

Lassi Tihinen

URHEILUSOVELLUKSEN LUOMINEN EXPO-VIITEKEHYKSELLÄ

URHEILUSOVELLUKSEN LUOMINEN EXPO-VIITEKEHYKSELLÄ

Lassi Tihinen
Opinnäytetyö
Kevät 2024
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, Ohjelmistokehityksen suuntautumisvaihtoehto

Tekijä: Lassi Tihinen

Opinnäytetyön nimi: Urheilusovelluksen luominen Expo-viitekehysellä

Työn ohjaaja: Jouni Juntunen (OAMK)

Työn valmistumislukukausi ja -vuosi: Kevät 2024

Sivumäärä: 33

Opinnäytetyön aiheena on urheilusovelluksen luominen Expo-viitekehystä käyttäen. Työlle ei ollut toimeksiantajaa. Työhön kehitettiin mobiilisovellus, joka sisältää helppokäyttöisen käyttöliittymän.

Teoriaosuus käsittelee projektin tekoon tarvittavia esitetietoja työn toteutuksen työkaluina. Osuus käsittelee React Nativea, Firebaseia, Expoa ja Expon kirjastoja. Osuuden aineistona käytettiin osittain viitekehysten Expo-sivustoja, mutta myös laajalti internet-lähteistä löytyvää tietoa.

Raporttiosuudessa käydään läpi sovelluksen toteutus, aloittaen uuden projektin luonnista. Osuudessa edetään näyttämällä sovelluksen eri osat ja selittämällä niiden merkitykset projektissa ohjelmakoodien muodossa.

Opinnäytetyön tulokseksi muodostui urheilusovellus, joka mahdollisti erinäisten juoksuilastojen ja karttatietojen seurannan. Tavoitteena oli luoda hyödyllinen sovellus, jotta kehittäisin omia taitojani Expon ja JavaScript-kielen kanssa. Sovellukselle voi kehittää paljon uusia ominaisuuksia tarpeen vaatiessa.

Asiasanat: ohjelmistokehitys, mobiilisovellus, käyttöliittymä, React, React Native, Expo, Firebase

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

Author(s): Lassi Tihinen
Title of thesis: Creating a sports app with the Expo framework
Supervisor(s): Jouni Juntunen (OAMK)
Term and year when the thesis was submitted: Spring 2024
Number of pages: 33

The subject of the thesis was to create a sports app with the Expo software development framework. This thesis did not have an orderer. An easy to use interface was developed as the result of this thesis.

The theory section of the thesis deals with the prerequisite knowledge needed to create a mobile app with the Expo framework. The section goes through the topics of React Native, Firebase, Expo and Expo libraries. The material of the section was partly Expo websites, but also widely available information from internet sources.

The report section of the thesis handles the concrete creation of the sports app, and starts with the creation of an app in Expo. Then it continues with showing the different parts of the app and explaining what each part does in the form of code.

As the result of this thesis, a sports app was created. It tracks different kinds of running stats and tracks the user on a map. The goal was to create a useful application that would develop my own skills with Expo and the programming language JavaScript. The app can be developed further with new features if there's ever a need for such measures.

Keywords: software development, mobile application, user interface, React, React Native, Expo, Firebase

SISÄLLYS

1	JOHDANTO	6
2	TYÖN TOTEUTUKSEN TYÖKALUT	8
2.1	Expo	8
2.2	Firebase	10
2.3	Expo-kirjastot.....	12
3	URHEILUSUORITUKSET-SOVELLUKSEN TOTEUTUS	16
3.1	Expo-projektin aloitus	16
3.2	Näytöt ja navigointi	18
4	POHDINTA	29
	LÄHTEET.....	31

1 JOHDANTO

Opinnäytetyön tavoitteena on ulkoilusovelluksen suunnittelu, teko ja julkaiseminen. Sovellus seuraa käyttäjän urheilusuorituksia. Sovelluksen keskeisin tehtävä on seurata lenkkipolkua GPS-paikannuksella. Seuraavaksi tärkeimmät tehtävät ovat reittikartan laatiminen, reitin ajastus ja keskinopeuden laskeminen tehdyille matkalle. Juoksujen säännöllisyys on tärkein osa juoksukehitystä, ja säännöllisyyden rakentaminen auttaa hyvään juoksu-olosuhteeseen tähdätessä (1). Valitut ominaisuudet ovat hyödyllisiä seurattavia aiheita, kun aloitteleva tai jo kokenut juoksija haluaa seurata treeniensä säännöllisyyttä joko yksittäisen treenin sisällä tai yleisesti trendinä.

