21. Sonarin laatutarkistuksen suoritus master-haaran päivittyessä.

```
sonarcloud-check:
  stage: test
  variables:
    SONAR_USER_HOME: '$(CI_PROJECT_DIR)/.sonar' #
    GIT_DEPTH: '0' # Tells git to fetch all the b
  image:
    name: sonarsource/sonar-scanner-cli:latest
    entrypoint: ['']
  cache:
    key: '$(CI_JOB_NAME)'
    paths:
      - .sonar/cache
  script:
    - sonar-scanner
  only:
    - master
```

### 3.6.2 Sovelluksen vieminen Google Play -palveluun

Expo-sovelluksia ei ole välttämätöntä viedä Play-kauppaan, sillä ne sijaitsevat käytettävinä myös Expon palvelimilla. Tässä tapauksessa niitä käytetään Expon sovelluksen välityksellä. Lähes kaikissa Android-puhelimissa on oletuksena Googlen palvelut kuten Play-kauppa asennettuna, minkä takia lähes kaikki Android-sovellukset asennetaan sitä kautta. Jotta sovellukseni saavuttaisi mahdollisesti suuremman yleisön, on järkevämpää viedä sovellus lisäksi Googlen Play -palveluun.

Jotta sovelluksen voi ladata Google Play -palveluun, tulee omistaa Googlen kehittäjätunnukset. Tunnukset maksoivat kertaluonteisen 20 euron maksun. Määrittelin aluksi Play-konsolin kautta kaikki tarvittavat luvat ja sovelluksen asetukset ja vaatimukset kuntoon. Google on tarkka esimerkiksi henkilötietojen keräämisestä, joten pitää vastata siihen liittyvään kyselyyn. Vaatimuksena oli myös kirjoittaa tietojen keräämisen ehdot (privacy policy). Loin ehtosivuston Flycricketin työkalulla, joka kirjoitti annetuilla tiedoilla ehtoartikkelin (Flycricket, 2020).

Expon dokumentaatiosta löytyy tarkat ohjeet, miten paketti rakennetaan Googlen tai Applen sovelluskauppoihin vientiä varten. Prosessi on ensimmäisellä kerralla melko monimutkainen, mutta helpottuu sen jälkeen huomattavasti. Ensin projektin juurihakemistossa sijaitsevaan ”app.json”-tiedostoon asetetaan iOS-sovelluksen ja Android-sovelluksen rakennusasetukset, kuten versionumero, nimi, ikoni (pikakuvakkeen kuvatiedoston polku), latauskuva ja favicon-ikoni websovellusta varten. Lisäksi voidaan määritellä lupia, jotka tarvitaan puhelimelta, että sovellus on käytettävissä (esimerkiksi lupa käyttää puhelimen kameraa). Jos tätä ei määritellä, käytetään Expon omaa oletusmääritelmää luvista (Expo, n.d.-d). Luvat ovat oleellisia Play-kauppaan vientiprosessin kannalta, että sovellus tulee hyväksytyksi Googlen tarkastusprosessissa. Tässä tapauksessa erillisiä lupia ei tarvita, sillä sovellus on hyvin yksinkertainen, joten määrittelin permissions-kohdan asetuksista tyhjäksi.

Ensimmäisellä julkaisukerralla rakensin sovelluksesta Expon avulla Android-puhelimelle asennettavan paketin kuten dokumentaatio ohjeistaa (Expo, n.d.-c). Latasin sen jälkeen tiedoston Play-palvelimelle Googlen kehittäjäkonsolin kautta. Ja kun kaikki lupa- ja tarkastusprosessit Googlen puolelta olivat valmiit, sain Play-palvelun hyväksynnän ja

julkaisuluvan. Sovelluksen voi asettaa aluksi myös eri julkaisukanaviin kuten ”sisäiset testaajat”, tai ”tuotanto” Google Playn kautta. Eri kanaville voi kutsua erikseen käyttäjiä, jotka saavat oikeuden ladata sovelluksen testiversion. Omaan automatisoituun julkaisuprosessiin en sisällytä toistaiseksi erillisiä kanavia, koska kehitän sovellusta yksin ja voin testata develop-haaran versiota itse ennen tuotantoon eli master-haaraan vientiä.

