OULUN AMMATTIKORKEAKOULU

FITNESS-SOVELLUS REACT NATIVELLA

Tommi Jouppila & Janne Lalli
Opinnäytetyö
Kevät 2025
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistokehitys

Tekijät: Janne Lalli, Tommi Jouppila
Opinnäytetyön otsikko: Fitness-sovellus React Nativella
Työn ohjaaja: Jukka Nevalainen
Työn valmistumislukukausi ja -vuosi: kevät 2025
Sivumäärä: 60

Opinnäytetyömme tarkoituksena oli luoda mahdollisimman monipuolinen ja helppokäyttöinen Mobiilisovellus kuntosali treenaamiseen ja ravinnon seurantaan. Sovelluksella voi luoda omia treenipohjia, joita käyttäjä voi suorittaa ja muokata mahdollisimman helposti. Ravinnon seurannassa käyttäjä voi asettaa omat päivittäiset kalori ja proteiini tavoitteet, sekä hakea ruokia tietokannasta tai lisätä omia ruokia. Opinnäytetyön idea oli opiskelijoiden oma.

Opinnäytetyö suoritettiin kehityspainotteisena työnä, jossa käytiin läpi kehityskaaren vaiheet suunnittelusta toteutukseen ja toteutuksesta testailuun. Ajanpuutteesta emme kerenneet luoda viimeistä versiota testailijoiden palautteiden jälkeen. Opinnäytetyö sisältää teoria ja toteutus -osuudet. Teoria osuudessa perustellaan, miksi valitsimme käytetyn ohjelmointikielen, tietokannan, sekä ulkonäön suunnittelua ja siihen käytettyjä työkaluja. Toteutuksessa käydään läpi, kuinka sovellus yhdistettiin tietokantaan ja kuinka sovelluksen eri näkymät toteutettiin. Lopuksi käydään läpi, kuinka sovellusta testailtiin.

Opinnäytetyön lopuksi meille jäi selvät jatkokehityksen suunnitelmat mahdollista julkaisua varten.

ABSTRACT

Oulu University of Applied Sciences
Bachelor of Engineering, Information Technology
Software development

Author(s): Janne Lalli, Tommi Jouppila
Title of thesis: Fitness app with React Native
Supervisor(s): Jukka Nevalainen
Term and year when the thesis was submitted: Spring 2025

Number of pages: 60

The purpose of this thesis was to create a diverse Mobile application for working out and tracking calorie and protein intake. In the application users can create and modify custom workout templates and use those for working out. In the nutrition part of the application users can set their goals for daily calorie and protein intake and search for food from a database as well as create custom foods. The idea for this thesis was students own.

This thesis was produced as a development-oriented project with all the phases of the development cycle in mind from planning to development and from development to testing. Due to the lack of time, we did not have enough time to build a release build based on the feedback we got from testers. This thesis consists of two parts. In the Theory part we explain our reasonings for the programming language and which backend we used, as well as UI planning. In the implementation part we go through how all the different screens of the application were build and how the application is connected to the database. At the end we go through how the application was tested.

At the end of the thesis, we had clear plans for further development and possible release of the application.

SISÄLLYS

SANASTO

Alustariippumattomuus	Ohjelmointikieli, joka ei ole sidoksissa tiettyyn käyttöjärjestelmään
API	Application Programming Interface
APK	Android laitteille paketti, josta sovellus rakentuu
Backend	Ohjelmistokehityksessä sovelluksen taustajärjestelmää.
EAS	Expo Application Services
Firebase	Googlen omistama backend-pilvipalveluiden ja sovelluskehitysalustojen kokonaisuus.
Frontend	Ohjelmistokehityksessä sovelluksen tai verkkosivuston käyttöliittymää.
GitHub	Versionhallintatyökalu
IOS	Apple laitteiden käyttöjärjestelmän nimi
Natiivi	Tietylle käyttöjärjestelmälle luotu ohjelmointikieli
Ohjelmistokehys	Valmiiden työkalujen, kirjastojen ja sääntöjen kokoelma.
RNN	React Native Navigation, navigointikirjasto
Tab-bar	Välilehtipalkki

1 JOHDANTO

Tämän opinnäytetyön tavoitteena on luoda helppokäyttöinen ja monipuolinen mobiilisovellus, joka yhdistää harjoittelun ja ravitsemuksen hallinnan yhteen sovellukseen.

Projektin inspiraationa on halu yhdistää sekä treeni- että ravintopuoli yhdeksi kokonaisuudeksi. Idea syntyi muiden samankaltaisten sovellusten kokeilujen jälkeen, joissa tuli puutteita ilmi. Monet sovellukset keskittyivät vain yhteen osa-alueeseen. Myös treenipohjien luonti- ja muokkaaminen oli useassa kokeillussa sovelluksessa puutteellinen. Nämä havainnot inspiroivat kehittämään sovelluksen, jossa pystyy mahdollisimman käyttäjäystävällisesti tekemään muutoksia.

Tässä opinnäytetyössä perustelemme ohjelmistokehyksen valinnan, sen hyvät ja huonot puolet. Lisäksi selitämme tarkemmin sovelluksen ominaisuuksista, ulkonäöstä ja sen suunnittelusta.

Aloitimme projektin keskustelemalla, mitä ohjelmointikieltä käytämme projektissa, mitä toimintoja sovellus tulee sisältämään, mikä meille on tärkeää sovelluksessa, miltä sovelluksen tulee näyttää ja ketkä sovellusta tulee käyttämään. Teknologian kartoitus oli suhteellisen nopea prosessi, sillä tiesimme heti aikaisemman kokemuksen pohjalta ja yhteisen työskentelyn perusteelta, että haluamme käyttää React Nativea ja backendinä Googlen Firebasea.

Kun perusidea sovellukselle ja sen toiminnoille oli selvillä, siirryimme suunnittelemaan käyttäjäliittymää. Suunnittelu ja käyttäjäliittymän pohjan rakennus tapahtui Figma-työkalulla. Figmaa käyttäen tarkastelemme tarkemmin sovelluksen värimaailmaa ja käyttäjäkokemusta. Kun käyttäjäliittymää rakennetaan, mietimme samalla, millainen sovelluksen käyttäjäpolku tulee olemaan. Käyttäjäpolkua suunnitellessa on hyvä tarkemmin ajatella, ketkä sovellusta tulee käyttämään. Sovelluksen aiheena on kuitenkin Fitness, joten voidaan päätellä, että käyttäjänä ovat ihmiset, joille on tärkeää oma hyvinvointi.

Kun sovellus otetaan käyttöön, ensimmäisenä käyttäjän täytyy kirjautua sisään luomalla käyttäjätili. Käyttäjän luoma tili tallentuu Firebasen

autentikointijärjestelmään. Käyttäjätili luodaan sähköpostia ja salasanaa käyttäen. Kun käyttäjätili on luotu, käyttäjälle luodaan automaattisesti henkilökohtainen käyttäjätunniste tietokantaan. Sovellus hakee kirjautumisen yhteydessä jokaisen käyttäjän henkilökohtaiset tiedot kuten luodut treenipohjat tämän käyttäjätunnisteen avulla. Kirjautumisen jälkeen käyttäjä ohjataan automaattisesti treenisivulle.

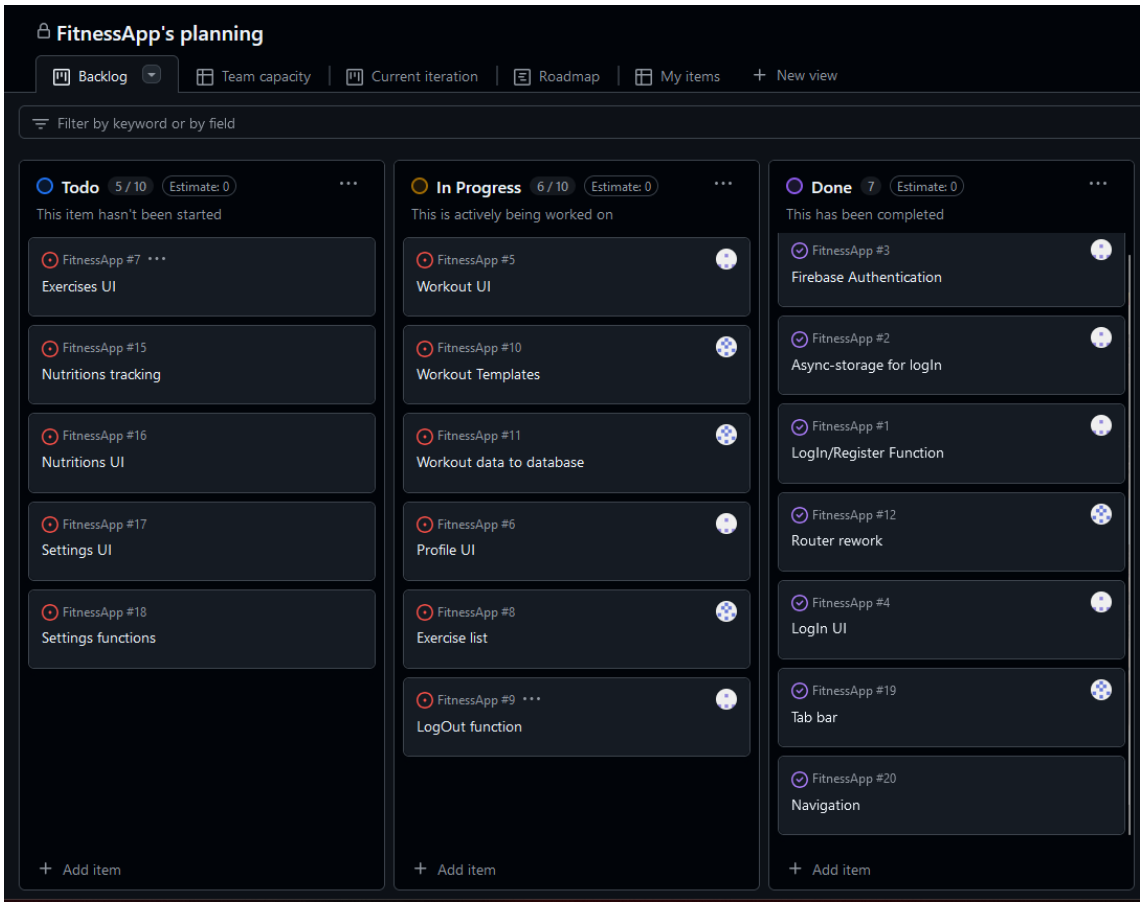
Käyttäjä voi sovelluksen treenisivulla tehdä itse treenipohjan, johon voi lisätä laajasta valikoimasta haluamansa liikkeet ja näille liikkeille toistot ja painot. Kun treenipohja on valmis, voi käyttäjä aloittaa treeninsä. Treenin aloittaessa sovellus aloittaa juoksevan kellon, joka näyttää, kuinka kauan suoritukseen kuluu aikaa. Treenin aikana käyttäjän on myös helppo seurata luodun treenipohjan avulla, mitä liikettä suoritetaan, montako toistoa tehdään ja kuinka paljon painoja käytetään. Suorituksen jälkeen tästä tallentuu käyttäjälle treenin tiedot profiiliinsa. Näin käyttäjän on helppo tarkkailla edistymistään.

Profiili sivulla käyttäjälle näkyy hyödylliset yhteenvedot suoritetuista treeneistä ja ravintokulutuksesta. Käyttäjä voi myös halutessaan kustomoida käyttäjäänsä profiilikuvalla.

Asetuksien kautta käyttäjä voi muuttaa sovelluksen kieltä, yksiköiden muotoa kuten painoyksiköt ja mittayksiköt, myös viikon aloituspäivää voi muuttaa halutessaan. Kaikki tämä lisää sovelluksen käyttäjäystävällisyyttä, joka on hyvin tärkeää tässä sovelluksessa.

Toisena osana sovellusta kehitetään ravitsemuksen seurantatoiminto, jonka avulla käyttäjät voivat kirjata ja seurata päivittäistä energiansaantiaan sekä makroravinteita, kuten proteiinia, rasvaa ja kuitua. Tavoitteena on tarjota käyttäjille selkeä ja visuaalinen tapa tarkastella ravitsemustottumuksiaan ja tehdä tarvittavia muutoksia terveellisemmän elämäntavan saavuttamiseksi. Ravinteiden kulutuksen määrä tallentuu myös tietokantaan. Ruokien ravintoarvot haetaan

Projektihallinnan työkaluina käytössämme oli GitHubin projektisuunnittelutyökalu, jossa seurasimme työn etenemistä ja jaoimme tehtävät (kuva 1). Keskustelu ja itse suunnittelu tapahtui suurimmaksi osaksi Discord-palvelimella.



KUVA 1. GitHub-projektihallintatyökalu (GitHub 2025)

Sovelluksen testaus on tärkeää, jotta käyttäjillä on varmasti toimiva sovellus käytössä heti julkaisusta asti. Tämän sovelluksen testaus suoritettiin henkilökohtaisesti ja ystävien toteuttamana.

2 TEKNOLOGIAN KARTOITUS

Vuonna 2025 tehdyssä tutkimuksessa nähdään, että markkinoilla olevista puhelimista valtaosa käyttää Android- tai iOS-käyttöjärjestelmiä. Laitteista noin 70 prosenttia on Android-laitteita ja 28 prosenttia iOS. (Ahmed 2025.)

Näiden tietojen perusteella pohdimme, että sovelluksen olisi hyvä tukea pääsijaisesti Android-laitteita ja mahdollisesti iOS-laitteita. Rupesimme perehtymään mahdollisiin alustariippumattomiin ohjelmistokehyksiin, jotka tukevat molempia käyttöjärjestelmiä. Pohdinnan jälkeen vaihtoehdoiksi jäi; Flutter tai React Native. Molemmilla vaihtoehdoista on omat hyvät ja huono puolet.

Vuonna 2024 kasatussa tutkimuksessa katsotaan maailmanlaajuisesti alustariippumattomien ohjelmistokehyksien käyttöä (Lionel 2024). Vuonna 2023 Flutter oli 46 prosenttia ja React Native 35 prosenttia käytetyistä ohjelmistokehyksistä.

Harkinnan jälkeen valitsimme ohjelmistokehykseksi React Nativen useista syistä:

Projektin aikataulu on aika tiukka, joten on hyvä hyödyntää mahdollisimman paljon aikaisempaa kokemusta. Projekti tiimillämme oli jo aiempaa kokemusta React Web -ohjelmoinnista. React Native hyödyntää Reactin laajoja kirjastoja. Tämä vähensi oppimiskäyrää huomattavasti ja mahdollisti nopean kehityksen.

React Native sisältää ominaisuuden Hot Reload. Tämän ominaisuuden ideana on mahdollistaa koodin muutosten soveltaminen ilman että koko sovellus pitäisi rakentaa, asentaa ja käynnistää uudelleen muutosten jälkeen. Muutoksia tehdessä Hot Reload pitää nykyisen tilan tallessa, joka nopeuttaa huomattavasti näkymien muokkaamista, kun sinun ei tarvitse navigoida jokaisen muutoksen jälkeen uudelleen haluttuun näkymään. Hot Reload on varsinkin hyödyllinen käyttöliittymää tehdessä, kun visuaaliset muutokset tulevat näkymiin käytännössä saman tien.

React Nativella on suuri ja aktiivinen yhteisö, josta löytyy runsaasti valmiita kirjastoja.

Vaikka Flutter on erittäin lupaava teknologia, se käyttää Dart-ohjelmointikieltä, joka on vähemmän tunnettu ja jolla on pienempi kehittäjäyhteisö. Tämä voisi olla riski projektin pitkän aikavälin ylläpidon kannalta. Näistä syistä React Native osoittautui paremmin tarpeitamme vastaavaksi teknologiaksi.

2.1 React Native

React Native on Facebookin kehittämä avoimen lähdekoodin alusta, joka on suunniteltu mobiilisovellusten kehitykseen sekä Android että iOS-laitteille yhdellä koodipohjalla. React Native toimii natiivielementtien kanssa hyödyntämällä Typescript-kieltä ja React-elementtejä koodin kirjoittamiseen ja kääntämällä tämän koodin natiivikomponenteiksi jokaiselle halutulle alustalle, mikä mahdollistaa lähes natiivin tasoisen suorituskyvyn. Vuoden 2023 alussa React Native siirtyi käyttämään TypeScriptiä oletusohjelmointikielenä version 0.71 yhteydessä.

2.1.1 TypeScript

TypeScript on Microsoftin kehittämä ohjelmointikieli, jonka tarkoituksena on parantaa JavaScriptin heikkouksia. JavaScript käyttää dynaamisia muuttujia, kun taas TypeScript käyttää staattisia. Javascript ohjelmoinnissa muuttujille ei tarvitse erikseen määritellä tyyppiä, vaan se dynaamisesti määrittelee tyypin, kun koodi suoritetaan. Tämä voi yksinkertaistaa ohjelmointia pienissä sovelluksissa, mutta voi johtaa ongelmiin koodin koon kasvaessa. TypeScript vaatii muuttujien määrittelyn jo kirjoitusvaiheessa, jonka avulla ongelma tilanteet tulevat ilmi jo kirjoitus vaiheessa ja on helpompi korjata. Antamalla muuttujille tyypit tekevät myös koodista helpommin luettavaa varsinkin isoissa projekteissa. Koska TypeScript on rakennettu Javascriptin päälle ja on hyvin samankaltainen käyttää, sen käyttöönotto ei tuottanut haasteita.