Käyttäjän näkemä käyttöliittymä luodaan Expo-viitekehityksellä, joka toimii JavaScript-kielellä. Tarvittavat taustatiedot tallennetaan palvelimelle Firebase-palvelun avulla. Firebasea käytetään käyttäjätietojen ja treenitulosten tallennukseen ja todentamiseen. Sovelluksessa käytetään Google Maps-karttapalvelua React-kirjaston kautta. Opinnäytetyössä käsitellään myös julkaisuvaihe. Versiohallintaa suoritetaan GitHub-palvelulla.

Tavoitteena on julkaista sovellus Googlen Play-kauppaan ja luoda käyttäjälle selkeä käyttöliittymä. Sovellusta ei julkaista Applen App Storeen, sillä sen tehdäkseen kehittäjällä pitää olla Apple-tili ja hänet tulee rekisteröidä Applen kehittäjäohjelmaan (2). Opinnäytetyön tekijällä ei ole tiliä eikä rekisteröintiä ohjelmaan.

Käyttöliittymän on mahdollistettava treeniseuranta ensimmäisessä kappaleessa mainittujen sovellustehtävien mukaan. Sovelluksen tulee sisältää kirjautumisjärjestelmä, jotta sitä voi käyttää eri tileillä eri käyttäjät. Käyttäjien tallentamat tiedot pitää olla käsiteltävissä vain näiden samojen käyttäjien toimesta.

Lenkkisovelluksen luominen päätettiin aiheeksi, koska se on yhden henkilön toteutettavissa oleva projekti. Samalla esitellään tavat ja menetelmät lopputuloksen julkaisemiseksi Googlen Play-kauppaan yleisohjeena. Opinnäytetyön tietoperustana käytetään ensisijaisesti erilaisia juoksemiseen ja ohjelmoimiseen liittyviä verkkojulkaisuja, ja selata yleisesti verkkoa tiedon löytämiseksi. Lopputuloksen julkaisemiseen käytetään ohjeita Googelta, Expolta ja internetlähteistä.

Opinnäytetyötä seurataan Trello avulla. Trello on virtuaalinen ilmoitustaulu, jossa voidaan järjestellä sekä priorisoida tehtäviä ja ajatuksia korttien muodossa. Trellossa seurataan senhetkistä työvaihetta ja työn alla olevia asioita. Trelloon kirjataan kaikki sovelluskehitysvaiheen mahdolliset ideat, jotka voivat olla tarpeen.

Opinnäytetyössä käytetään hyväksi ChatGPT-palvelua esimerkkikoodien luomiseen, muttei luoteta sen tarjoamiin ratkaisuihin pintapuolisesti. Sen tarjoamat ratkaisut tarkistetaan yleisellä tiedonhalla ja testauksella.

2 TYÖN TOTEUTUKSEN TYÖKALUT

2.1 Expo

Expo on viitekehys, joka tarjoaa lisätoiminnallisuuksia React Nativeen. Viitekehukset ovat rakenteita, joiden tukemana voidaan rakentaa tietokoneohjelmia (3). Viitekehukset eroavat kirjastoista siinä, että kirjastot ovat joukko ennalta määrättyjä koodinpalasia joita voi käyttää oman koodin rakennukseen, kun taas viitekehukset pakottavat kehittäjän tekemään koodin viitekehysten tarvitsemaan malliin tai sovellus itsessään ei toimi (3). Ero tulee selkeäksi, kun vertaillaan näiden kahden väärinkäytöstä koituvia ongelmia. Kun viitekehystä käytetään väärin, niin sovellus itsessään lakkaa toimimasta. Jos kirjastoa käytetään väärin, vain kirjastoa tarvitseva osa sovellusta ei toimi, kuten Google Maps -kartta.

Expo on rakennettu jo aikaisemman toiminnallisuuden päälle, jonka nimi on React Native. React Native on avoimen lähdekoodin JavaScript-viitekehys, joka mahdollistaa mobiilisovelluskehityksen Android- ja IOS-käyttöjärjestelmille sekä selainsovelluksen luomisen yhdellä koodikannalla kaikille sovelluksille (4). React Native itsessään on jatkokehitetty versio alkuperäisestä Reactista (5), joka ei ole tämän opinnäytetyön aihe.