### 3.6.3 Continuous Delivery

Koska haluan jatkossa automatisoida sovelluksen päivityksen Googlen Play -kauppaan käyttämällä Expon työkaluja, loin ohjeiden mukaan sovellukselle Google Service Account -tunnuksen. Tämä johtuu siitä, että tässä tapauksessa GitLabilla pitää olla oikeudet julkaista puolestani päivitys sovellukseen. Yhdistin Google Service Accountin Googlen kehittäjäkonsolin kautta sovellukseen ja latasin Google Service Accounttiin liittyvän JSON-tiedoston, joka pitää sisällään tärkeitä avaimia, joilla voi tunnistautua sovelluksen päivittämistä varten. Säilytän tätä tiedostoa erikseen sekä lokaalisti omalla tietokoneella, että GitLabissa salattuna tiedostona automaattista päivitystä varten.

Continuous Delivery tai Deployment, eli jatkuva julkaisu tarkoittaa käytännössä julkaisuprosessien helpottamista automatiikan avulla. Omassa tapauksessani projektikoodin vieminen versionhallinan master-haaraan suorittaa automatisoidun projektin rakennuksen ja päivittämisen Google Play -kauppaan. Sekä CI, että CD vaiheen automatisoidut skriptit ovat määritelty tiedostossa ”.gitlab-ci.yml”, jossa käytetään GitLabin tarjoamia komentoja määrittämään asioita. GitLabin CI/CD-palvelu luo käytännössä virtuaalisen järjestelmän, joka tarkoittaa käytännössä erittäin karsittua käyttöjärjestelmää ilman fyysistä tietokonetta (tässä tapauksessa Linux), johon projekti voidaan asentaa. Järjestelmän sisällä ajetaan haluttuja komentoja, jotka määritellään ”.gitlab-ci.yml”-tiedostossa tietyillä avainsanoilla. Omassa tapauksessa tuotantoon vienti tapahtuu kolmella komennolla, jotka näkyvät kuvassa 22 script-avaimen alla. Muuttujat $EXPO_USERNAME ja $EXPO_PASSWORD viittaavat GitLabiin

tallennettuihin salattuihin muuttujiin ja $PLAY_STORE_JSON salattuun tiedostoon, jolla saadaan GitLabille oikeudet päivittää sovellus Googlen palveluun.

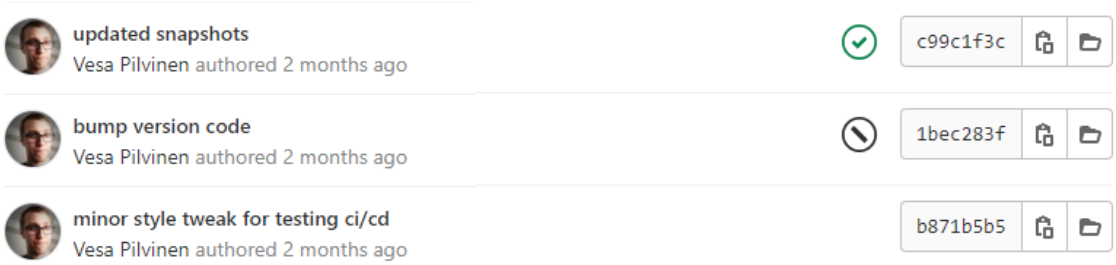
Kuva 22. Play-kauppaan vienti master-haaran päivittyessä.

```
expo-deployments:
  image: node:latest
  stage: deploy
  cache:
    key: ${CI_COMMIT_REF_SLUG}
    paths:
      - .yarn
  only:
    - master
  before_script:
    - yarn install --cache-folder .yarn
  script:
    - npx expo login -u $EXPO_USERNAME -p $EXPO_PASSWORD
    - npx expo publish --non-interactive
    - npx expo upload:android --use-submission-service --key $PLAY_STORE_JSON --latest
```