2.1.2 Expo

Expo on joukko työkaluja, jotka auttavat React-Native kehittäjien työtä. Expo sisältää myös kirjastoja, joiden avulla on esimerkiksi helpompaa ohjelmoida mobiilisovelluksien tunnetuimpia toimintoja, koska niitä ei tarvitse koodata alusta asti. Mutta Expo on myös paljon enemmän kuin pelkkä kokoelma kirjastoja. Se mahdollistaa projektien helpon luonnin pelkän komentorivin avulla, työkalut sovellusten suorittamiseen emulaattorilla ja välittömän muutoksien testaamisen omalla laitteella. Näin ollen Expo tarjoaa erittäin laajat mahdollisuudet React Native –teknologiaa käyttävien sovellusten rakentamiseen. (Ideo Software 2025.)

Expo tarjoaa merkittäviä hyötyjä kehittäjien lisäksi myös yrityksille, jotka tilaavat tällaisia projekteja. Yrityksien näkökulmasta Expon testausmahdollisuus on iso hyöty, sillä Expon avulla voidaan helposti testata erilaisilla laitteilla, mikä parantaa projektin tasokkuutta ja läpinäkyvyyttä. Lisäksi Expon vakaus ja jatkuvat päivitykset varmistavat, että sovellus tulee toimimaan pitkällä aikavälillä. (Ideo Software 2025.)

Vaikka Expo tarjoaa lukuisia etuja voi sen käyttöön liittyä myös rajoituksia, kuten sovelluksen koko suurentuu. Useimmiten sovelluksen koko ei kuitenkaan ole ratkaiseva tekijä, kannattaa kuitenkin pitää mielessä, että Expon käyttö kasvattaa sovelluksen kokoa hieman. Expo ei myöskään sovellu aivan kaikkiin mobiilisovelluksiin. Joskus voi olla tarpeen käyttää “puhdasta” React Nativea tai Flutteria. Ennen projektin aloitusta on järkevää, tutkia tukeeko Expo sovelluksesi erityisvaatimuksia. (Ideo Software 2025.)

Otimme projektiimme Expon käyttöön juuri sen kirjastojen, testaamisen helppouden ja vakauden takia. Expo oli meille tuttu jo entuudestaan, joten sekin vaikutti valintaan, koska tiesimme jo valmiiksi mitä se sisältää ja miten sitä kannattaa käyttää. Tiesimme myös, että se on oikea valinta projektiimme.

2.2 GitHub

GitHub on äärimmäisen suosittu ohjelmointiresurssi, jota käytetään pääsääntöisesti koodin jakamiseen. Se on myös sosiaalinen verkosto ohjelmoijille, yrityksille ja organisaatioille. Koodin jakamisen lisäksi GitHub on hyvä myös projektinhallintaan ja yhteistyön edistämiseen. (Gaba 13.8.2024.)

Koodin tai projektin tallentaminen GitHubiin lisää sen näkyvyyttä ja tavoitettavuutta. Ohjelmoijat voivat löytää lähdekoodeja monilla eri kielillä ja käyttää komentorivityökalua, Gittiä, muutosten tekemiseen ja seuraamiseen. (Gaba 13.8.2024.)

GitHub auttaa tiimin jäseniä työskentelemään yhdessä projektin parissa sijainnista riippumatta ja helpottaa yhteistyötä. Lisäksi se mahdollistaa aiempien versioiden tarkastelun, jotka on luotu projektin aikaisemmissa vaiheissa. (Gaba 13.8.2024.)

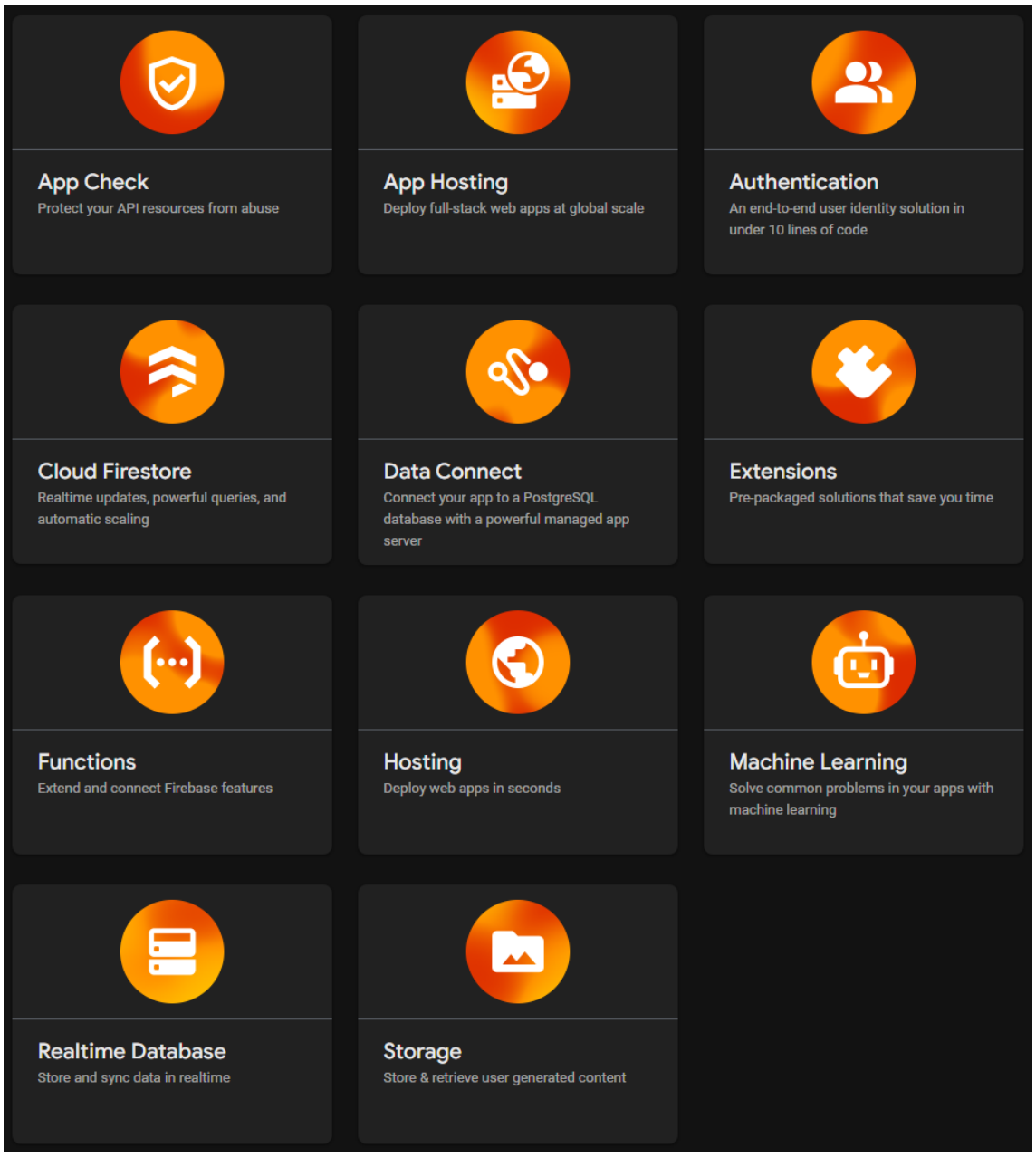
Me valitsimme GitHubin käytön juuri näiden yllä mainittujen syiden takia. Se on yksinkertaisesti paras työkalu koodin jakamiseen ja siinä samalla on äärimmäisen helppo ylläpitää projektinhallintaa sillä, se hoituu samassa paikassa missä koodikin sijaitsee.

2.3 Firebase

Päätimme että käytämme Firebasea sovelluksessa backend -tarpeisiin sekä tietokantana, koska se sisältää helposti säädettävät maksurajat, ja on tiettyyn pisteeseen asti ilmainen. Firebase on hyvin suosittu palvelu mobiililaite sovellusten keskuudessa ja sille löytyy paljon erilaisia ohjeita usealle eri ohjelmointikielelle. Firebase sisältää useita hyödyllisiä ominaisuuksia, joista tälle projektille keskeisimpiä ovat Authentication sekä Firestore Database (kuva 2).

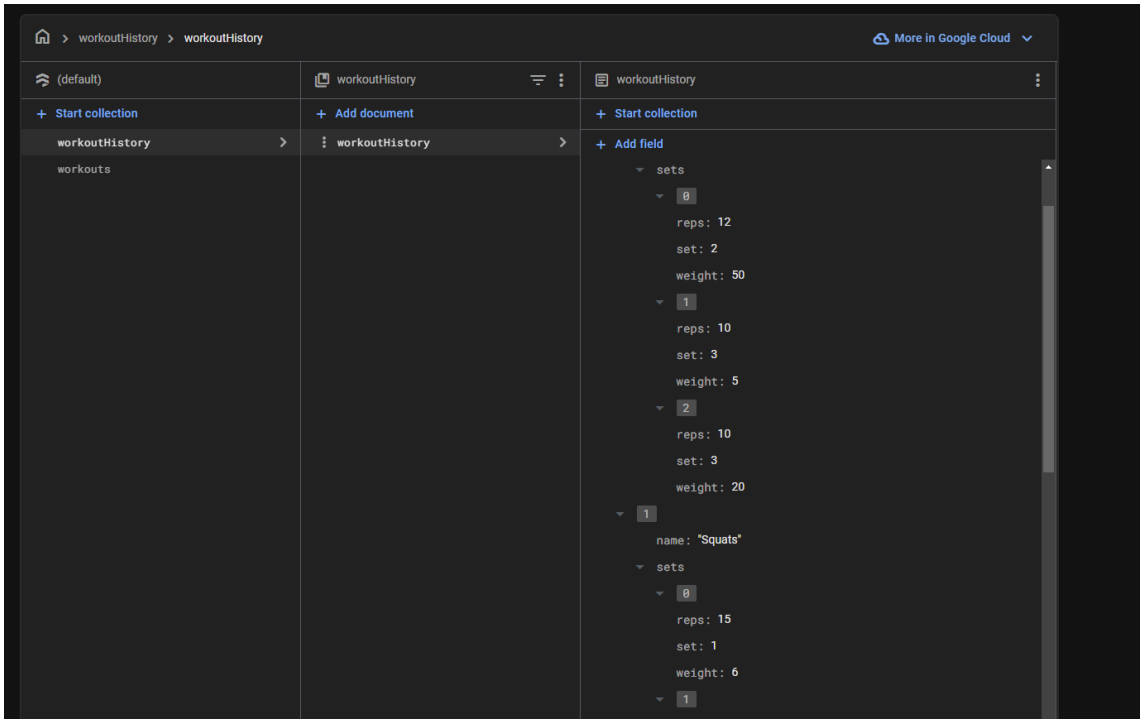
Firebasen Authentication kirjasto tarjoaa tehokkaat työkalut käyttäjien hallintaan ja kirjautumistoimintoihin. Authentication tukee useita kirjautumis- vaihtoehtoja

kuten sähköposti ja salasana, puhelinnumero ja kolmannen osapuolen tarjoajat kuten Google, Facebook, Twitter ja muita. (Firebase 2025.) Kehittäjät voivat itse valita mitkä kirjautumisvaihtoehdot sovellus tarjoaa. Meidän tapauksessamme kirjautuminen tapahtuu pääasiassa perinteisellä sähköposti ja salasana yhdistelmällä. Firebase Authentication säästää huomattavasti aikaa, koska se käsittelee automaattisesti kirjautumistilan hallinnan, tokenien päivityksen ja käyttäjän istuntojen ylläpidon. Tämä tarkoittaa sitä, että meidän ei tarvitse luoda



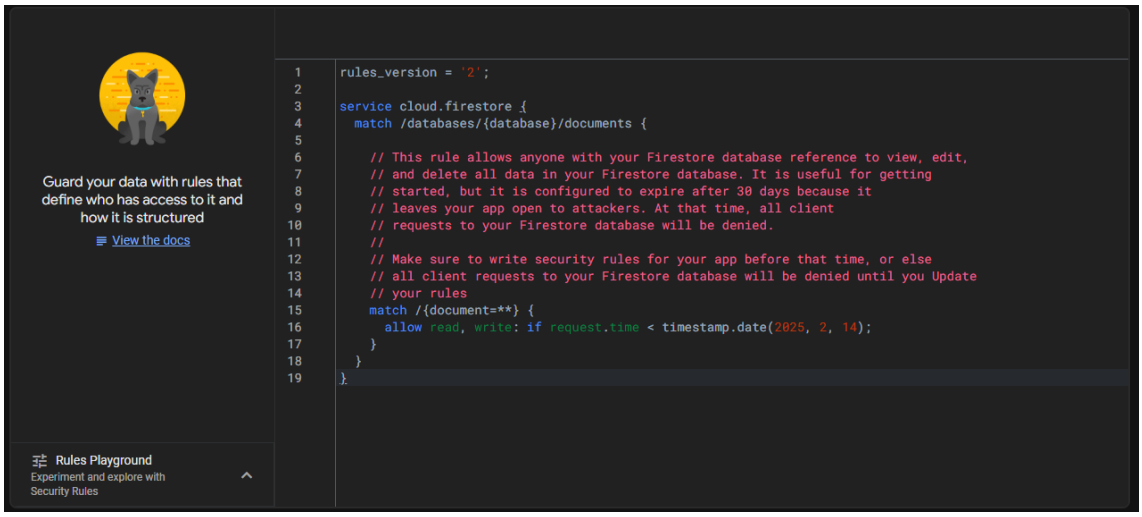
KUVA 2. Firebase palvelun tarjomia ominaisuuksia (Firebase 2025)

Firebase tarjoaa kahta eri pilvi pohjaista tietokantaa, joista valitsimme Firestore-tietokannan Realtime Databasen sijaan. Firestorella on useita ominaisuuksia, jotka tekevät siitä paremman vaihtoehdon projektillemme. Firestoressa Data tallentuu “kokoelmiin” jotka koostuvat dokumenteista. (Kuva 3.) Tämä tekee datan tallentamisesta, käsittelystä ja lukemisesta huomattavasti helpompaa kuin Realtime Database, joka tallentaa kaiken yhteen JSON puuhun, joka tekisi tiedon tallentamisesta ja hausta huomattavasti hankalampaa.



KUVA 3. Tietokannan rakenne projektinalkuvaiheessa (Firebase 2025)

Projektin alussa tietokannan muokkaussäännöt olivat hyvin avoimet. Säännöt sallivat kaikkien käyttäjien muokata kaikkia dokumentteja siitä huolimatta, oliko dokumentti käyttäjän luoma vai ei (kuva 4). Tämä ei projektin alkuvaiheessa haitannut, mutta turvallisuussyistä oli kuitenkin hyvä muokata sääntöjä. Päätimme tehdä hyvin yksinkertaiset säännöt, jotka sallivat vain autentikoitujen käyttäjien lukea ja muokata dokumentteja (kuva 5).