React Native sekä Expo vaativat Node.js-ympäristön toimiakseen, joka taas mahdollistaa JavaScriptin suorittamisen webiselaimen ulkopuolella. Node.js on Runtime Environment, suoritusajan ympäristö. Runtime Environment (RTE) on tila, jossa ohjelmisto suoritetaan. Se määrittelee, mihin projektinlaajuisiin kohteisiin ohjelmasi pääsee käsiksi ja miten hyvin itse projekti suorituu. JavaScriptin aihepiirissä on kaksi RTE:tä, selain ja Node. (6.)

Esimerkkinä internetiselaimessa suoritettavasta RTE-tapahtumasta on se, että jonkun tiedoston HTML-koodia voi lukea näytölle ponnahdusikkunaan. Sovellukset, jotka luodaan ja suoritetaan selaimessa, ovat front-end sovelluksia. Node RTE on erilainen selainversioon verrattuna. Se kehitettiin palvelemaan niitä tilanteita, joissa ei tarvita selainta. Nodeilla voidaan suorittaa JavaScript-koodia full-stack-sovellusten tekoon. Node antaa kehittäjille pääsyn palvelimen tiedostojärjestelmään, tietokantaan ja verkkoon. (6.)

Reactia ylläpitää Meta. Kirjaston käyttämää JavaScript-kieltä hyödynnetään dynaamisen websisälön luomiseen ja hallinnoimiseen (7). React-pohjaisille viitekehyksille ominaista on niiden komponenttipohjainen arkkitehtuuri. Sen avulla kehittäjät voivat luoda käyttöliittymiä uudelleenkäytettävistä ja itsenäisistä komponenteista. Tällainen voi olla esimerkiksi kirjautumisnäkyvä, jota voidaan muotoilla itsenäisesti eri tiedostosijainnissa kuin missä kyseinen komponentti luodaan käyttäjänäkömään. Tämä tekee komponentista uudelleenkäytettävän eri ikkunoihin sovelluksen sisällä.

Expo nopeuttaa ja helpottaa sovelluskehityksen aloitusta. Kehittäjä voi testata sovellustaan Expo-Go-sovelluksessa, puhelinemulaattorilla tai webiselaimessa. Se myös tarjoaa mahdollisuuden kutsua muita tarkastelemaan rakentuvaa sovellusta QR-koodilla tai web-linkillä. Tämä toiminto tekee sovelluksen testauksesta sujuvaa (4).

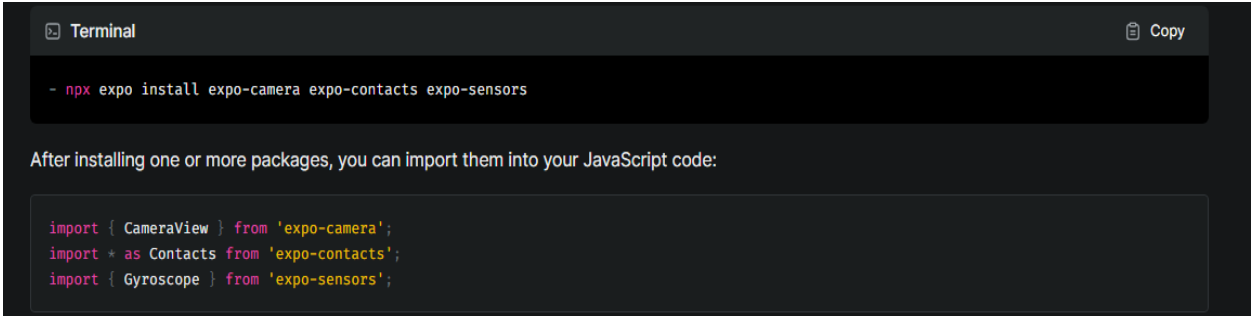
Käytännössä Expo mahdollistaa projektin aloituksen yhdellä koodirivillä ja aloittaa sen toisella koodirivillä. Sen jälkeen kehittäjä voi käyttää omaa puhelintaan tai tietokoneella suoritettavaa emulaattoria muutosten tekemiseen ja testata sovelluksen toimivuutta. Kehittäjä voi toimia mobiililaitteella joko langattomasti WiFin välityksellä tai USB-yhteydellä kohdelaitteeseen liitettynä. Kaikki muutokset päivittyvät reaaliajassa kohdelaitteessa kun ne tehdään, esimerkiksi tallentamalla tiedosto koodinhallintaohjelmissa.