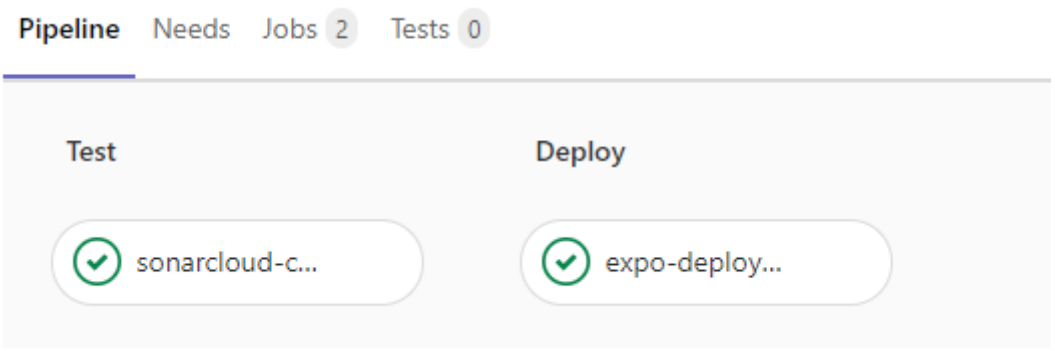
## 3.7 Jatkokehittämisen testaaminen

Testatakseni jatkokehittämisen helppoutta, tein sovellukseen tyyleihin testiksi pienen päivityksen. Tämän jälkeen vielä nostin sovelluksen versionumeroa ”app.json”-tiedostossa, sekä päivitin uudet snapshot-testit lokaalisti, koska tulostus muuttuu tyylejä muutettaessa (Kuva 23). Kuvassa 23 näkyvä vihreä merkki tarkoittaa, että CI-prosessin testien ajaminen on suoritettu onnistuneesti. Kun testit olivat onnistuneesti suoritettu, yhdistin koodin master-haaraan. Kuten kuvasta 24 näkyy, sekä Sonarin tarkastus, että Expon julkaisukomentojen ajaminen onnistui ja sovellus päivittyi automaattisesti Google Play -palveluun.

Kuva 23. Tyylipäivityksen testaaminen jatkokehittäessä.



Kuva 24. Onnistunut automatisoitu julkaisu.




# 4 Ongelmakohdat ja yhteenveto

## 4.1 Vastaan tulleita ongelmia ja ratkaisuja

### 4.1.1 Miten ratkaisin ongelmakohdat

Ehkä jopa suurin osa kehitystyöstä on ongelmien ja virheiden korjaamista. Kirjastojen ja kielien dokumentaatio pyrkii yleensä kertomaan kattavasti ominaisuuksista ja kuvaamaan kuinka niitä käytetään, mutta yhdistäessä monta erilaista teknologiaa keskenään tulee vastaan lähes aina jonkinlaisia ongelmatilanteita ja virheitä. Koska kehittäminen ja ohjelmointi on luovaa työtä ja koska kehittäjät yhdistelevät erilaisia teknologioita keskenään, muodostuu eri kehittäjillä erilaisia virhetilanteita. Siten kaikkiin ongelmiin vastaaminen dokumentaatiossa on hankalaa. Tästä johtuen kehitystyössä joutuu usein turvautumaan esimerkiksi kirjastojen keskustelupalstoihin, sillä joku muu on useimmiten törmännyt vastaavaan ongelmaan. Moniin pieniin virhetilanteisiin löytyi vastaus keskustelupalstoista hakemalla.

### 4.1.2 Testien kirjoittamisen ongelmatilanteet

Koska en ole aikaisemmin kirjoittanut montaa testiä, virheellisten testien korjaamiseen kului reilusti aikaa. Testeissä aiheutti ongelmia esimerkiksi käytetyt koodikirjastot kuten Redux. Redux käärii koko sovelluksen juurikomponentin tasolla Redux-kontekstiin, jota ei ole yksittäisessä komponentissa, jos sitä käytetään sovelluksen ulkopuolella. Tämän takia jouduin testejä varten käärimään komponentit erikseen Redux-kontekstiin. Lisäksi

esimerkiksi Firebase-ominaisuudet aiheuttivat ongelmia testeissä ja jouduin luomaan ”vääristetyn” eli mock Firebase-instanssin testejä varten, jotta komponentit toimisivat oikein testeissä.

### 4.1.3 CI/CD asetuksiin liittyvät ongelmatilanteet

Tein ensimmäistä kertaa CI/CD-määrityksiä työstäessä tätä projektia, joten tähänkin meni aika paljon aikaa. GitLabin automatisoitujen testien kanssa haastavinta oli se, että korjauksen testaaminen vie hyvin paljon aikaa. Pieninkin muutos menee koko prosessin läpi, ja virheviestin palautumiseen voi mennä useita minuutteja. Mikäli virheviestistä ei suoraan ilmene mikä on vialla, voi virheen selvittämiseen mennä useita yrityksiä ja useita tunteja tai jopa päiviä aikaa. Versionhallinnassa jopa suurin osa Git-commiteista eli projektiin versioiduista koodimuutoksista oli CI/CD-prosessiin liittyviä korjausyrityksiä.