KUVA 4. Firebasen alkuperäiset säännöt (kuvakaappaus Firebasen konsolista)

```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {

    // Allow authenticated users to read and write any document in any collection
    match /{document=**} {
      allow read, write: if request.auth != null;
    }
  }
}
```

KUVA 5. Uudet tietokannan säännöt, jotka sallivat vain autentikoidun käyttäjän lukea ja muokata tiedostoja. (kuvakaappaus Firebasen konsolista)

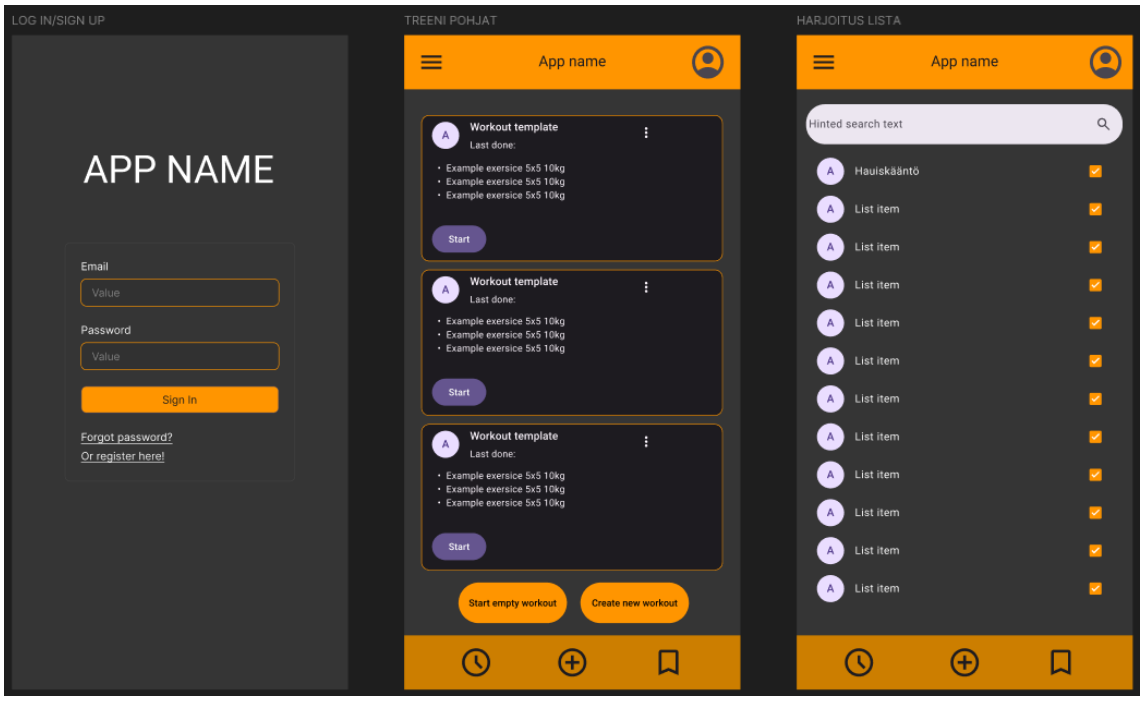
3 KÄYTTÖLIITTYMÄN SUUNNITTELU

3.1 Figma

Käyttöliittymän suunnittelussa käytettiin työkalua nimeltään Figma. Suunnittelu työkalua on hyvä käyttää ennen itse ohjelmoinnin aloitusta. Näin saadaan ennen ohjelmointia kuvan siitä minkälainen sovelluksesta olisi tarkoitus tulla. (Figma Learn 2019.)

Figmassa on laajat valmiit pohjat ja työkalut, jolla voi tehdä juuri sellaisen käyttöliittymän kuin haluaa. Figma voi olla ensimmäisellä kerralla hieman sekavan oloinen ja toiminnot vaikeasti löydettävissä. Työkalulle kuitenkin löytyy todella paljon ohjeita ja esimerkkejä. (Figma Learn 2019.)

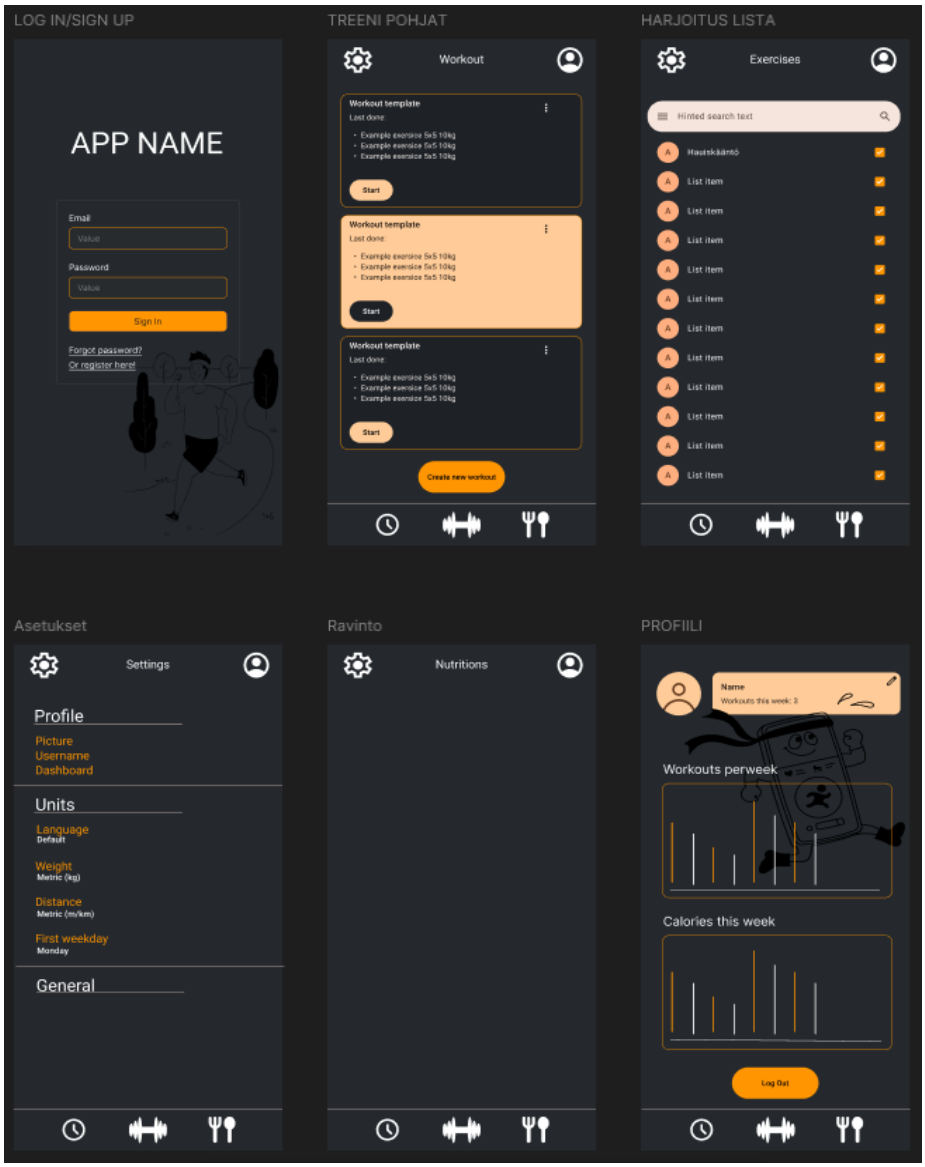
Aloitimme käyttöliittymän suunnittelun luomalla pohjat kirjautumis-, treeni- ja harjoitusliikesivuille (kuva 6).



KUVA 6. Figmalla luodut käyttöliittymän pohjat (Figma 2025)

Kun käyttöliittymän suunnittelu Figmassa alkaa näyttämään hyvältä, on aika siirtyä ohjelmoinnin pariin ja toteuttaa suunnitelma. Tässä vaiheessa, kun kolme sivua on suunniteltu lähes valmiiksi, aloitimme ohjelmoinnin. Kun ohjelmointi lähti

etenemään, niin voi samalla jo miettiä, miltä muut sivut kuten asetukset voisivat näyttää (kuva 7). Näinpä käyttöliittymän suunnittelu jatkuu myös ohjelmoinnin ohella eikä välttämättä lopu missään vaiheessa projektin edetessä.



KUVA 7. Käyttöliittymän suunnittelu jatkuu ohjelmoinnin ohella (Figma 2025)

3.2 Värimaailma

Sovelluksen värimaailma on toiseksi tärkein osa-alue sovelluksen toiminallisuuden jälkeen. Värit auttavat käyttäjiä näkemään ja tulkitsemaan sovelluksen sisältöä, käyttämään oikeita elementtejä ja ymmärtämään toimintoja. Jokaisella

sovelluksella on aina oma värimaailmansa, ja se käyttää päävärejä tärkeimmillä alueillaan. (Babich 25.1.2017.)

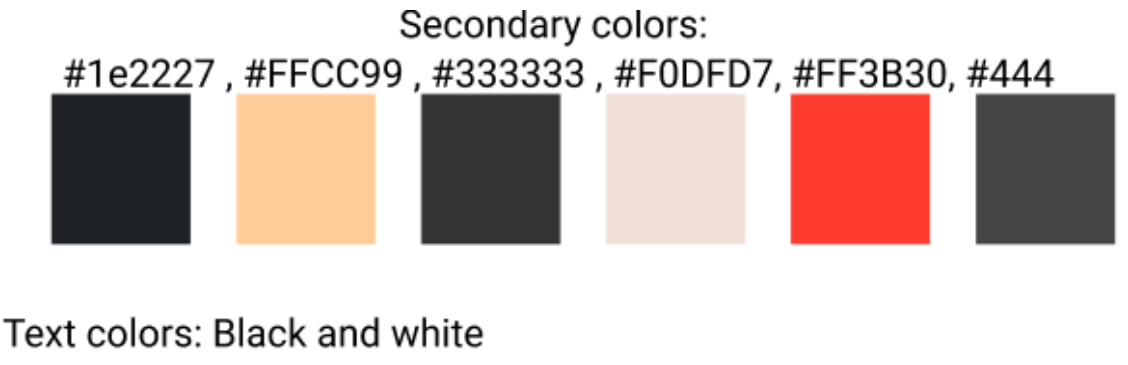
Sovelluksissa on yleensä ainakin kaksi päväriä ja useampi sivuväri. Päävärit valitaan ensin ja niitä käytetään esimerkiksi sovelluksen taustaan ja isompiin yksityiskohtiin. Tämän jälkeen on helpompi kartoittaa, mitkä värit sopivat näiden kanssa pienempiin yksityiskohtiin, joihin sivuvärejä yleensä käytetään. (Babich 25.1.2017.)

On tärkeää varmistaa, että käyttöliittymäelementtien välillä on riittävästi kontrastia. Tämän voi tarkistaa kontrastisuhteiden avulla. Kontrastisuhteet ilmaisevat, kuinka paljon värit eroavat toisistaan. Yleensä kontrastisuhteet merkitään muodossa 1:1 tai 21:1. Mitä suurempi ero suhdelukujen kahden numeron välillä on, sitä suurempi värien välinen suhteellinen kirkkausero on. Suositus kontrastisuhde leipäteksteille ja kuviin sijoitetuille teksteille on yleensä pienen tekstin taustaan nähden 4,5:1 ja suuren tekstin taustaan nähden 3:1. Tämä ohjeistus auttaa myös heikkonäköisiä käyttäjiä, värisokeita sekä niitä, joiden näkökyky heikkenee, näkemään ja lukemaan tekstin näytöltäsi. (Babich 25.1.2017.)

Tässä projektissa päävärit ovat tummanharmaa ja oranssi (kuva 8).

KUVA 8. Kuvakaappaus sovelluksen päävärеistä

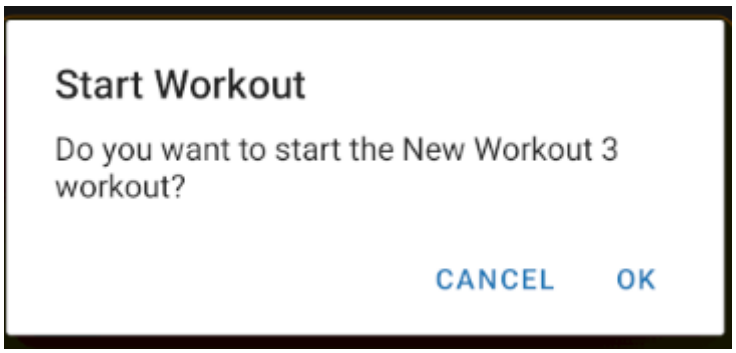
Sivuväreinä käytössä on eri sävyisiä harmaita, oransseja ja hieman punaista ja beigeä. Tekstin väreinä käytetään pääosin mustaa ja valkoista. (Kuva 9.)



KUVA 9. Kuvakaappaus sovelluksen sivu- ja tekstiväreistä

3.3 Varoitusikkuna

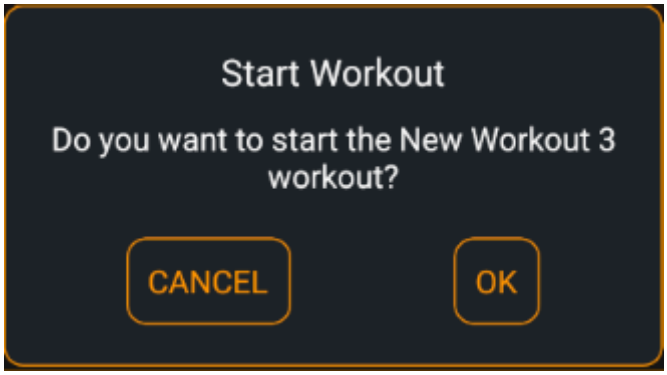
Sovelluksissa käytetään varoitusikkunoita, kun käyttäjä tekee jotain peruuttamatonta. Näitä asioita voivat olla esimerkiksi, jos käyttäjä tekee muutoksia treenipohjaan ja on lähtemässä tallentamatta sivulta tai jos käyttäjä kirjautuu ulos voi tulla varoitusikkuna myös. Varoitusikkunat ovat suunniteltu käyttäjää varten, ettei vahingossakaan tapahdu jotain peruuttamatonta. Sovelluksissa alustavasti käytetään käyttöjärjestelmän mukaista varoitusikkunaa ja tämä ikkuna voi poiketa hyvin paljon sovelluksen teemasta (kuva 10). Siksi on järkevää tehdä sovellukselle täysin oma varoitusikkuna, jota hyödynnetään vain tämän sovelluksen sisällä.



KUVA 10. Alkuperäinen varoitusikkuna (kuvakaappaus sovelluksesta)

Uusi, sovelluksen oma varoitusikkuna luotiin käyttämällä React Nativen omaa kirjastoa ja sen sisältämää Modal-toimintoa (kuva 11). Varoitusikkuna tehtiin uuteen komponenttiin, jossa ikkunalle määriteltiin sisältö ja ulkonäkö (kuva 12). Se, että

varoitusikkuna on luotu erikseen, tarkoittaa, että voimme kutsua tätä samaa varoitusikkunaa sovelluksen muille sivuille yksinkertaisesti.



KUVA 11. Sovelluksen oma varoitusikkuna (kuvakaappaus sovelluksesta)

```tsx
interface CustomAlertProps {
  visible: boolean;
  title?: string;
  message?: string;
  cancelText?: string;
  confirmText?: string;
  onCancel?: () => void;
  onConfirm: () => void;
}

const CustomAlert: React.FC<CustomAlertProps> = ({
  visible,
  title,
  message,
  cancelText = 'CANCEL',
  confirmText = 'OK',
  onCancel,
  onConfirm,
}) => {
  return (
    <Modal visible={visible} transparent animationType="fade">
      <View style={styles.overlay}>
        <View style={styles.modalContent}>
          {title && <Text style={styles.title}>{title}</Text>}
          {message && <Text style={styles.message}>{message}</Text>}
          <View style={styles.buttonContainer}>
            {/* Only render the cancel button if onCancel is provided */}
            {onCancel && (
              <TouchableOpacity style={styles.cancelButton} onPress={onCancel}>
                <Text style={styles.cancelButtonText}>{cancelText}</Text>
              </TouchableOpacity>
            )}
            <TouchableOpacity
              style={onCancel ? styles.confirmButton : styles.singleButton}
              onPress={onConfirm}
            >
              <Text style={styles.confirmButtonText}>{confirmText}</Text>
            </TouchableOpacity>
```

KUVA 12. Kustomoidun varoitusikkunan luonti (kuvakaappaus koodista)