Kuvassa 1 on Expon ja React Nativen yleisimpien ominaisuuksien eroja. (8).

Feature	With expo	Without expo (bare React Native)
Develop complex apps entirely in JavaScript.	✓	✗
Write JSI native modules with Swift and Kotlin.	✓	✗
Develop apps without Xcode or Android Studio.	✓	✗
Create and share example apps in the browser with Snack .	✓	✗
Major upgrades without native changes.	✓	✗
First-class TypeScript support.	✓	✗
Install natively compatible libraries from the command line.	✓	✗
Develop performant websites with the same codebase.	✓	✗
Tunnel your dev server to any device.	✓	✗

KUVA 1. Ominaisuustaulukko Expon sivuilta englanniksi (8).

Expo mahdollistaa kaikkien kohdelaitteen normaaliominaisuuksien käytön sen Expo SDK-referenssin avulla. Näitä ominaisuuksia ovat mm. kamera, gyroskooppi ja kiihdytysanturi. Käyttämättä jääneet ominaisuudet eivät lisää sovelluksen kokoa ylimääräisillä käyttämättömillä tiedostoilla (9). Expo SDK-referenssin (9.) avulla kehittäjä voi asentaa ja tuoda ne projektiinsa kuvan 2 mukaisesti.



```
Terminal Copy  
- npx expo install expo-camera expo-contacts expo-sensors  
  
After installing one or more packages, you can import them into your JavaScript code:  
  
import { CameraView } from 'expo-camera';  
import * as Contacts from 'expo-contacts';  
import { Gyroscope } from 'expo-sensors';
```

KUVA 2. Asennus- ja sisällytysohjeet Expo SDK -sisällöille (9).

2.2 Firebase

Firebase on Googlen tuottama tietokantapalvelu web- ja mobiilikehitykseen, joka toimii Google Cloudin rinnalla. Firebasen voi luokitella opinnäytetyön käyttötarkoituksessa Mobile backend as a Service-palveluksi (MBaaS). Tätä luokittelua käytetään sellaisesta palvelusta, joka on suunniteltu olemaan ”kaikki yhdessä”-ratkaisu mobiililaitteiden backend-sovelluskehityksessä (10).

Opinnäytetyöhön palvelu on hyödyksi, sillä luodakseen projektin tietokannan kehittäjän ei tarvitse itse luoda, suorittaa tai ylläpitää omaa tietokantapalvelinta ja tallentaa näin ollen tietoja omalle laitteelle. Palvelu suorittaa kehittäjän puolesta mm. rekisteröinnin, kirjautumisen sekä istuntojen hallitsemisen (10). Palvelu on myös skaalautuva, eli jos sovelluksen käyttäjäkunta kasvaa, niin sovellus kasvaa käyttäjäkunnan mukana menettämättä suorituskykyä sovelluksen toiminnassa (10).

Firebasen keskeisimpiä tarjottuja palveluja ovat reaaliaikainen tietokanta, tietojen todennus sekä analytiikka. Reaaliaikaisesta tietokannasta on kahta versiota, jotka kantavat Realtime Database- ja Cloud Firestore-nimiä. Molemmat ovat NoSQL-pilvitietokantoja, joka suoraan tarkoittaa ”ei pelkästään SQL”. Firebase on suosituimpien tietokantapalvelujen joukossa React-pohjaisten sovelluksien kehityksessä(11), mutta vasta sijalla 39 verrattaessa kaikkia NoSQL-tietokantapalveluja yleisesti (12).

NoSQL-tietokannat ovat ei-taulukkomuotoisia tietokantoja ja tallentavat dataa dokumenttimuotoisena. Yksinkertaisin esimerkki tällaisesta tietojentallennuksesta on avain-arvopari (kuva 3). Voit esittää käyttäjän iän käyttämällä ikä:25, ja käyttäjän nimeä käyttämällä nimi:Lassi Tihinen. Tässä esimerkissä avaimia ovat ikä ja nimi, ja arvoja ovat 25 ja Lassi Tihinen. (13.)

Data




```
key: value
```

KUVA 3. Tietojen esitys avain-arvoparina (13).

Kun kaikki mahdolliset avain-arvo parit ryhmitellään yhteen, tulee lopputulokseksi dokumentti (kuva 4). Dokumentit helpottavat asiakokonaisuuksien niputtamista, ja asiat löytyvät helpommin, kun ne voi niputtaa suuremmiksi asiakokonaisuuksiksi. Esimerkkinä tilanne, jossa tarvitaan dokumentti nimeltä vaatekaappi, jonka sisällä olevat avain-arvo parit voivat esittää erilaisia vaatekokonaisuuksia tai vaatekappaleita.