## 4.2 Mobiilisovelluksen toteutuksen vaiheet

Tehdessä Disc Golf Putting Practice -sovellusta tutustuin käytännön kautta mobiilisovelluksen toteutuksen prosessiin ja sen eri vaiheisiin. Prosessin tämän sovelluksen osalta voi pilkkoa seuraaviin osiin:

- sovelluksen ominaisuuksien suunnittelu
- selvitys teknisistä tarpeista, jotka ilmenevät esimerkiksi ominaisuuksista ja julkaisualustasta
- muut tekniset kehitysvalinnat kuten esimerkiksi työskentely tavat tai versionhallinnan järjestelmän valinta
- sovelluskoodin ohjelmointi ja testaaminen emulaattorissa
- dokumentaatio, kuten ”README.md”-tiedosto ja omassa tapauksessani opinnäytetyö. GitLabista löytyy myös työkalut wiki-sivuston luomiseen tarpeen vaatiessa
- testien kirjoittaminen
- erilaisten tunnusten luominen mahdollisille julkaisualustoille
- sovelluksen testaaminen käytössä
- sovelluksen ensimmäinen julkaisu
- julkaisuprosessien automatisointi (CI / CD)

- jatkokehittäminen.

### 4.3 Yhteenveto sovelluksen toteutuksesta

Kehitystyö eteni hitaasti, mutta varmasti pienehköissä paketeissa yksi kerrallaan. Lopputuloksena sain tuotettua tavoitteen mukaisen sovelluksen. Sain valmiiksi oleellisia ominaisuuksia kuten harjoituskertojen selaaminen listalta, Dashboard-sivu, jossa oli kuvaajia, tulosten merkkaamisen käyttöliittymä ja alustava profiilisivu asetuksia varten. Lisäksi sovellus tunnistaa puhelimen kieliasetuksen, ja näyttää sovelluksen tekstit joko englanniksi tai suomeksi. Sovelluksen ehkä tärkein toiminnallisuus, eli puttien merkkaaminen onnistui hyvin ja käyttöliittymästä tuli oman kokemukseni mukaan intuitiivinen ja helppokäyttöinen.

Sovellus on nyt ladattavissa Google Play-palvelusta ja sen jatkokehittäminen on automatisoitujen prosessien (CI/CD) ansiosta helppoa. Sovellus toimii teknisesti hyvin ja esimerkiksi tietokantayhteys ja käyttäjän oman datan lukeminen onnistuu.

Eniten haasteita kehitystyön aikana tuotti selkeästi erilaisten konfiguraatioiden tekeminen järjestelmien välille. Esimerkiksi Sonarin yhdistäminen versionhallintaan, testien automatisoitu ajaminen versionhallinnassa ja automatisoitu julkaisu Google Play -palveluun veivät huomattavan osan koko kehityksen ajasta prosessin monimutkaisuuden vuoksi. Lisäksi testien kirjoittaminen oli hidasta, koska siitä minulla ei ollut aikaisempaa kokemusta. Sovelluksen toiminta ja logiikka olivat melko yksinkertaisia, joten muutaman vuoden web-kehittämisen kokemuksella sovelluksen ohjelmakoodin kehitys ei tuottanut ongelmia ja oli sovelluksen pienehkön koon vuoksi melko nopea toteuttaa.

## 5 Johtopäätökset

Mobiilisovelluksen kehitys on monivaiheinen prosessi ja siihen liittyy ohjelmakoodin kirjoittamisen lisäksi paljon muuta huomioitavaa. Työn tavoitteena oli tuottaa Google Play-palvelusta ladattavissa oleva sovellus halutuilla ominaisuuksilla, ja pureutua kehitysprosessin eri vaiheisiin. Lisäksi tavoitteena oli saada jatkokehittämisestä sekä Applen alustalle julkaisusta helppoa. Kaikki asetetut tavoitteet saavutettiin, ja päästiin tutustumaan prosessin eri vaiheisiin sekä yleisellä tasolla, että osittain hieman tarkemmallakin tasolla. Sovelluksesta

tuli helppokäyttöinen ja se toimi testatusti sekä Android-puhelimella, että Applen puhelimella.

Sovelluksesta jäi puuttumaan joitakin haluttuja ominaisuuksia, kuten kielen vaihtaminen asetuksista. Myös erilaisten kuvaajien lisääminen jäi vielä tekemättä. Lisäksi käyttöliittymän tyylien hienosäätäminen jäi vähälle, joten siitä löytyy vielä paranneltavaa. Sovelluksen jatkokehittäminen tehtiin kuitenkin helpoksi ja sitä testattiin tekemällä sovellukseen pieni päivitys, jolloin automatisoitu julkaisuprosessi meni läpi onnistuneesti.