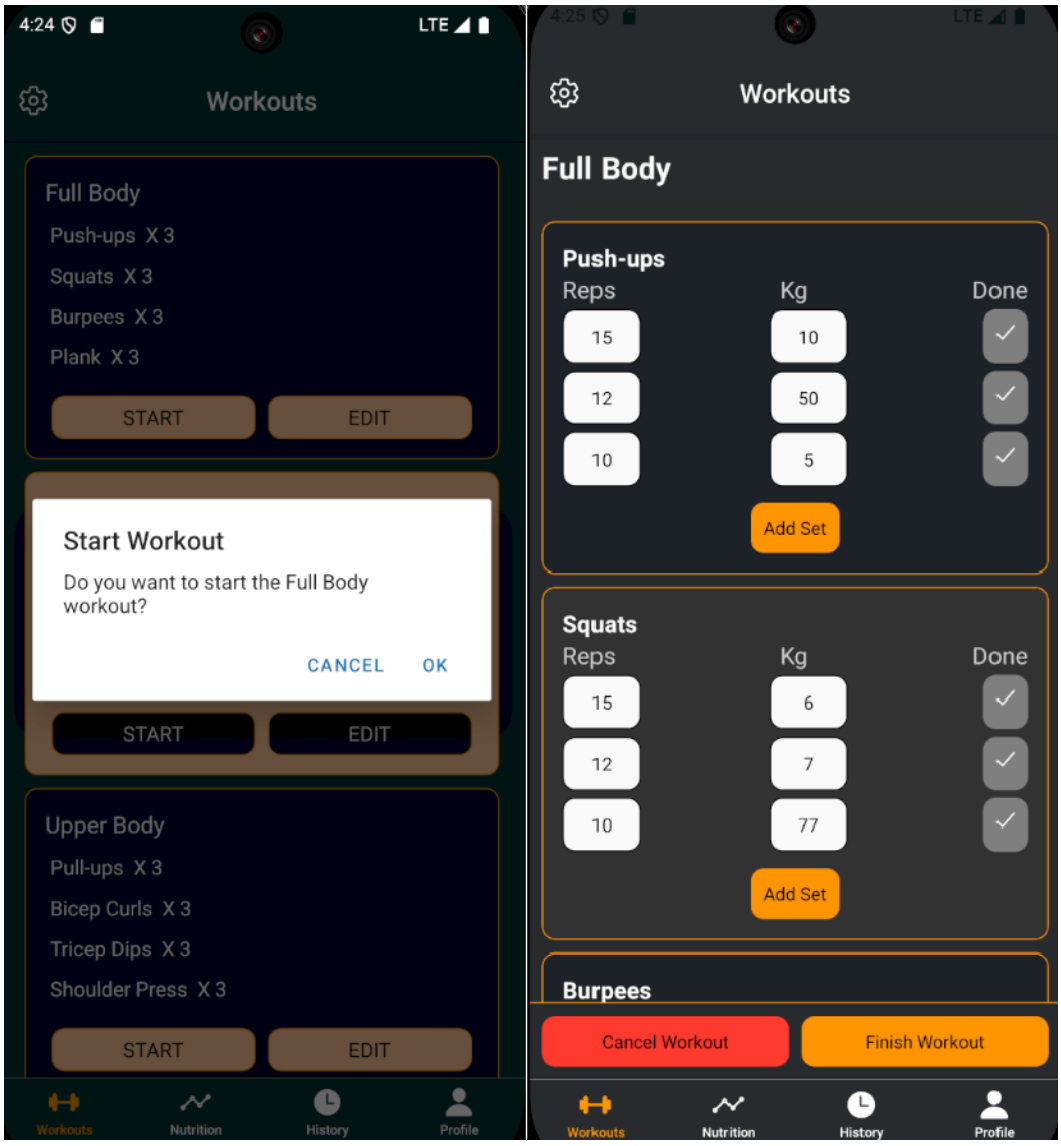
4 TOTEUTUS

4.1 Aloitus

Aloitimme projektin ottamalla Expon projektin pohjan käyttöön komentorivi komennolla npx create-expo-app@latest FitnessApp, joka luo uusimman saatavilla olevan version Expon luomista pohjista. Päätimme kuitenkin vähentää Expo-kirjastojen riippuvuutta, ja vaihdoimme Expon navigoinnin tilalle React Native Navigation -kirjaston (RNN). RNN käyttää natiivikomponentteja navigointiin, mikä tekee siitä nopeamman ja sulavamman verrattuna Expo Routeriin. RNN antaa myös enemmän hallintaa silloin, kun navigointi vaatii enemmän logiikkaa, esimerkiksi sisäisessä navigoinnissa näkymien välillä, kuten sovelluksen navigoinnissa Workouts -näkymästä ActiveWorkout-näkymään (kuva 13). Kun käyttäjä valitsee haluamansa treenin, sovellus navigoi ActiveWorkout-näkymään ja tuo mukaansa valitun treenin tiedot (kuva 14). Tiedon siirtyminen näkymästä toiseen onnistui helpommin RNN:n kanssa.

```
type RootStackParamList = {
  index: undefined;
  Profile: undefined;
  Workouts: undefined;
  Nutrition: undefined;
  History: undefined;
  ActiveWorkout: { workoutName: string; moves:
                   { name: string; sets:
                   { set: number; reps: number; weight: number; }[] }[] };
  Settings: undefined;
};

// Create Bottom Tabs Navigator
const Tab = createBottomTabNavigator();
const Stack = createStackNavigator<RootStackParamList>();

const WorkoutsStack = () => {
  return (
    <Stack.Navigator>
      <Stack.Screen
        name="Workouts"
        component={Workouts}
        options={{ headerShown: false }} // Hide the header for the workouts screen
      />
      <Stack.Screen
        name="ActiveWorkout"
        component={ActiveWorkout}
        options={{ headerShown: false }}
      />
    </Stack.Navigator>
  );
};
```

KUVA 13. Navigointilogiikka React Native Navigationilla (kuvakaappaus koodista)

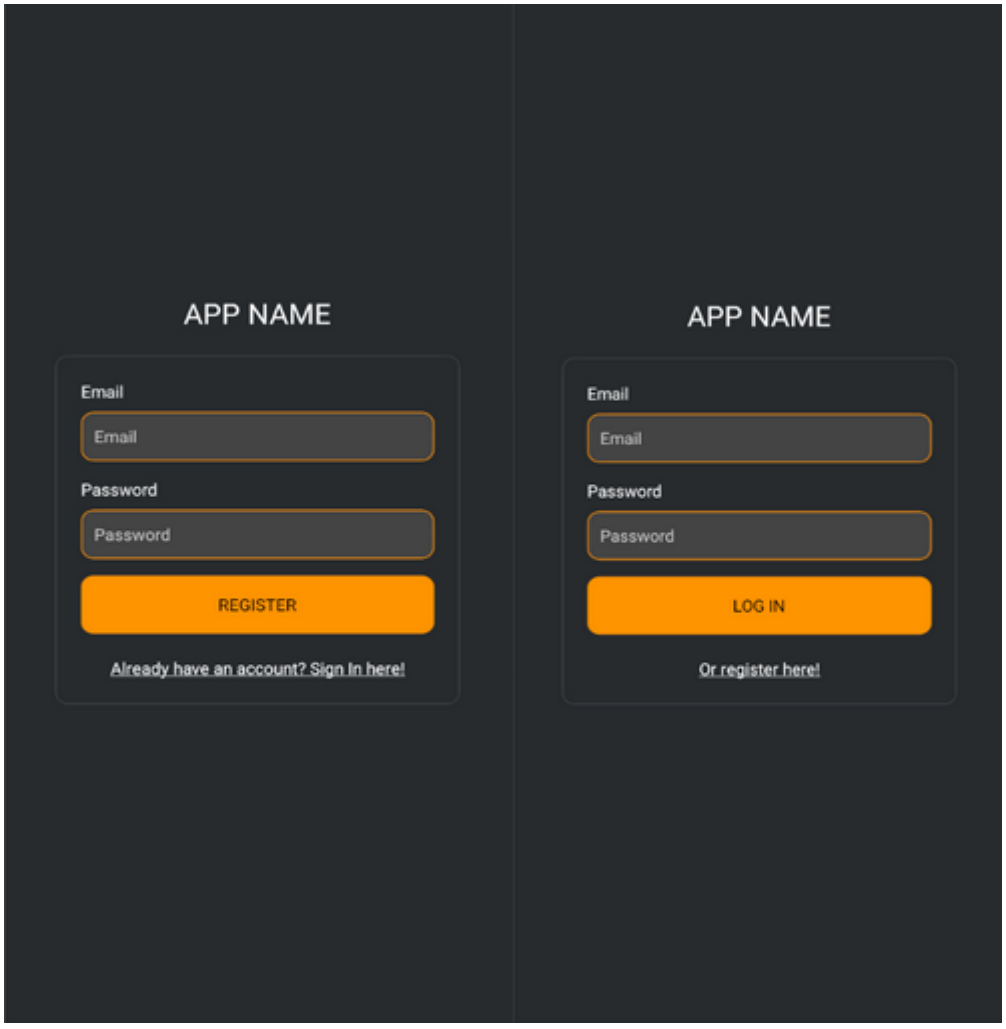


KUVA 14. Treenitiedon siirtyminen aktiivisen treenin näkymään (kuvakaappaus sovelluksesta kehitysvaiheessa)

4.2 Kirjautuminen ja rekisteröinti

Navigaatiologiikan yhteydessä valmistui myös kirjautumis- ja rekisteröintisivu. Kirjautumisnäkymän ohjelmointi aloitettiin Figmassa tehdyn käyttöliittymän pohjalta. Näkymässä täytyy olla tekstikentät käyttäjätietoja varten. Tässä sovelluksessa kirjautuminen ja rekisteröinti onnistuu sähköpostia ja salasanaa käyttämällä. Ennen kirjautumista käyttäjän täytyy kuitenkin luoda käyttäjä eli rekisteröityä. Kirjautumis- ja rekisteröintitoimintojen välinen vaihto tapahtuu

yksinkertaisesti painamalla joko Or register here! -painike kirjautumisen yhteydessä tai Already have an account? Sign in here! -painiketta rekisteröinnin yhteydessä (kuva 15). Näkymä vaatii myös kirjautumispainikkeen. Tätä nappia painamalla käyttäjä rekisteröi käyttäjänsä ja kirjautuu sisään sovellukseen, jos tiedot ovat syötetty oikein. Jos tiedot eivät ole kuitenkaan oikein tai käyttäjä ei ole rekisteröitynyt, kirjautuminen ei onnistu. (Kuva 16.)



KUVA 15. Rekisteröinti- ja kirjautumisnäkymä (kuvakaappaus sovelluksesta)

```
// Käsittelee kirjautumisen tai rekisteröinnin
const handleAuthAction = async () => {
  if (!validateInputs()) return;

  setLoading(true);
  try {
    if (isRegister) {
      // Rekisteröinti
      const userCredential = await createUserWithEmailAndPassword(auth, email, password);
      saveUID(userCredential.user.uid);
      Alert.alert('Success', 'User registered successfully!');
      setIsRegister(false); // Switch back to login after successful registration
      //console.log('User registered:', userCredential.user);
    } else {
      // Kirjautuminen
      const userCredential = await signInWithEmailAndPassword(auth, email, password);
      saveUID(userCredential.user.uid);
      Alert.alert('Success', 'User signed in successfully!');
      };

  } catch (error: any) {
    if (error.code === 'auth/user-not-found') {
      Alert.alert('Error', 'User not found. Please check your email.');
    } else if (error.code === 'auth/wrong-password') {
      Alert.alert('Error', 'Incorrect password. Please try again.');
    } else if (error.code === 'auth/email-already-in-use') {
      Alert.alert('Error', 'This email is already in use.');
    } else {
      Alert.alert('Error', error.message || 'An unknown error occurred.');
    }
  } finally {
    setLoading(false);
  }
};
```

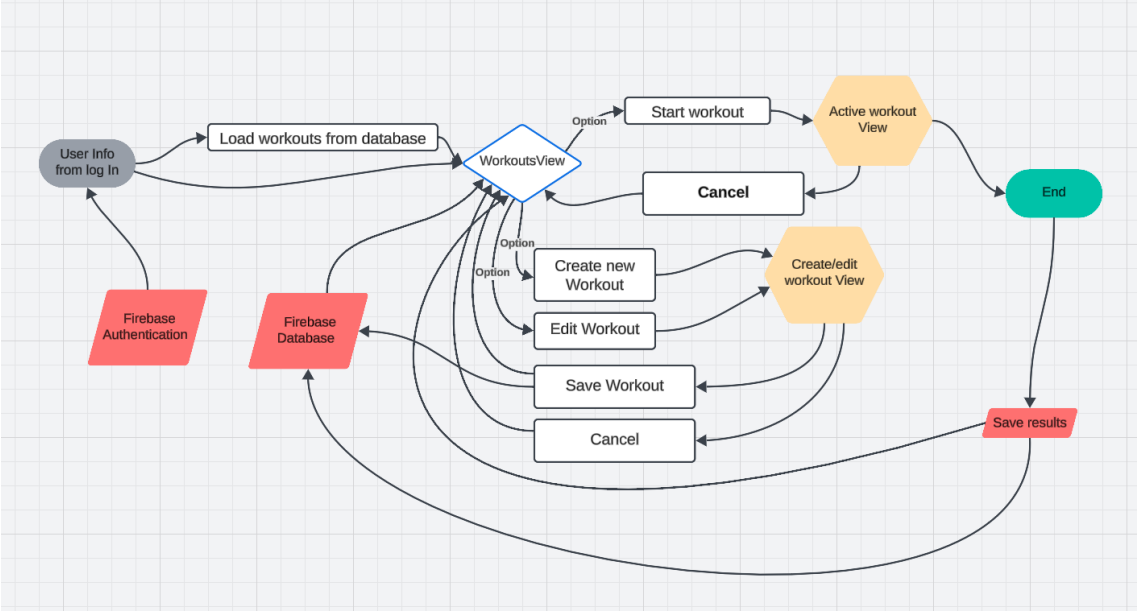
KUVA 16. Kirjautuminen ja rekisteröinti koodissa (kuvakaappaus koodista)

Sisäänkirjautumisen yhteydessä sovellus ottaa käyttäjän User ID:n Firebasesta ja tämän avulla näyttää käyttäjän omat suoritukset ja treenipohjat. Onnistuneen kirjautumisen jälkeen käyttäjä ohjataan treeninäkymään.

Kun navigaatio- ja kirjautumislogiikka oli valmis, aloitimme tekemään sekä Workouts -näkymää että profiili sivun ulkonäköä väliaikaisen testi datan kanssa.

4.3 Workouts

Workouts -näkymä on sovelluksen treeni osuuden ydin. Se sisältää runsaasti logiikkaa sekä navigointiin, että tiedonhakuun paikallisesti, sekä tietokannasta (kuva 17). Se sisältää ominaisuudet aloittaa treenin suoritus, luoda treenipohjia, sekä muokata treenejä. Kun uusi käyttäjä kirjautuu sisään, sovellus tarjoaa valmiiksi luotuja treenipohjia, joita käyttäjä voi suorittaa tai muokata. Käyttäjä voi myös luoda omia treenejä tyhjältä pohjalta. Näkymässä haetaan sekä paikallisesta tiedostosta sovelluksen treenipohjia, että Firestoresta sisään kirjautuneen käyttäjän luomat treenipohjat (kuva 18). Molemmat datat ovat rakenteeltaan identtistä JSON dataa, joka helpotti koodin kirjoittamisessa. Identtisten data rakenteiden ansiosta molemmat datat voivat käyttää samaa Workoutitem -funktiota valittavien treenien visualisointiin.



KUVA 17. Workouts näkymän logiikka (kuvankaappaus Lucidchart ohjelmasta)

```
useEffect(() => {
  const fetchUserWorkouts = async () => {
    try {
      const userid =  getAuth().currentUser?.uid;
      if (!userid) {
        throw new Error('User is not authenticated or UID is missing.');
      }
      const q = query(collection(db, 'userWorkouts'), where('userid', '==', userid));
      const querySnapshot = await getDocs(q);

      const workouts: Workout[] = querySnapshot.docs.map((doc) => ({
        id: doc.id, // Store document ID
        ...doc.data(),
      })) as Workout[];

      setUserWorkouts(workouts);
    } catch (error) {
      console.error('Error fetching user workouts:', error);
      alert('Failed to fetch user workouts.');
    }
  };

  fetchUserWorkouts();
}, []);
```

KUVA 18. Käyttäjän luotujen treenien haku tietokannasta. (kuvakaappaus koodista)

Kun käyttäjä päättää luoda uuden treenin joko tyhjältä pohjalta, tai muokata omaa tai sovelluksen tarjoamaa pohjaa, sovellus navigoi WorkoutEditor -näkymään tarvittavien tietojen kanssa. Jos Näkymään siirrytään tyhjän pohjan kautta, navigaatioon ei tarvitse tuoda mukaan dataa. Jos käyttäjä päättää muokata sovelluksen omia treenipohjia, Navigaatiossa otetaan mukaan Treenin nimi, liikkeet, toistot ja käytetyt painot liikkeille. Jos käyttäjä päättää muokata itse tehtyjä treenejä, tämän lisäksi mukaan otetaan kyseisen treeni dokumentin Nimi, jotta tehdyt muutokset saadaan tallennettua valittuun treeniin, eikä luoda uuttaa treeniä (kuva 19). Tällä saadaan vähennettyä turhien kopioiden luontia tietokantaan.

KUVA 19. Treenien muokkaus näkymään siirtyminen käyttäjän luomasta treenistä (kuvakaappaus koodista)

4.4 WorkoutEditor

WorkoutEditor näkymässä sovellus tuo esille treenin minkä käyttäjä valitsi. Käyttäjä voi muokata treenin nimeä, liikkeiden nimiä, poistaa tai lisätä toistoja sekä sarjoja. Jos näkymään siirrytään “Create new workout” nappulan kautta, Workouteditor luo tyhjän treeniin nimella “New Workout”. Kun käyttäjä lisää uuden liikkeen, sovellus luo tyhjän liike pohjan, joka sisältää yhden sarjan, jossa toistot ja painot ovat arvoltaan 0. Käyttäjä voi lisätä liikkeisiin “Add set” nappulasta niin monta toistoa, kun haluaa (kuva 20). Kun toistojen arvoja muutetaan, niiden arvot pitää myös siirtää tallennettuun dataan, joka siirtyy tallentaessa tietokantaan. Tämä tapahtuu Reactin UseState Funktion avulla. (React). Kun toiston sisäisessä kentässä muutetaan toistojen tai painojen arvoa, sovellus kutsuu handleupdateSet, jonka kautta toiston molemmat arvot päivittyvät myös dataan (kuva 21).