Document



```
{  
  key: value,  
  key: value,  
  key: value,  
  key: value  
}
```

KUVA 4. Dokumentin rakenne kuvana (13).

Dokumentteihin voi myös aina lisätä tietokenttiä tarpeen vaatiessa, esim. käyttäjätietojen tallennukseen voidaan lisätä tiedot pituuteen lisäämällä yksi avain-arvo pari. Monta dokumenttia muodostaa

kokoelman. Dokumenttien ei tarvitse olla identtisiä toisiinsa verrattaessa, vaan ne toimivat omina kokonaisuuksinaan. (13.)

Tässä opinnäytetyössä käytetään Firebasen palvelua Cloud Firestore, jossa käytetään datantallennusmetodina dokumenttia. Firestoressa tiedot tallentuvat JSON-dokumentteina pilveen. Tiedot pidetään synkronisoina reaaliaikaisten päivitysten kautta asiakkaille (13).

Firebasen toiminnallisuuksien liittäminen projektiin on helppoa. Kehittäjän tulee ensiksi rekisteröidä sovellus Firebase-palvelun sivustolla, ja sitten liittää Firebase osaksi projektia liittämällä sen konfiguraatitiedosto (kuva 5) koodipohjaan.

```
firebaseConfig.js

import { initializeApp } from 'firebase/app';

// Optionally import the services that you want to use
// import {...} from "firebase/auth";
// import {...} from "firebase/database";
// import {...} from "firebase/firestore";
// import {...} from "firebase/functions";
// import {...} from "firebase/storage";

// Initialize Firebase
const firebaseConfig = {
  apiKey: 'api-key',
  authDomain: 'project-id.firebaseio.com',
  databaseURL: 'https://project-id.firebaseio.com',
  projectId: 'project-id',
  storageBucket: 'project-id.appspot.com',
  messagingSenderId: 'sender-id',
  appId: 'app-id',
  measurementId: 'G-measurement-id',
};

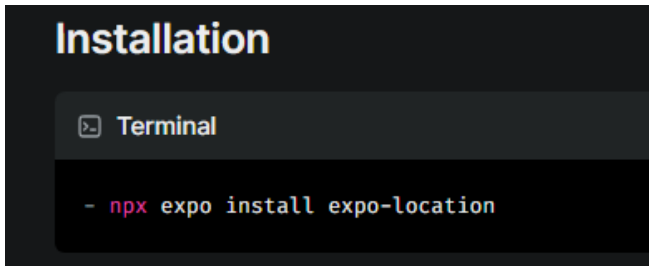
const app = initializeApp(firebaseConfig);
// For more information on how to access Firebase in your project,
// see the Firebase documentation: https://firebase.google.com/docs/web/setup#access-firebase
```

KUVA 5. Firebase konfiguraatitiedoston esimerkki Expon sivuilta (14).

2.3 Expo-kirjastot

Opinnäytetyössä käytetään erilaisia Expo-kirjastoja, joista tärkeimpiä ovat expo-location, expo-sensors sekä react-native-maps. expo-location tarjoaa kehittäjälle kattavan valikoiman työkaluja sijainnin hallintaan ja hyödyntämiseen Expo-sovelluksissa, laajat konfigurointivaihtoehdot sekä etualalla

että taustalla tapahtuvalle sijainnin seurannalle (15). Tämän ja muiden tämän kappaleen kirjastojen asennus toimii samoin kuin aikaisempien Expo SDK -referenssin komennoilla (kuva 6).



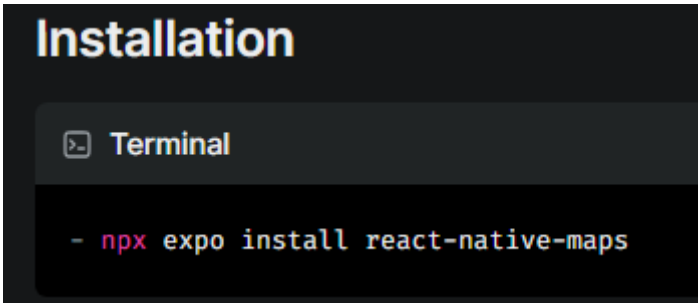
```
Installation  
Terminal  
- npx expo install expo-location
```

KUVA 6. expo-location asennus (15).