## Lähteet


Butler, C. (n.d.). *Github. Nvm-windows.* Haettu 1.4.2021 osoitteesta https://github.com/coreybutler/nvm-windows

Cloudflare. (2021). *What is serverless.* Haettu 1.4.2021 osoitteesta https://www.cloudflare.com/learning/serverless/what-is-serverless/

Expo. (n.d.-a). *Documentation. App Stores.* Haettu 24.4.2021 osoitteesta https://docs.expo.io/distribution/app-stores/

Expo. (n.d.-b). *Documentation. Localization.* Haettu 1.4.2021 osoitteesta https://docs.expo.io/versions/latest/sdk/localization/

Expo. (n.d.-c). *Documentation. Manually Uploading your app for the first time.* Haettu 25.4.2021 osoitteesta https://docs.expo.io/distribution/uploading-apps/#manually-uploading-your-app-for-the-first

Expo. (n.d.-d). *Documentation. Permissions.* Haettu 25.4.2021 osoitteesta https://docs.expo.io/versions/latest/config/app/#permissions

Expo. (n.d.-e). *Documentation. Routing & Navigation.* Haettu 1.4.2021 osoitteesta https://docs.expo.io/guides/routing-and-navigation/

Expo. (n.d.-f). *Documentation. Secure store.* Haettu 24.4.2021 osoitteesta https://docs.expo.io/versions/latest/sdk/securestore/

Expo. (n.d.-g). *Documentation. Using firebase.* Haettu 1.4.2021 osoitteesta https://docs.expo.io/guides/using-firebase/

Firebase. (3.5.2021). *Documentation. Choose a database.* Haettu 1.4.2021 osoitteesta https://firebase.google.com/docs/database/rtdb-vs-firestore/

Flutter. (n.d.-a). *Etusivu.* Haettu 27.12.2020 osoitteesta https://flutter.dev

Flutter. (n.d.-b). *Showcase.* Haettu 27.12.2020 osoitteesta https://flutter.dev/showcase

Flycricket. (2020). *Disg Golf Putting Practice privacy statement.*Haettu 27.12.2020 osoitteesta https://disg-golf-putting-pr.flycricket.io/privacy.html

GitLab. (n.d.-a). *Documentation. Landing page.* Haettu 24.4.2021 osoitteesta https://docs.gitlab.com/ee/user/project/working_with_projects.html#projects-landing-page

GitLab. (n.d.-b). *Documentation. GitLab CI/CD.* Haettu 24.4.2021 osoitteesta https://docs.gitlab.com/ee/ci

Ionic. (24.3.2021a). *Documentation. Overview.* Haettu 1.4.2021 osoitteesta https://ionicframework.com/docs

Ionic. (24.3.2021b). *Documentation. Web view.* Haettu 1.4.2021 osoitteesta https://ionicframework.com/docs/core-concepts/webview

Ionic. (2021c). *Showcase. Haettu 1.4.2021 osoitteesta* https://ionic.io/customers

PDGA. (2021a). *Introduction.* Haettu 6.4.2021 osoitteesta https://www.pdga.com/introduction

PDGA. (2021b). *Rules. Putting area.* Haettu 6.4.2021 osoitteesta https://www.pdga.com/rules/official-rules-disc-golf/80601

Prettier. (n.d.). *Documentation. Configuration.* Haettu 24.4.2021 osoitteesta https://prettier.io/docs/en/configuration.html

React Native. (2020). *Etusivu.* Haettu 27.12.2020 osoitteesta https://reactnative.dev/

React Native. (12.3.2021a). *Documentation. Testing Rendered Output.* Haettu 1.4.2021 osoitteesta https://reactnative.dev/docs/testing-overview#testing-rendered-output

React Native. (12.3.2021b). *Documentation. Unit tests.* Haettu 1.4.2021 osoitteesta https://reactnative.dev/docs/testing-overview#unit-tests

React Native. (2021c). *Showcase.* Haettu 1.4.2021 osoitteesta https://reactnative.dev/showcase

React Navigation. (n.d.). *Documentation. Authentication flows.* Haettu 1.4.2021 osoitteesta https://reactnavigation.org/docs/auth-flow/

React-native-svg-charts. (n.d.). *Motivation.* Haettu 1.4.2021 osoitteesta https://www.npmjs.com/package/react-native-svg-charts#motivation

Redux. (n.d.-a). *FAQ. Code Structure.* Haettu 24.4.2021 osoitteesta https://redux.js.org/faq/code-structure

Redux. (n.d.-b). *Introduction. Core concepts.* Haettu 25.4.2021 osoitteesta https://redux.js.org/introduction/core-concepts

Redux Toolkit. (n.d.). *Introduction. Getting Started.* Haettu 25.4.2021 osoitteesta https://redux-toolkit.js.org/introduction/getting-started