Kun käyttäjä on valmis ja haluaa tallentaa treenin, näkymän alaosasta löytyy Save Workout painike, joka kutsuu saman nimistä funktiota. Tallennus funktiossa koodi tarkistaa, onko tallennettava treeni uusi tai aiemmin luotu treeni, jota muokataan. Jos Treeni on aiemmin luotu treeni, jota muokataan, navigoinnin mukana otettiin talteen dokumentin nimi, jota käytetään tallentamiseen, jolloin dokumentti päivittyy, eikä luo turhaan kopiota samasta treenistä. Jos kyseessä on uusi treeni,

sen nimeksi luodaan uuid kirjastosta v4 (npmJS) funktiota, joka luo uniikin nimen jokaiselle dokumentille (kuva 22).

KUVA 20. Useita toistoja lisättynä liikkeeseen (kuvakaappaus ohjelmasta)

```
{move.sets.map((set, setIndex) => (
    <View key={setIndex} style={styles.textInputContainer}>
        <Text style={styles.textstyle}>Reps:</Text>
        <TextInput
            style={styles.textInput}
            value={set.reps.toString()}
            keyboardType="numeric"
            onChangeText={(value) => handleUpdateSet(moveIndex, setIndex, 'reps', value)}
        />
        <Text style={styles.textstyle}>Weight:</Text>
        <TextInput
            style={styles.textInput}
            value={set.weight.toString()}
            keyboardType="numeric"
            onChangeText={(value) => handleUpdateSet(moveIndex, setIndex, 'weight', value)}
        />
```

KUVA 21. Toistojen koodi, joka kutsuu päivitys funktiota arvojen muuttuessa (kuvakaappaus koodista)

```
const docId = workoutId || `${v4()}-${user.uid}`;
```

KUVA 22. Tarkistetaan onko kyseessä aiempi treeni tai luodaan uusi dokumentti (kuvakaappaus koodista)

Tallennuksen yhteydessä sovellus navigoi takaisin workouts näkymään. Kun workouts näkymään siirrytään, sovellus tunnistaa navigaation ”addListener” funktiota käyttäen, että näkymä on saanut kohdistuksen, ja hakee tietokannasta käyttäjän treenit, joita voidaan käyttää treenin suorituksessa (kuva 23).

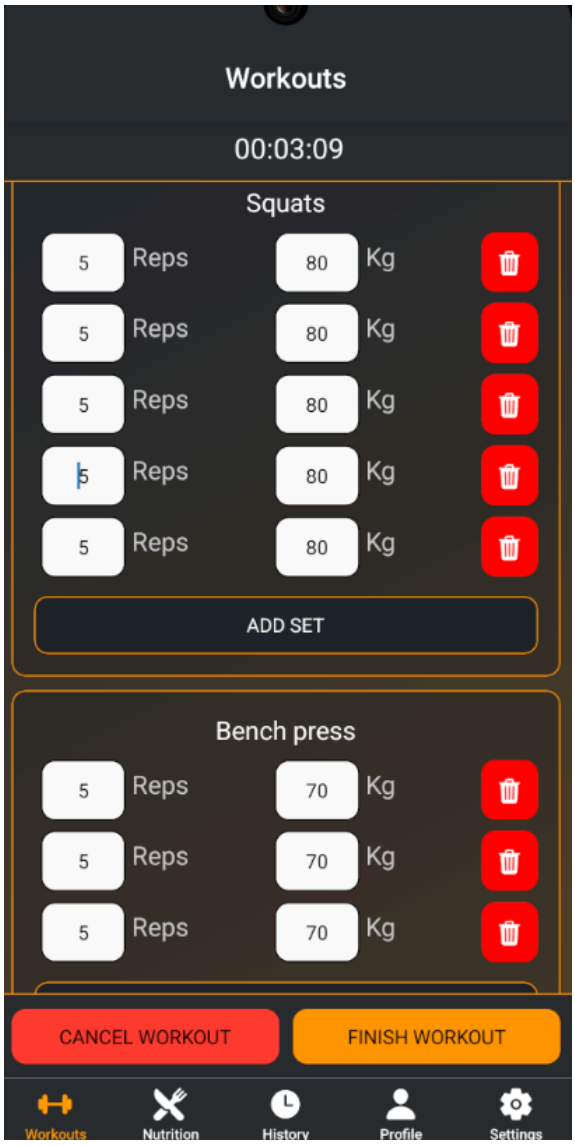
```
  const unsubscribe = navigation.addListener('focus', fetchUserWorkouts)
  return unsubscribe;
}, [navigation]);
```

KUVA 23. Näkymän tunnistaa saaneensa kohdistuksen (kuvakaappaus koodista)

4.5 ActiveWorkouts

Kun käyttäjä valitsee treenin mitä haluaa suorittaa, sovellus navigoi Activeworkouts näkymään tarvittavien tietojen kanssa. Kun näkymään siirrytään, Valittu treeni visualisoidaan samankaltaisesti kuin Workouteditor näkymään siirtyessä.

Tämän lisäksi näkymän Yläosassa näkyy ajastin, joka kertoo, kuinka pitkään käynnissä oleva treeni on kestänyt (kuva 24). Ensimmäinen versio ajastimesta toteutettiin logiikalla, joka sekunnin välein muutti muuttujan arvoa yhdellä. Tämä toteutus oli toiminnallinen, kun sovellus oli auki, mutta ongelmia ilmeni, kun sovellus oli taustalla tai puhelin oli lukittu, jolloin kello pysähtyi. Käytännöllisessä tilanteessa on todennäköistä, että käyttäjän puhelin on treenin aikana lukittuna tai taustalla, joten ajastimelle täytyi keksiä uusi keino. Ratkaisuna käytimme Date().getTime funktiota. Date funktio antaa objektin, joka perustuu aikaleimaan, joka alkaa vuoden 1970 ensimmäisestä päivästä. getTime() funktio palauttaa saman ajankohdan mutta millisekuntien muodossa (mdn web docs.). Tätä käyttäen, sovellus ottaa näkymään siirtyessä ensimmäisen aikaleiman talteen, ja vertaa tätä muuttujaa toiseen aikaleimaan, joka päivittyy sekunnin välein. Täten aika muuttuja saadaan päivitettyä, vaikka sovellus olisi pysäytettynä. Aikaleima täytyy vielä muuttaa tunti, minuutti ja sekunti muotoon sekä visualisointia että tallennusta varten (kuva 25). Sekunnit muutetaan tunneiksi jakamalla 3600:lla ja tämä vielä 60 niin saadaan minuutit, jonka lisäksi eri laskuvaiheessa käytetään JavaScriptin Math.floor() funktiota, jolla saadan varmistettua, että arvot ovat tasalukuja. Lisäksi käytettii .padStart(2,0) muuttujille jolla varmistettiin että aika on aina hh:mm:ss muotoa.



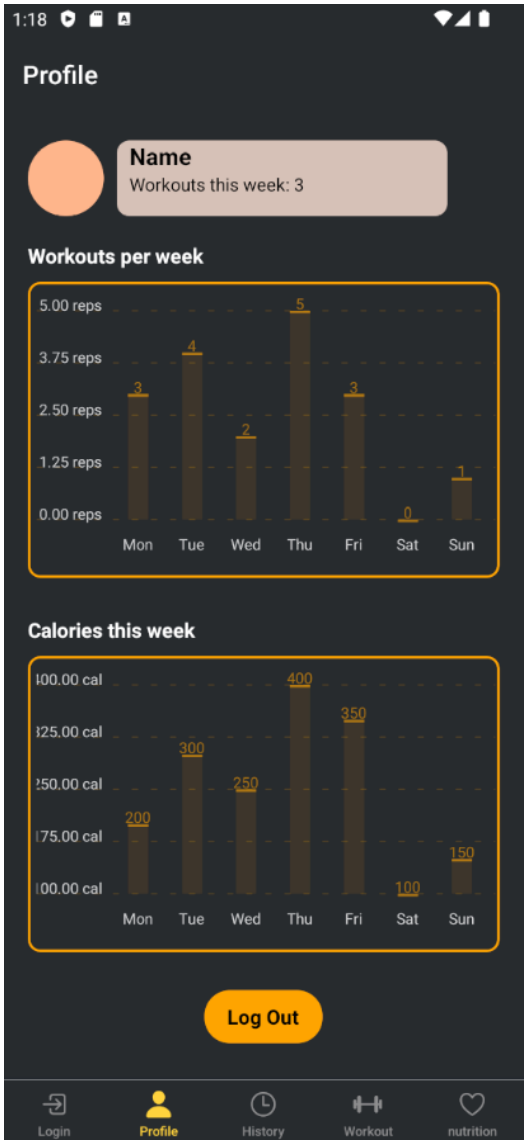
KUVA 24. Activeworkout näkymä (kuvakaappaus ohjelmasta)

```
const [elapsedTime, setElapsedTime] = useState(0);

const startTime = new Date().getTime();
useEffect(() => {
  const interval = setInterval(() => {
    const currentTime = new Date().getTime();
    setElapsedTime(Math.floor((currentTime - startTime) / 1000));
  }, 1000);

  return () => clearInterval(interval);
}, []);



const formatTime = (seconds: number) => {
  const h = Math.floor(seconds / 3600)
    .toString()
    .padStart(2, '0');
  const m = Math.floor((seconds % 3600) / 60)
    .toString()
    .padStart(2, '0');
  const s = (seconds % 60).toString().padStart(2, '0');
  return `${h}:${m}:${s}`;
};
```

KUVA 25. Ajastin funktio ja kellon formatointi (kuvakaappauskoodista)

4.6 Profiili ja historia

Profiilisivulle tehtiin aluksi vain käyttöliittymä väliaikaisella pohjadatalla. Väliaikaisella datalla ei ollut mitään oikeaa tarkoitusta ja sitä käytettiin yksinkertaisesti vain ulkonäön suunnittelu vaiheessa. Niin kuin muillekin sivuille ensin suunniteltiin sivun pohja ja sen jälkeen itse toiminnot. Profiilissa tarkoituksena oli näyttää käyttäjän käyttäjätunnus tai sähköposti, profiilikuva, viikon kokonais- treenimäärä sekä pylväsdiagrammit viikoittaisille treeneille ja kaloreille (kuva 26).



KUVA 26. Profiili näkymä, jossa tilapäistä dataa (kuvakaappaus sovelluksesta)

Profiilin väliaikainen data korvattiin oikealla workouts datalla sen jälkeen, kun olimme ottaneet historian käyttöön.

Historian käyttöönotto oli tärkeää tässä vaiheessa, jotta käyttäjä saa profiiliin tiedot viikoittaisista suorituksistaan. Historian käyttöönotto vaati suorituksien tallentumisen tietokantaan. Suoritukset tallentuvat tietokantaan, kun käyttäjä viimeistelee treeninsä activeWorkout näkymässä painamalla nappia “Finish Workout”. (Kuva 27.)

```
const addtoworkoutHistory = async () => {
  try {
  let textval = v4();
  let auth = getAuth();
  let user = auth.currentUser;
  console.log(user?.uid);
  let docRef = doc(db, "workoutHistory", textval+user?.uid);

  let date = new Date();
  let formattedDate = `${date.getDate()}.${date.getMonth() + 1}.${date.getFullYear()}`;

  await setDoc(docRef, {
    userid: user?.uid,
    workoutName: workoutName,
    moves: updatedMoves,
    date: formattedDate,
  });
  navigation.goBack();
}
catch (error) {
  console.error('Error adding workout: ', error);
  alert('Failed to add workout.');
}
}
```

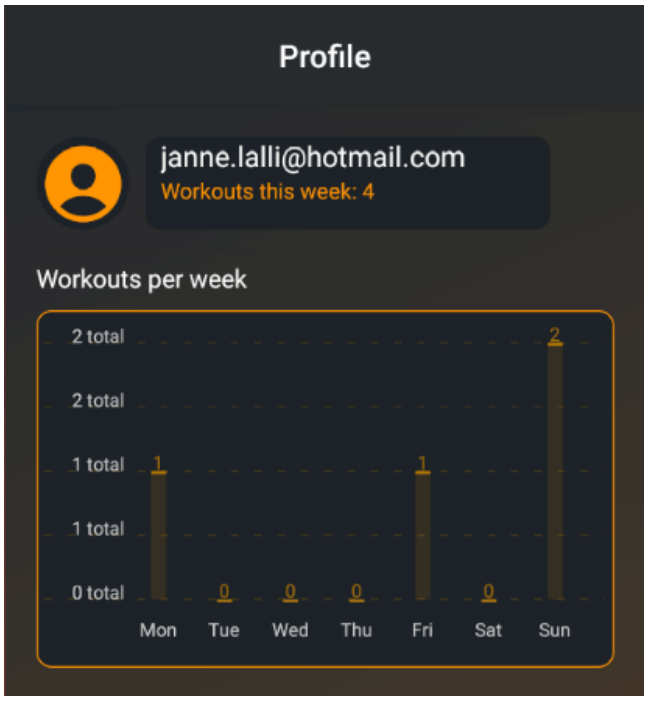
KUVA 27. Treenin tallentuminen historiaan koodissa (kuvakaappaus koodista)

Yllä olevassa kuvassa suoritettu treeni tallentuu Firestore tietokantaan ja sieltä saamme nämä suoritukset näkymään sovelluksen historia näkymässä. Jotta suoritetut treenit saadaan näkyville historia näkymään, täytyy sovelluksen hakea nämä suoritukset tietokannasta. Täytyi siis luoda hakufunktio, joka hakee treenihistorian tietokannasta, tarkistaa käyttäjän, jotta käyttäjä saa henkilökohtaisen historian sekä tallentaa tiedot setWorkoutHistory –tilamuuttujaan (kuva 28).

```
// Fetch Workout History
const fetchWorkoutHistory = async () => {
  setLoadingWorkout(true);
  try {
    if (!userId) {
      console.warn('No user is logged in');
      setLoadingWorkout(false);
      return;
    }
    const workoutHistoryQuery = query(
      collection(db, 'workoutHistory'),
      where('userid', '==', userId)
    );
    const querySnapshot = await getDocs(workoutHistoryQuery);
    const data: WorkoutEntry[] = querySnapshot.docs.map(doc => ({
      id: doc.id,
      ...doc.data(),
    })) as WorkoutEntry[];
    setWorkoutHistory(data);
  } catch (error) {
    console.error('Error fetching workout history:', error);
  } finally {
    setLoadingWorkout(false);
  }
};
```

KUVA 28. Treenihistorian hakufunktio (kuvakaappaus koodista)

Kun suoritukset ovat tallentuneet historiaan, oli aika liittää oikeaa dataa myös profiilisivulle. Profiili sivu ottaa suorituksien lukumäärän historiasta ja tallentaa sen pylväsdiagrammiin, jossa näkyy meneillään olevan viikon suoritukset (kuva 29).



KUVA 29. Sovelluksen profiilisivun dataa

Pylväsdiagrammi haetaan React Nativen kirjastosta nimeltään react-native-chart-kit. Se tuodaan koodiin kutsumalla “import { BarChart } from 'react-native-chart-kit';”. Tämän jälkeen diagrammi pitää konfiguroida haluamilla ominaisuuksilla (kuva 30). Ominaisuuksia voivat olla esimerkiksi värit, pylväiden leveys ja desimaalien paikat. Kun haluamat ominaisuudet on tehty ja diagrammiin haetaan jonkinlaista dataa, voidaan kutsua Barchart koodin return kohdassa. Tässä voi vielä muokata esimerkiksi itse diagrammin kokoa ja akselien nimiä, tärkeää on myös muistaa kutsua konfiguraatio funktiota täällä Barchartin sisällä (kuva 31). React Nativen chart-kit kirjasto on oikein laaja ja hyvä, koska se sisältää myös erilaisia kaavioita. Mahdollisuutena on lisätä myös pylväsdiagrammin lisäksi esimerkiksi piiraskaavio ja viivakaavio. (NpmJs 2022.)

```
const chartConfig = {
  backgroundColor: '#1e2227',
  backgroundGradientFrom: '#1e2227',
  backgroundGradientTo: '#1e2227',
  decimalPlaces: 0,
  color: (opacity = 1) => `rgba(255, 165, 0, ${opacity})`,
  labelColor: (opacity = 1) => `rgba(255, 255, 255, ${opacity})`,
  strokeWidth: 3,
  barPercentage: 0.3,
};
```

KUVA 30. Diagrammin konfiguraatio ja ominaisuudet (kuvakaappaus koodista)

```
<BarChart
  data={workoutData}
  width={screenWidth - 50}
  height={220}
  chartConfig={chartConfig}
  verticalLabelRotation={0}
  yAxisLabel=""
  yAxisSuffix=" total"
  showValuesOnTopOfBars={true} // Optional, to display values on bars
/>
```

KUVA 31. Pylväsdiagrammin kutsuminen return osiossa (kuvakaappaus koodista)

Diagrammin koodissa oli alussa vika, joka ei tallentanut tuloksia diagrammiin, jos ne olivat suoritettu sunnuntaina. Harvemmin tuli muutoksia tehtyä sunnuntaisin, joten tätä ei huomattu aivan heti. Vika oli viikoin aloitus koodissa, jostain syystä se ei laskenut sunnuntaita kuluvaan viikkoon ollenkaan (kuva 32). Tämä virhe korjattiin ottamalla huomioon sunnuntai erikseen koodissa (kuva 33).