Expo-locationia käytetään opinnäytetyössä hakemaan mobiililaitteen sijainti sen GPS-paikantimen avulla. Kirjasto tukee etualalla ja taka-alalla tapahtuvia sijaintipyyntöjä, eli sijaintitietoja voidaan myös lähettää, kun käyttäjällä ei ole sovellusikkuna aukinaisena. Ennen sijainnin lähettämistä sovelluksen pitää pyytää käyttäjältä lupa käyttää sijaintia. Tämä onnistuu käyttämällä kirjastoon sisäänrakennettua metodia `Location.requestForegroundPermissionsAsync()`. Tämä metodi kysyy käyttäjältä hänen suostumuksensa sijaintitietojen käyttöön (15).

Sijaintitietojen lähetys laitteelta sovellukseen toimii metodilla `Location.getCurrentPositionAsync()`. Tämä metodi hakee käyttäjän nykyisen sijainnin kertaluonteisella haulla mukautettavalla tarkkuudella. Tarkkuus vaihtelee kolmen neliökilometrin alueen ja metrin alueen välillä. Metodi hakee sovellukselle käyttäjän nykyisen sijainnin ja yksityiskohtaisia tietoja, kuten tarkat koordinaatit, korkeuden, sijainnin tarkkuuden ja tallennusajan. Se tekee tämän palauttamalla lupauksen, mikä tarkoittaa, että metodi saa sijaintitiedot mobiililaitteelta ja lähettää ne käytettäväksi, kun ne ovat valmiina (15).

React-native-maps on toinen keskeisistä kirjastoista tälle opinnäytetyölle. Se tarjoaa monipuolisen karttakomponentin käyttämällä Google Mapsia Android-sovelluksissa. Expo SDK-referenssiä käytettäessä se mahdollistaa karttanäkymän kehittämisvaiheessa, mutta julkaisuvaiheessa Googlen pitää luoda sertifikaatti karttanäkymän käyttöä varten (16). Kuvassa 7 esitellään kirjaston asennus.



```
Installation  
Terminal  
- npx expo install react-native-maps
```

KUVA 7. React-native-maps-kirjaston asennus (16).

Kirjaston ominaisuuksista opinnäytetyössä käytetään MapView-komponentti, jota käytetään kartan esittämiseen sovelluksessa. Komponentti sisällytetään osaksi sovellusta tuomalla se react-native-mapsista (kuva 8).

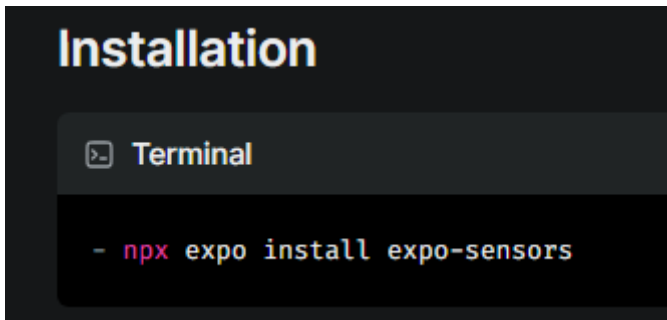
```
import MapView from 'react-native-maps';
```

KUVA 8. MapView-komponentin tuominen react-native-maps-kirjastosta sovellukseen (16).

Komponentille lähetetään erilaisia parametreja, joihin lukeutuvat ref, style ja region. ref on viite karttanäkymään, style on kartan tyylitys, ja region määrittää, mitä aluetta kartalla näytetään (17). MapView on keskiössä opinnäytetyössä, sillä sen avulla käyttäjä näkee missä hän on menossa ja mistä hän on tullut.

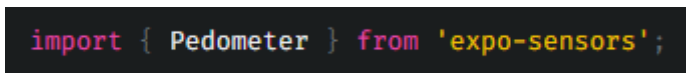
Opinnäytetyössä käytetään myös react-native-maps kirjaston ominaisuuksista Polyline-komponenttia, joka esittää kartalla matkatun reitin kartalle piirretyn viivan muodossa. Polyline sisällytetään sovellukseen samaan tapaan kuin MapView. Sille voidaan antaa parametreiksi esimerkiksi koordinaatit, viivan leveys ja väri (17).

Viimeinen tärkeistä Expo-kirjastoista on expo-sensors. Sen avulla kehittäjä voi luoda toiminnallisuksia mobiililaitteen erinäisten laiteantureiden kautta. Näihin antureihin lukeutuvat kiihtyvyyssanturi, barometri, liiketunnistin, gyroskooppi, magnetometri, valoanturi ja askelmittari. Kirjaston avulla sovellukset voivat mitata liikettä, suuntaa, painetta, magneettikenttiä, ympäristön valoa ja askelmäärää. Tähän opinnäytetyöhön käytetään hyväksi askelmittaria. Askelmittarin saa sisällytettyä projektiin ensin asentamalla expo-sensors-kirjaston (kuva 9). (18.)



KUVA 9. expo-sensors-kirjaston asennus (18).

Pedometer, askelmittari, on osa expo-sensors-kirjastoa, joka tarjoaa pääsyn laitteen askelmittari-anturiin (kuva 10). expo-sensors askelmittari käyttää järjestelmän laitteistoa. Se saa Androidissa käyttäjän askelmäärän anturilta ja mahdollistaa myös askelmittarin päivitysten tilaamisen (19).



KUVA 10. Askelmittarin sisällytys (19).

3 URHEILUSUORITUKSET-SOVELLUKSEN TOTEUTUS

3.1 Expo-projektin aloitus

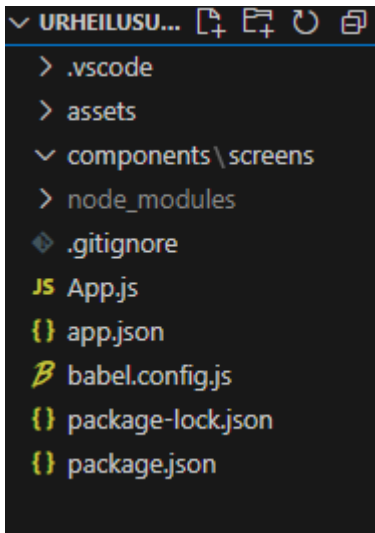
Sovellustoteutus aloitettiin asentamalla Node.js RTE tietokoneen juurihakemistoon, joka on yleisimmin levy C. Tämän voi tehdä joko komentokehoteella tai lataamalla node.js asennusohjelma Node.js sivustolta (20). Samalla asentuu npm, joka on paketin hallinta ja ohjelmistorekisteri, josta kehittäjät voivat löytää, rakentaa ja hallita koodipaketteja. Tällä hetkellä npm sisältää yli 800 000 pakettia erilaisille sovelluksille käyttöliittymästä ja robotiikasta mobiilisovelluksiin (21). Opinnäytetyötä varten Node asennettiin asennusohjelman avulla. Työssä käytettiin Visual Studio Code -ohjelmaa koodin muokkaukseen.

Node.js-asennuksen jälkeen projektin työstö voitiin aloittaa. Expo-projekti aloitetaan asentamalla se haluttuun paikkaan tietokoneella kehittäjän suosimalla komentokehoteella. Tässä tapauksessa luodaan projekti työpöydälle.

```
$ npx create-expo-app Urheilusooritukset --template blank
```

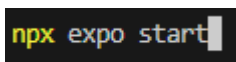
KUVA 11. Projektin luominen expo-ympäristöön

Kun asennus on valmis, on se luonut uuden kansion kehittäjän määrittelemällä nimellä, tässä tapauksessa Urheilusooritukset, joka sisältää Expo-projektin ilman ennalta-asetettuja sisältöjä. Projekti on tällä hetkellä vain malli, johon voidaan lisätä halutut toiminnallisuudet asteittain. Projektiin luotiin aluksi kansiot components ja sen sisälle screens. Tarkastelu aloitetaan avaamalla kansio Visual Studio Code -ohjelmalla (kuva 12).



KUVA 12. Projektin Urheilusuoritukset alkurakenne

Aloitetaan projektin suoritus Node.js-suoritusympäristössä. Projektin nykyisen tilan voi nähdä aloittamalla ympäristön suoritus Visual Studio Coden sisäisellä terminaalikomennolla Urheilusuoritukset-kansion sisältä (kuva 13).



KUVA 13. Expo-projektin suorituksen aloituskäsky

Aloitetaan projektin tarkastelu Expo Go -sovellusta käyttäen. Ensin täytyy varmistua, että kohdelaite ja tietokone käyttävät samaa WiFi-yhteyttä sovellustoiminnan takaamiseksi. Expo Go-sovelluksella huomataan, että aloitusnäky sisältää vain yhden tekstirivin, joka kehottaa tiedoston App.js muokkausta, johon on tällä hetkellä rakennettu käyttöliittymäksi tämä yksi tekstirivi viitekehyksen toimesta (kuva 14).