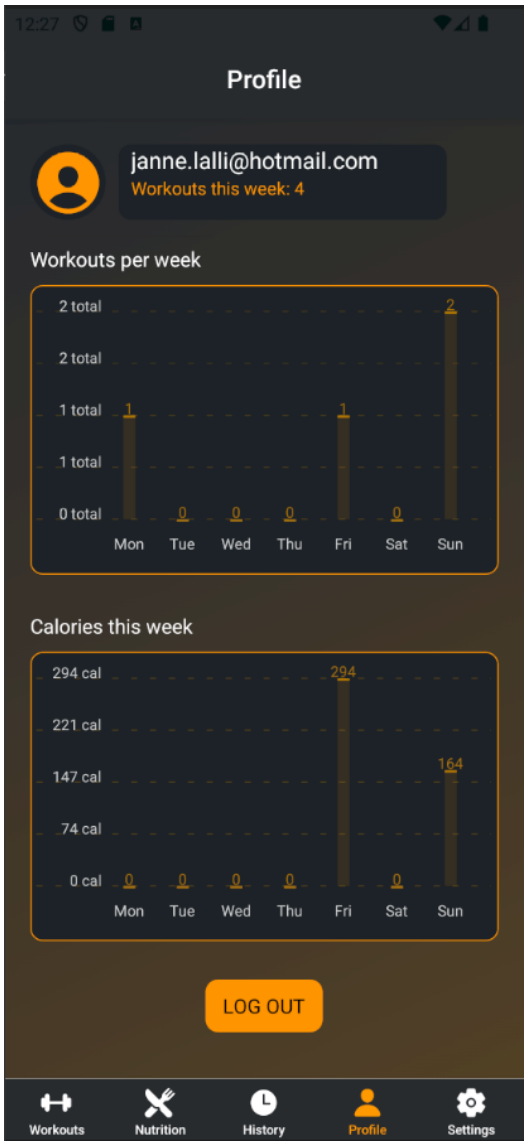
```
const startOfWeek = new Date(
  now.getFullYear(),
  now.getMonth(),
  now.getDate() - now.getDay() + 1 // Adjust to start of the week (Monday)
);
```

KUVA 32. Koodi, joka ei lue sunnuntaina tehtyjä suorituksia (kuvakaappaus koodista)

```
const now = new Date();
const diff = now.getDay() === 0 ? 6 : now.getDay() - 1;
const startOfWeek = new Date(now.getFullYear(), now.getMonth(), now.getDate() - diff);
```

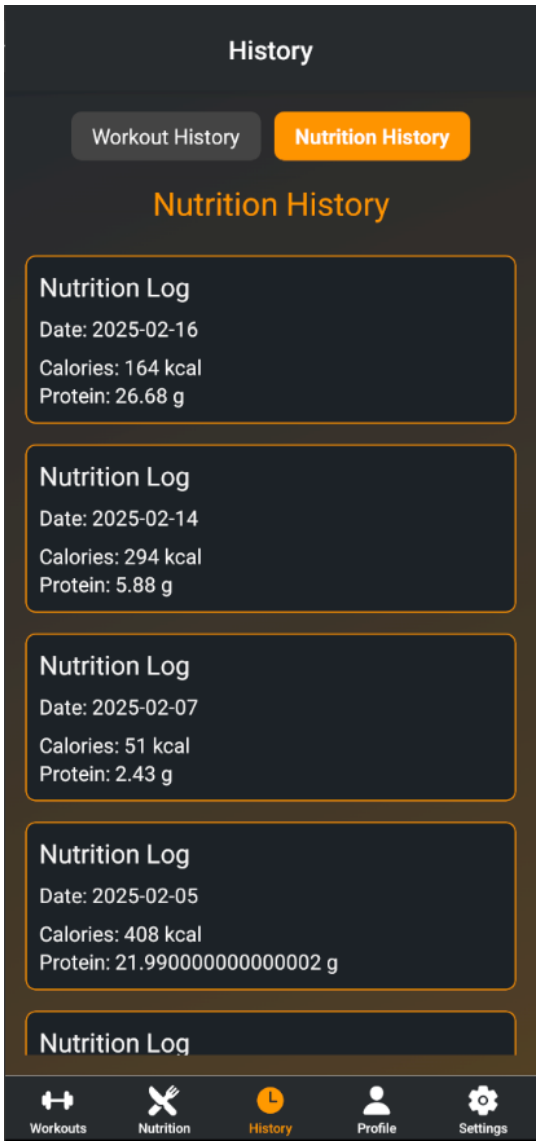
KUVA 33. Toimiva koodi, joka ottaa sunnuntain huomioon (kuvakaappaus koodista)

Profiilisivulle lisättiin myös samanlainen kaavio viikoittaisia kaloreita varten ja uloskirjautumis nappi (kuva 34).



KUVA 34. Profiilisivu kokonaisuudessaan (kuvakaappaus sovelluksesta)

Historiasivulle lisättiin myös ravintotiedoille paikka (kuva 35). Ravintotiedot tallennetaan samanlaisella funktiolla kuin treenin tulokset, ravintotiedoissa erona on vain, että tallennetaan koko päivän tiedot samalle kortille.

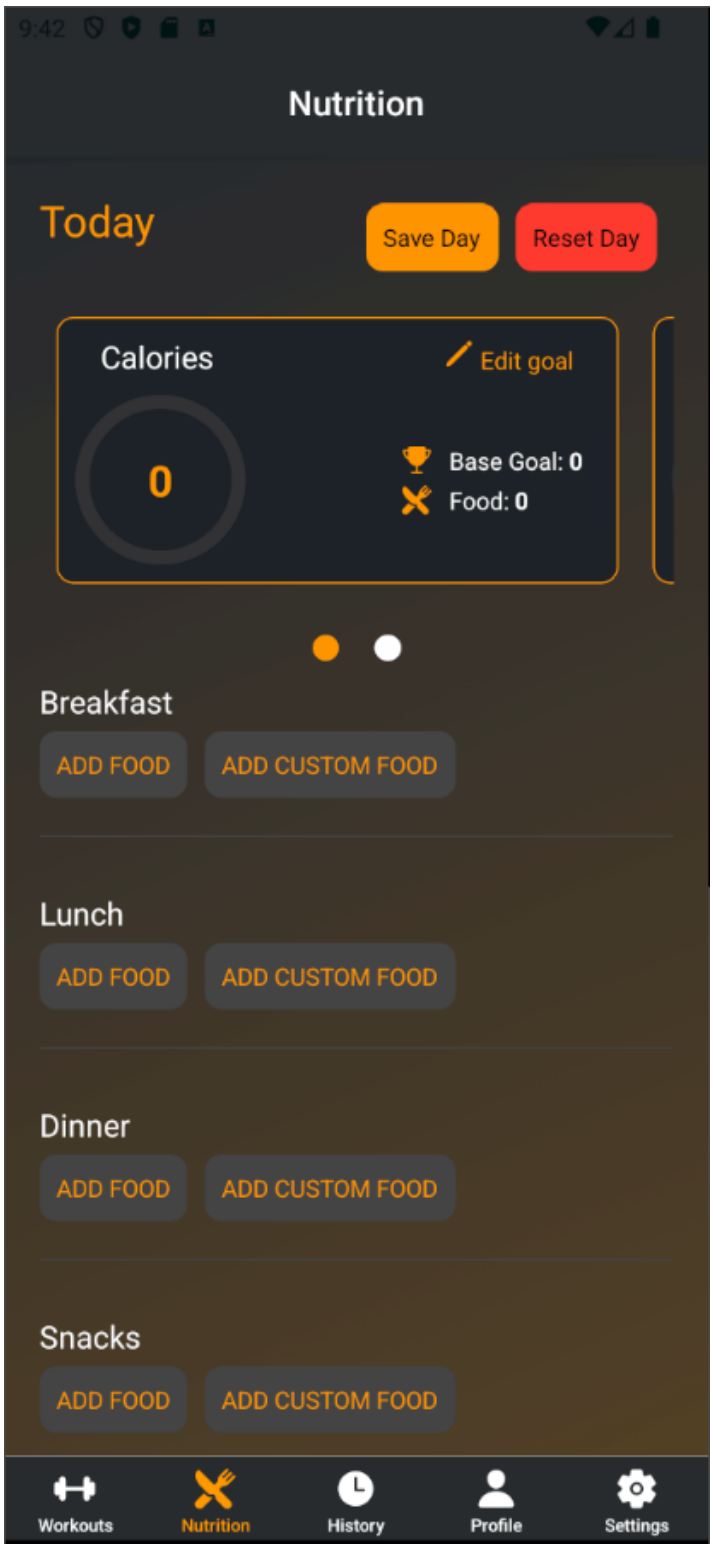


KUVA 35. Historiasivun ravinto-osuus (kuvakaappaus sovelluksesta)

4.7 Ravinto

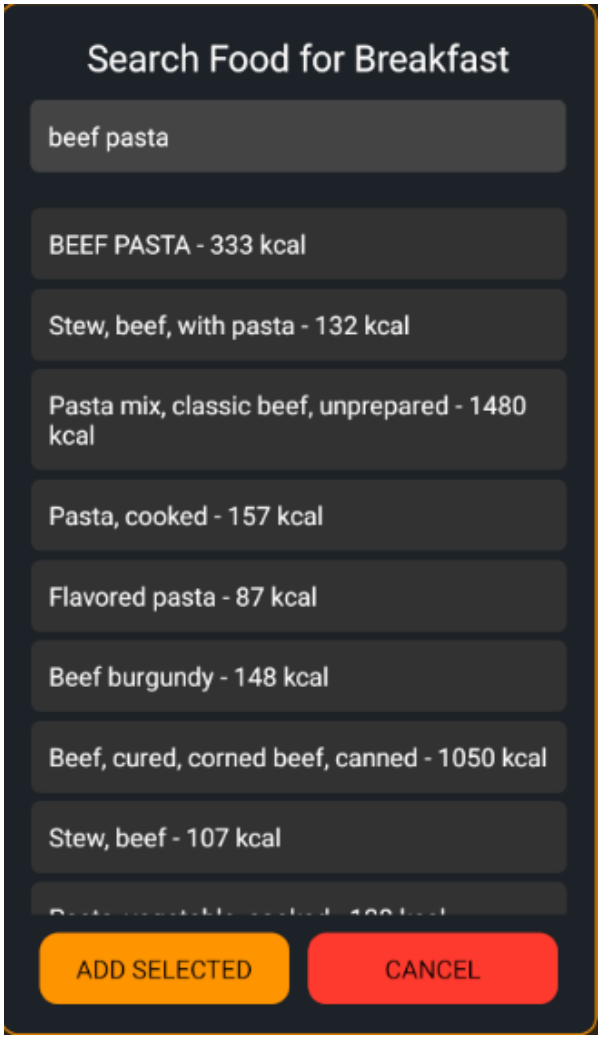
Ravintosivu sisältää sovelluksen toiset päätoiminnot. Ravintosivulla käyttäjä pystyy seuraamaan omaa ruokavaliota, kalorien ja proteiini määrien avulla. Sivu sisältää kortin kaloreille ja proteiineille, jossa käyttäjä voi määrittää kaloreille oman päivittäisen tavoitemäärän. Kalori- ja proteiinikorttien alapuolella on ateriaosiot

päivän jokaiselle aterialle, kuten aamupala, lounas, illallinen ja välipalat (kuva 36).

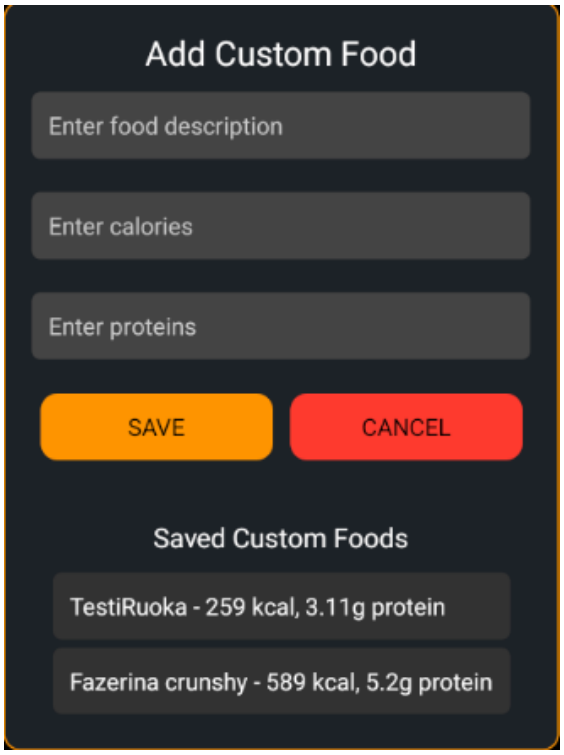


KUVA 36. Ravintosivu kokonaisuudessaan (kuvakaappaus sovelluksesta)

Näihin osioihin käyttäjä pystyy lisäämään syömänsä ruuan, joko Amerikan maatalousministeriön tietokannasta tai halutessaan lisätä täysin oman ruuan kaloreineen ja proteiineineen. Jos käyttäjä haluaa lisätä ruuan USDA-tietokannasta, onnistuu se painamalla painiketta ADD FOOD. Painikkeesta aukeaa modaali, jossa käyttäjä pystyy etsimään ruuan hakusanalla ja modaaliin ilmestyy lista ruuista ja niiden kaloreista (kuva 37). Painikkeesta ADD CUSTOM FOOD käyttäjälle aukeaa oman ruuan lisäämistä varten tehty modaali (kuva 38). Käyttäjän itse lisätyt ruuat tallentuvat myös tähän modaaliin, jotta käyttäjän on helpompi lisätä sama ruoka uudestaan.



KUVA 37. Ruuan haku tietokannasta (kuvakaappaus sovelluksesta)



KUVA 38. Oman ruuan lisäys (kuvakaappaus sovelluksesta)

Ruokien lisäys tietokannasta onnistui lisäämällä sovellukseen funktio nimeltään fetchFoodData. Tässä funktiossa kutsutaan tietokantaa, otetaan tietokannasta saadut tiedot vastaan ja haetaan ruokadataa, kun hakukentässä tapahtuu muutoksia. (Kuva 39.)

```
// Function to fetch food data from USDA FoodData Central API
const fetchFoodData = async (query: string) => {
  try {
    const USDA_API_KEY = Constants.expoConfig?.extra?.usdaApiKey;
    const response = await fetch(
      `https://api.nal.usda.gov/fdc/v1/foods/search?api_key=${USDA_API_KEY}&query=${query}`
    );
    const data = await response.json();
    // USDA returns food items in data.foods
    setSearchResults(data.foods || []);
  } catch (error) {
    console.error('Error fetching food data:', error);
  }
};

// Effect to fetch food data when the search query changes
useEffect(() => {
  if (searchQuery.length > 2) {
    fetchFoodData(searchQuery);
  } else {
    setSearchResults([]);
  }
}, [searchQuery]);
```

KUVA 39. Funktio ruuan hakuun tietokannasta (kuvakaappaus koodista)

Valitsimme USDA:n tietokannan koska se on hyvin laaja, ilmainen ja helposti käyttöönotettava. Tietokannasta on mahdollista saada myös muut ravintoarvot kaloreiden ja proteiinien lisäksi, jos tulevaisuudessa on tarve laajentaa sovelluksen ravintopuolta (USDA 2025).

Käyttäjän lisättyä ruuat haluamallaan tavalla, ruuan kalorit ja proteiinit päivittyvät näiden omille korteille. Ravintoarvojen kokonaismäärien näyttämiseen käytetään React Nativen kirjastosta saatua ympyräkaaviota, nimeltään CircularProgress. Kaavio saadaan käyttöön import lauseella:” import CircularProgress from 'react-native-circular-progress-indicator’; “. CircularProgress toimintoa kutsutaan ravintoainekorttien yhteydessä (kuva 40). CircularProgress toiminnon ulkonäköä voidaan muokata samassa kohtaa missä sitä kutsutaan. (NpmJs 2023.)

```
    <View style={styles.cardContent}>
      <CircularProgress
        key={`${currentCalories}-${baseGoal}`} // Force re-render if values change
        value={item.currentValue}
        maxValue={
          item.type === 'Calories'
            ? item.goal === 0
              ? 1
              : item.goal
            : item.currentValue || 1
        }
        radius={50}
        duration={300}
        progressValueColor="#FF9500"
        activeStrokeColor="#FF9500"
        inActiveStrokeColor="#444"
        inActiveStrokeOpacity={0.5}
      />
      {item.info}
    </View>
  </View>
);
```

KUVA 40. CircularProgress toiminto koodissa (kuvakaappaus koodista)

Ravintoarvoille tehtiin kaikille omat kortit. Tällä hetkellä sovelluksessa on vain kalorit ja proteiinit, mutta tulevaisuutta varten muidenkin ravintoarvojen lisäys on yksinkertaista laajan tietokannan ja ravintokorteille tehdyn toiminnon vuoksi. (Kuva 41.)

```
const progressCards = [
  {
    type: 'Calories',
    currentValue: currentCalories,
    goal: baseGoal,
    info: (
      <View style={styles.goalsInfo}>
        <View style={styles.infoRow}>
          <Ionicons name="trophy" size={18} color="#FF9500" />
          <Text style={styles.infoText}>
            Base Goal: <Text style={styles.boldText}>{baseGoal}</Text>
          </Text>
        </View>
        <View style={styles.infoRow}>
          <Ionicons name="restaurant" size={18} color="#FF9500" />
          <Text style={styles.infoText}>
            Food: <Text style={styles.boldText}>{foodTotal}</Text>
          </Text>
        </View>
      </View>
    ),
  },
  {
    type: 'Proteins',
    currentValue: currentProteins,
    info: (
      <View style={styles.goalsInfo}>
        <View style={styles.infoRow}>
          <Ionicons name="flask" size={18} color="#FF9500" />
          <Text style={styles.infoText}>
            Proteins: <Text style={styles.boldText}>{currentProteins.toFixed(3)}</Text>
          </Text>
        </View>
      </View>
    ),
  },
];
```

KUVA 41. Ravintoarvokorttien funktio (kuvakaappaus koodista)

Ravintokorteille käytetään React Nativen kirjastosta SwiperFlatList toimintoa. Kirjasto otetaan käyttöön kutsumulla sitä lauseella ” import { SwiperFlatList } from 'react-native-swiper-flatlist'; ”. (NpmJs 2025.)

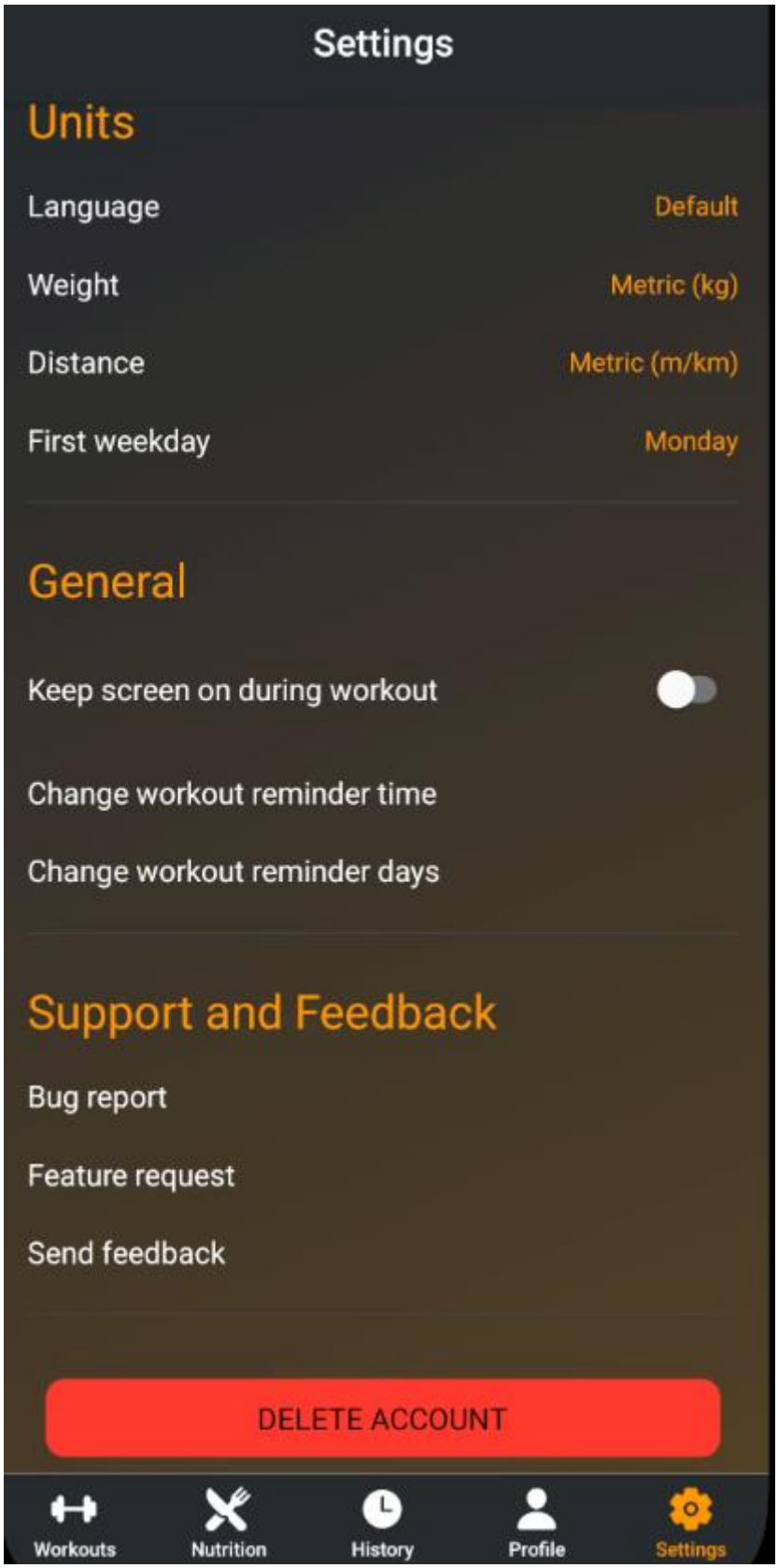
SwiperFlatListin sisällä kutsutaan ravintoarvokortteja ja annettiin muita toimintoja kuten kontin ja sivutuksen tyylit (kuva 42).

```
{/* Carousel for Progress Cards */}
<SwiperFlatList
 data={progressCards}
 renderItem={renderProgressCard}
 showPagination
 paginationActiveColor="#FF9500"
 paginationDefaultColor="#fff"
 contentContainerStyle={{ alignItems: 'center' }}
 paginationStyle={{ position: 'relative', marginTop: 10 }}
/>
```

KUVA 42. SwiperFlatListin toiminnot (kuvakaappaus koodista)

4.8 Settings

Settings näkymä sisältää useita asetuksia kuten sovelluksen kielen, paino ja pituus yksiköiden valinnan (kuva 43). Suurin osa näistä valinnoista on kuitenkin tämän projektin laajuuden ulkopuolella, mutta niille on käyttöliittymä luota valmiiksi jatkokehitystä varten. Tämän projektin kannalta näkymän tärkein ominaisuus on käyttäjän poistaminen. Käyttäjän poistaminen on tärkeä ominaisuus varsinkin, jos jatkokehityksen yhteydessä sovellus halutaan julkaista Google Play Storeen tai Applen App storeen. Google Play Storen vaatimuksissa kerrotaan, että jos sovelluksessa pystyy luomaan käyttäjän, vaaditaan myös sovelluksen sisäinen käyttäjän poisto toiminnallisuus ja käyttäjään liittyvien tietojen poistaminen (Play Console Ohjeet). Applen App store sisältää vastaavankaltaiset vaatimukset (developer apple).

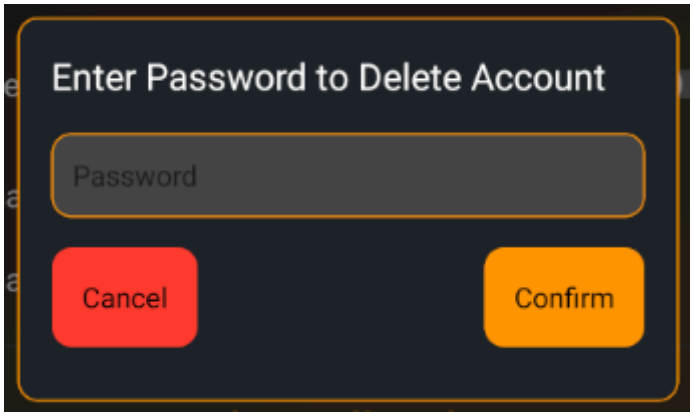


KUVA 43. Kuvakaappaus Settings näkymästä (kuvakaappaus sovelluksesta)

Euroopan Yleinen tietosuoja-asetus eli GDPR myös vaatii, että rekisteröidyllä käyttäjällä on oikeus poistaa tietonsa (digiturvamalli). Amerikasta löytyy myös vastaavia vaatimuksia eri osavaltioissa, kuten Kalifornian California Consumer Privacy Act (CCPA) joka on hyvin samankaltainen, kuin GDPR euroopassa. Myös näitä syitä mukaan lukematta, on hyvä poistaa turhat tiedot, jos käyttäjä poistaa tilin. Tällä saadaan pienennettyä tietokannan kokoa.

Nämä vaatimukset huomioon ottaessa käyttäjän poisto toiminnallisuudessa täytyy myös poistaa tietokannasta Käyttäjän luomat treenipohjat, treenihistoria, ravinto historia sekä itseluodut ruoat ravinnon laskentaan.

Kun käyttäjä painaa Delete Account painiketta, ohjelma vaatii käyttäjää antamaan tilin salasanan (kuva 44). Salasanaa vaaditaan uudelleen, sillä Firebase sisältää vaatimukset, että käyttäjä on uudelleen autentikoitu hiljattain turvallisuudelle herkissä toiminnoissa kuten käyttäjän poisto tai salasanan vaihtoa tehdessä (Firebase 2025). Tämä tuli ilmi toiminnallisuuden testaamisessa, kun käyttäjä oli kirjautunut sisään noin 10 minuuttia, ja poistoyrityksen yhteydessä saattin virhekoodi ja konsoliin ilmoitus "This operation is sensitive and requires recent authentication. Log in again before retrying this request.".



KUVA 44. Salasanan vaatimus käyttäjän poistoon (kuvakaappaus sovelluksesta)

Kun käyttäjä on painanut käyttäjänpoisto painiketta ja kirjoittanut salasanansa, kutsutaan uudelleen autentikointi funktiota (kuva 45). Jos uudelleen autentikointi onnistuu, ohjelma suorittaa käyttäjän tietojen poiston kaikista Firebasen

kokoelmista. Käyttäjän tietojen poistaminen kokoelmista onnistuu helposti, koska jokaisessa kokoelmassa dokumentit sisältävät saman käyttäjäkohtaisen userid -kentän. Tätä käyttäen voidaan for loop -funktiota käyttäen käydä läpi kaikki kokoelmat ja poistaa käyttäjän tiedot. tätä toiminnallisuutta käyttäen on myös jatkokehityksessä mahdollisesti ilmeneviä uusia kokoelmia lisätä käyttäjätietojen poistoon. Kun käyttäjän tiedot ovat poistettu kokoelmista, sovellus poistaa käyttäjän tietokannasta ja palaa kirjautumis- näkymään (kuva 46).

```
try {
  const credential = EmailAuthProvider.credential(user.email!, password);
  await reauthenticateWithCredential(user, credential);
} catch (error) {
  console.error('Error deleting account:', error);
  alert('Error deleting account. Please check your password.');
  return;
}
```

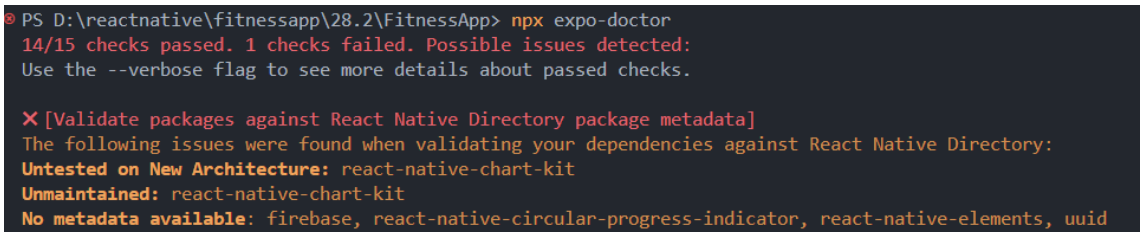
KUVA 45. Uudelleen autentikointi käyttäjän poistoa varten (kuvakaappaus koodista)

```
  const collections = ['userWorkouts', 'workoutHistory', 'customFoods', 'dailyLogs'];
  for (const collectionName of collections) {
    const colRef = collection(db, collectionName);
    const q = query(colRef, where('userid', '==', user.uid));
    const querySnapshot = await getDocs(q);
    querySnapshot.forEach(async (doc) => await deleteDoc(doc.ref));
  }
  await deleteUser(user);

};
```

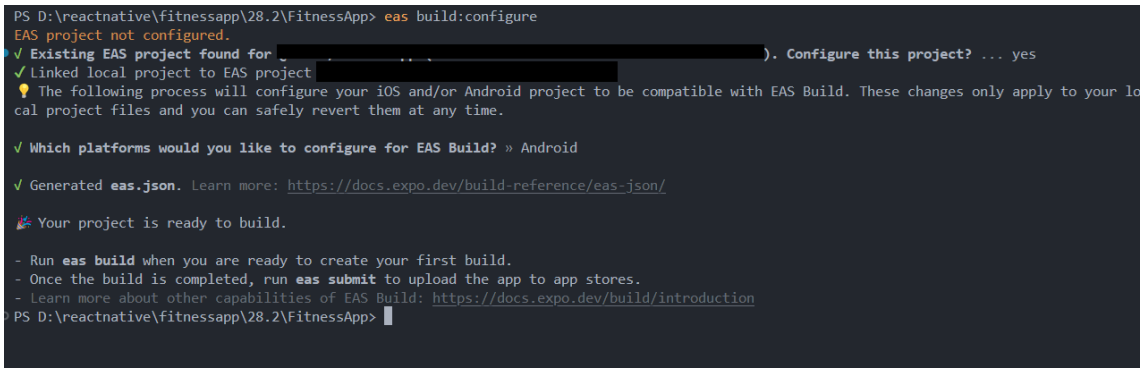
KUVA 46. Käyttäjätietojen ja käyttäjän poisto funktio (kuvakaappaus koodista)

5 TESTAAMINEN

Kun sovelluksen pääominaisuudet olivat valmiita, treenaus- ja ravintopuoli ja sovellus oli todettu toimivaksi React Nativen kevyellä kehyksellä aloitimme testaamisen rakentamalla sovelluksen kokonaisversion. Päätimme tässä vaiheessa rakentaa ladattavan apk eli Android Package version sovelluksesta. Rakentamiseen seurasimme Expon tarjoamaa eas build palvelua. Ennen rakennusta varmistimme, että kaikki käytetyt kirjastot toimivat Expon kanssa ja ovat oikeita versiota. Tämä suoritettiin lataamalla Expo-doctor kirjasto, jolla tarkistettiin, että käytetyt kirjastot ovat oikeat versiot ja yhteensopivia Expon kanssa. Expo-doctor tarkistuksessa saatiin päivitettyä paketit oikeisiin versioihin, mutta yksi paketti antoi varoitusta (kuva 47). Olimme aiemmin todenneet, että tämä ei kuitenkaan haitannut rakennusta. Tämän jälkeen seurasimme Expon tarjoamia ohjeita apk:n rakennukseen, jotka löytyivät heidän sivuiltaan (Expo 2024). Ohjeita seuratessa asensimme eas-cli kirjaston, jonka kautta sovellus saadaan rakennettua. Kirjaston asennuksen jälkeen, vaaditaan Expo käyttäjän luontia ja kirjautumista sisään konsolin kautta. Tämän jälkeen täytyy luoda konfiguraatio tiedostot projektille, jossa määritellään, mille alustoille projekti halutaan rakentaa. Tässä vaiheessa rakensimme sovelluksen ainoastaan androidille (kuva 48).

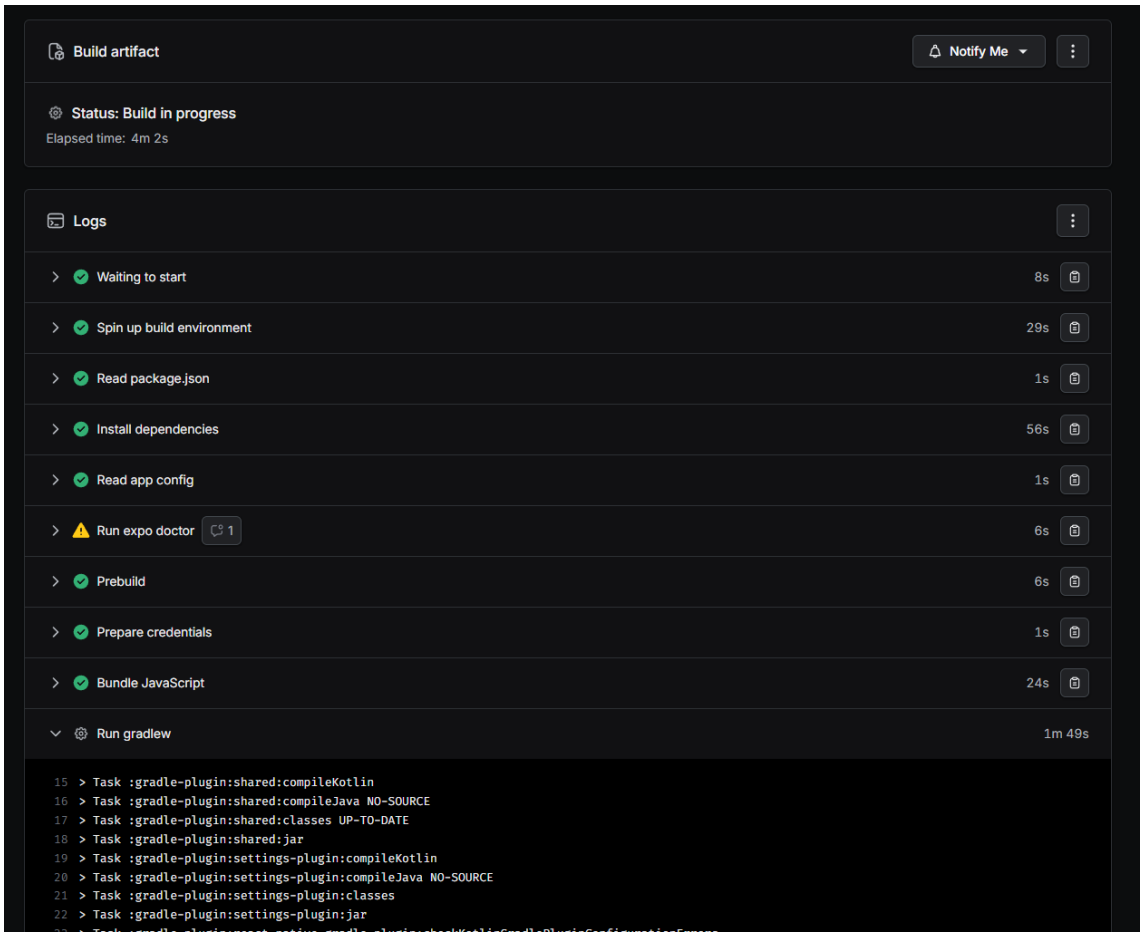


KUVA 47. Expo-doctor tarkistus (kuvakaappaus projektin terminaalista)



KUVA 48. Projektin konfiguraation luonti (kuvakaappaus projektin terminaalista)

Tässä vaiheessa päätimme luoda eas palvelun kautta linkin, josta voi ladata laitteelle suoraan apk, josta sovellus saadaan rakennettua laitteelle. Päätimme käyttää tätä Play storen sijaan tässä vaiheessa, koska se on nopeampi ja helpompi tapa saada testaukseen. Konfiguraation luomaan eas.json tiedostoon muokattiin esikatselu osiota, johon lisättiin android: buildtype: “apk”. Tämän jälkeen ei tarvitse muuta kuin ajaa konsolista eas build -p android --profile preview komento ja odottaa että sovellus rakentuu eas palvelun kautta. Rakentamisen kulkua voi seurata kirjautumulla expo.dev sivustolle, jossa käyttäjä voi katsella missä tilanteessa mennään sekä mahdollisia ongelmia (kuva 49). Rakentamisessa kuluu noin 10–30 minuuttia palvelun ruuhkan mukaan. Kun sovellus rakentui, se myös loi linkin, jonka kautta apk:n pystyi ladata.



KUVA 49. Sovellus rakennusvaiheessa eas palvelun kautta (kuvakaappaus)

Aloitimme testailun omilla laitteilla ja jaoimme myös linkin muutamille kavereille, jotka harrastavat salilla treenaamista ja joilla on kokemusta vastaavista sovelluksista. Ajan puutteesta emme kerenneet luoda Microsoft Forms -kyselyä tai vastaavaa, vaan kysyimme mikä sovelluksessa toimi tai mikä ei, mahdollisia koodivirheitä, kehityskohteita ja ehdotuksia.

Palaute oli enimmäkseen positiivista. Käyttäjät kehuivat varsinkin sovelluksen ulkonäköä. Myös sitä, että sekä treeni ja ravintopuoli löytyvät samasta sovelluksesta kehuttiin. Palautteessa saimme myös hyviä huomioita sovelluksen puutteista sekä hyviä ehdotuksia jatkokehitykselle. Osa näistä meillä oli jo tiedossa mutta uusiakin ideoita saimme.

Aktiivisen treenin näkymässä huomattiin ongelma, että liikkeitä ei voi poistaa. Tämä olisi hyvä jatkossa implementoida, varsinkin kun se löytyy jo Treenin luonti

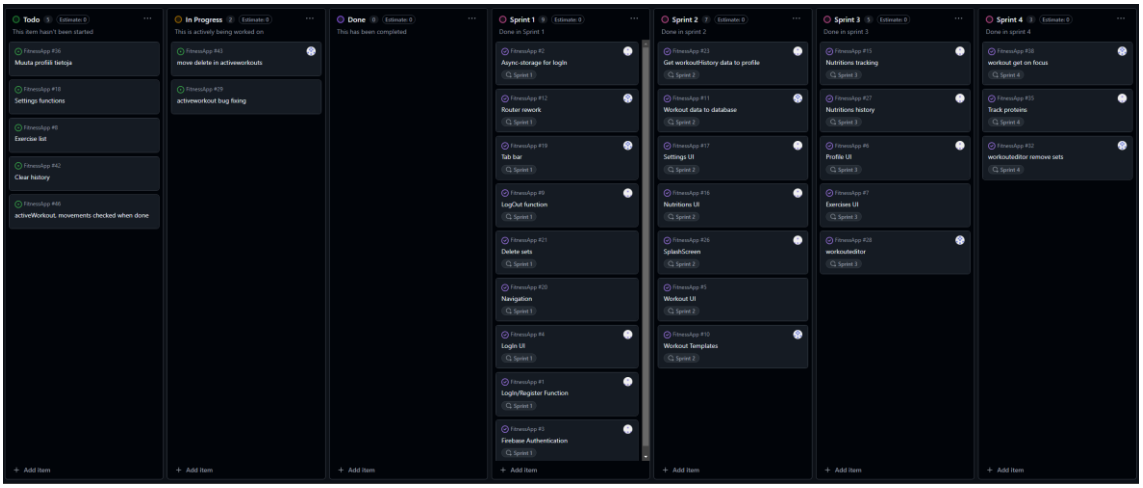
näkymässä. Myös sarjoille toivottiin kuittauspainiketta, kun sarja on tehty, joka helpottaisi seurantaa, sekä treenin lopettaessa sovellus voisi varoittaa, jos kaikkia sarjoja ei ole tehty.

Ravinto näkymään toivottiin, että haettu ravinto vastaisi enemmän suomalaista tai eurooppalaista ravintoa, kuin sovelluksessa käytetty amerikkalainen API. Myös historia näkymässä huomattiin päivitys ongelmia, jos käyttäjä juuri suoritti treenin tai lisäsi ravintoa.

6 POHDINTA

Projektin lopussa, olimme molemmat tyytyväisiä aikaansaatuun tuotokseen. Vaikka aivan kaikkia toivottuja ominaisuuksia emme kerennetkään saada toteutettua, Sovelluksen pääominaisuudet saimme tehtyä ja sovellus on toiminnallisesti valmis. Jos päätämme jatkaa projektia, meillä on selvät tavoitteet ja jatkokohdat mistä jatkaa. Opimme myös työskentelemään tiiminä. Molemmat meistä hoitivat osansa hyvin ja ajoissa.

Hyödynsimme GitHubia projektissa sekä versionhallintaan, että työnjakoon Kanban-taulua käyttäen (kuva 50). Kanban-tauluun lisäsimme tarvittavia ominaisuuksia ja huomattuja virheitä. Pidimme aina viikoittain palaverin, jossa päätimme mitä kukin ottaa työnalle Kanban-taulusta, niin että listaan jäi kuitenkin vielä lisätehtäviä, jos ehdimme tehdä sovitut tehtävät.



KUVA 50. Projektin Kanban-taulu (kuvakaappaus projektin GitHub sivulta)

Projektin tulevaisuutta miettiessä tiedossa on jo useampia lisäyksiä. Asetuksien lisäys sovellukseen on seuraava hyvin tärkeä päivitys. Tällä hetkellä asetuksissa on ainoistaan käyttäjän poisto toiminto ja pohjat tulevaisuutta varten lisättäviin asetuksiin, kuten kielen vaihtoon ja ilmoituksiin.

Useamman kielen lisääminen sovellukseen on tärkeää käyttäjäystävällisyyden sekä sovelluksen käyttäjämäärän lisäämisen suhteen. Tutkimme valmiiksi jo, miten saisimme lisättyä toisen kielen projektiin ja mitä kirjastoja tämä vaatii.

Yksi hyvä tapa lisätä useampi kieli React Native sovellukseen on käyttää i18Next kansainvälistämiskehystä. I18Next tarjoaa laajennuksia, joiden avulla voidaan tunnistaa käyttäjän kieli, ladata käännöksiä, tallentaa käännöksiä ja laajentaa toiminnallisuutta jälkikäsittelyn avulla. (I18Next documentation 2025.)

I18Next otetaan käyttöön samanlailla kuin muutkin kirjastot. Ensin asennetaan se projektiin lauseella “npm i react-i18next i18next”. Sen jälkeen kutsutaan kirjasto koodiin lauseilla “import i18n from 'i18next';” ja “import {initReactI18next} from 'react-i18next';”. Tämän jälkeen luodaan halutuille kielille omat Json tiedostot projektiin, nämä tiedostot auttavat organisoimaan ja hallitsemaan käännösavaimia. (Prashant Telangi 2024.)

Toinen tulevaisuudelle tärkeä toiminto on liikelista, mistä käyttäjä voi valita treeneillensä liikkeet, nähdä liikkeistä pienen infotekstin ja mahdollisesti ohjaavan kuvan liikkeestä. Tämä parantaisi käyttäjäystävällisyyttä ja auttaisi vastaa aloittaneita treenaajia tietämään oikeat liikkeet.

Myös olemassa olevia toimintoja olisi tarpeen hioa paremmaksi. Kuten ravintoarvoja seuraavia kortteja voisi parannella antamaan enemmän tietoja. Tämä onnistuisi esimerkiksi käyttämällä saman react-native-circular-progress-indicator kirjaston toista toimintoa nimeltään CircularProgressBase. Tällä toiminnolla voisi yhdistää useamman ympyräkaavion samalle kortille. (NpmJs, CircularProgressBase 2023.)

Aktiivisen treenin näkymässä hyvä toiminto jatkokehityksen kannalta olisi lisätä painike, jolla kuitata, kun sarja on valmis. Tämä helpottaisi treenin seurasta käyttäjille, jolloin käyttäjän ei tarvitse itse muistaa montako sarjaa on tehnyt ja montako on vielä jäljellä. Lisäksi treenin lopettaessa olisi hyvä, jos sovellus tarkistaisi, että käyttäjä on kuitannut kaikki sarjat, eikä vahingossa lopeta keskeneräistä treeniä.

Pohdimme myös julkaisuun liittyviä asioita projektin aikana. Kuten julkaisemmeko useammalle alustalle vai ainoastaan Androidin play kauppaan. Kuitenkin päädyimme tulokseen, jossa julkaisu tapahtuu myöhemmin sovelluksen kehityksen edettyä. Ja todennäköisesti vain Androidin play kauppaan, johtuen kustannuksellisista ja käytännöllisistä syistä. Googlen Play kauppaan on maksettava 25 euron rekisteröinti maksu, jotta kaupan sisälle voi julkaista sovelluksia. Applen App store vaatii vuosittaisen 99 euron maksun sovelluksien julkaisuun ja ylläpitämiseen. (Sphinx 22.11.2024.)

Sovellukselle olisi tarkoituksena myös keksiä täysin uusi nimi. Tämänhetkinen nimi Fitness App, ei ole kovin yksilöllinen ja voi sekoittua muiden samankaltaisten sovelluksien kanssa. Joten ennen tulevaisuudessa tapahtuvaa julkaisua täytyy sovellukselle keksiä täysin oma nimi. Nimestä olisi hyvä saada jo idea minkälainen sovellus on kyseessä. Nimi voi toisaalta olla mitä vain ja jos nimen perään lisää sanan fit niin käyttäjät tietävät, että kyseessä on fitness sovellus.

Vaikka projektin suunnittelu vaiheessa emme käyttäneet hirveän paljon aikaa, projekti eteni hyvin ilman suurempia ongelmia. Tätä helpotti se, että sovellus oli käytännöllisesti jaettu kahteen pääosaan, joista molemmat tekivät toisen osan.

LÄHTEET

Ahmed, S. 16.1.2025. Market share of mobile operating systems worldwide from 2009 to 2024, by quarter. Luettavissa: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/. Luettu: 2025.

Apple Developer 2025. Offering account deletion in your app. Luettavissa: https://developer.apple.com/support/offering-account-deletion-in-your-app/. Luettu: 24.2.2025.

Babich, N. 25.1.2017. The Underestimated Power Of Color In Mobile App Design. Smashing Magazine. Luettavissa: https://www.smashingmagazine.com/2017/01/underestimated-power-color-mobile-app-design/. Luettu: 29.1.2025.

Expo Create your first build 27.12.2024 Luettavissa: https://docs.expo.dev/build/setup/ Luettu: 28.2.2025.

Figma Learn 2019. What is Figma? Luettavissa: https://help.figma.com/hc/en-us/articles/14563969806359-What-is-Figma. Luettu: 21.1.2025.

Firebase 27.1.2025. Firebase Authentication. Luettavissa: https://firebase.google.com/docs/auth. Luettu: 27.1.2025.

Firebase 21.2.2025. Manage Users in Firebase. Luettavissa: https://firebase.google.com/docs/auth/ios/manage-users. Luettu: 24.2.2025.

Gaba, I. 13.8.2024. What is GitHub And How To Use It? Simplilearn. Luettavissa: https://www.simplilearn.com/tutorials/git-tutorial/what-is-github. Luettu: 27.1.2025.

Ideo Software 2025. React Native – What is Expo and is it worth using? Luettavissa: https://www.ideosoftware.com/blog/react-native-what-is-expo-and-is-it-worth-using,275.html. Luettu: 27.1.2025.

I18Next documentation 21.2.2025. Introduction. Luettavissa: https://www.i18next.com/. Luettu: 28.2.2025.

Lionel, V. 3.5.2023. Cross-platform mobile frameworks used by developers worldwide 2019-2023. Statista. Luettavissa: https://www.statista.com/statistics/869224/worldwide-software-developer-working-hours/. Luettu: 20.1.2025.

mdn web docs. Date. Luettavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date Luettu: 20.2.2025.

NpmJs 2022. React-native-chart-kit. Luettavissa: https://www.npmjs.com/package/react-native-chart-kit. Luettu: 2.2.2025.

NpmJs 2023. react-native-circular-progress-indicator. Luettavissa: https://www.npmjs.com/package/react-native-circular-progress-indicator. Luettu: 22.2.2025.

NpmJs, CircularProgressBase 2023. react-native-circular-progress-indicator. Luettavissa: https://www.npmjs.com/package/react-native-circular-progress-indicator. Luettu: 4.3.2025.

NpmJs 2025. react-native-swiper-flatlist. Luettavissa: https://www.npmjs.com/package/react-native-swiper-flatlist?activeTab=readme. Luettu: 24.2.2025.

NpmJs 2025. Uuid. Luettavissa: https://www.npmjs.com/package/uuid#options-handling-for-timestamp-uuids. Luettu: 5.2.2025.

Oikeus tietojen poistamiseen (”oikeus tulla unohdetuksi”). Fakta digiturvamalli päivitetty: 22/5/2017. Luettavissa: https://fakta.digiturvamalli.fi/gdpr-asetus/17-oikeus-tietojen-poistamiseen-oikeus-tulla-unohdetuksi. Luettu: 24.2.2025.

Prashant Telangi 10.12.2024. Adding Multilingual Support to Your React Native App. Mobisoft. Luettavissa: https://mobisoftinfotech.com/resources/blog/add-multilingual-support-react-native. Luettu: 28.2.2025.

Play Console Ohjeet 2025. Käyttäjä data. Luettavissa: https://support.google.com/googleplay/android-

developer/answer/10144311?visit_id=638760030042201096-4290685740&rd=1#account_deletion. Luettu: 24.2.2025.

React v19 2024. useState. Luettavissa: https://react.dev/reference/react/useState. Luettu: 16.2.2025.

Rob Bonta 2025. California Consumer Privacy Act (CCPA). Attorney General. Luettavissa: https://oag.ca.gov/privacy/ccpa. Luettu: 24.2.2025.

Sphinx 22.11.2024. The Cost to Publish an App on Google Play vs. the Apple Store. Medium. Luettavissa: https://medium.com/@sphinxshivraj/the-cost-to-publish-an-app-on-google-play-vs-the-apple-store-76a576f5f58d. Luettu: 7.3.2025.

U.S. DEPARTMENT OF AGRICULTURE (USDA) 2025. FoodData Central API Guide. Luettavissa: https://fdc.nal.usda.gov/api-guide. Luettu: 30.1.2025.