Open up App.js to start working on your app!

KUVA 14: Projektin aloitusnäky Expo go -sovelluksen näytöltä.

3.2 Näytöt ja navigointi

Projektia jatkettiin tiedoston App.js työstöllä. Tiedoston koodi määrittää Expo-perussovelluksen Firebase-todennuksen kanssa. Se kuuntelee tilanhallintamuutoksia, sallii käyttäjien kirjautua sisään tai rekisteröityä ja hallitsee navigointia sovelluksessa (kuva 14). AppNavigator-komponentti vastaa navigointilogiikan käsittelystä, joka ei ole tässä näkyvillä. Navigointilogiikka käydään läpi myöhemmin.

```
import React, { useState, useEffect } from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { getAuth, createUserWithEmailAndPassword, signInWithEmailAndPassword, onAuthStateChanged, signOut } from '@firebase/auth';
import AppNavigator from '../components/AppNavigator';
import {app} from '../components/FirebaseCfg'

export default function App() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [user, setUser] = useState(null);
  const [isLogin, setIsLogin] = useState(true);
  const auth = getAuth(app);

  useEffect(() => {
    const unsubscribe = onAuthStateChanged(auth, setUser);
    return () => unsubscribe();
  }, [auth]);

  const handleAuthentication = async () => {
    try {
      if (user) {
        // Jos käyttäjä on jo todennettu, kirjaudu ulos
        console.log('Kirjaututtu ulos');
        await signOut(auth);
      } else {
        // Kirjaudu sisään tai rekisteröidy
        if (isLogin) {
          // Kirjaudu sisään
          await signInWithEmailAndPassword(auth, email, password);
          console.log('Kirjaututtu sisään');
        } else {
          // Kirjaudu ulos
          await createUserWithEmailAndPassword(auth, email, password);
          console.log('Käyttäjää luotu');
        }
      }
    } catch (error) {
      console.error('Tapahtui virhe:', error.message);
    }
  };

  return (
    <NavigationContainer>
      <AppNavigator
        user={user}
        handleAuthentication={handleAuthentication}
        email={email}
        setEmail={setEmail}
        password={password}
        setPassword={setPassword}
        isLogin={isLogin}
        setIsLogin={setIsLogin}
      />
    </NavigationContainer>
  );
}
```

KUVA 14. App.js tiedoston koodi

App.js ei kuitenkaan käytännössä toimi, jos sitä ennen ei olla määritelty Firebasen konfiguraatio-tiedostoa. Tämä on tehty kansion components tiedostossa FirebaseCfg.js (kuva 15).

```
import { initializeApp } from 'firebase/app';
import { getFirestore } from 'firebase/firestore';

const firebaseConfig = {
  apiKey: process.env.API_KEY,
  authDomain: process.env.AUTH_DOMAIN,
  projectId: process.env.PROJECT_ID,
  storageBucket: process.env.STORAGE_BUCKET,
  messagingSenderId: process.env.MESSAGING_SENDER_ID,
  appId: process.env.APP_ID,
  measurementId: process.env.MEASUREMENT_ID
};

const app = initializeApp(firebaseConfig);
const db = getFirestore(app);
export {db, app};
```

KUVA 15. FirebaseCfg.js tiedoston koodi

Firestore.js määrittää Firebase-sovelluksen ja Firestore-tietokannan tarvittavat määrittämiset ja alustukset. Se tuo tarvittavat toiminnot Firebase SDK:sta, määrittää asetukset ympäristömuuttujien avulla, alustaa Firebase-sovelluksen, määrittää Firestoren ja vie nämä tiedot käytettäväksi muissa sovelluksen osissa. Tätä asetusta ei voi ohittaa Firebase-palvelujen, kuten todennus-, tietokanta- ja tallennusosien käyttöönotossa.

Projektiin tehtiin sitten kirjautumisnäyttö Authscreen.js, jossa määritellään kirjautumisen ulkonäkö käyttöliittymään (kuva 16). Se koostuu komponenteista, joihin lukeutuvat Image, TextInput, Button ja Text. Kirjautumisnäyttö vastaanottaa propseja tiedostosta App.js, kun sitä kutsutaan tiedoston AppNavigator.js sisältä. props on avainsana React-pohjaisissa viitekehyksissä, joilla viedään dataa yhdestä komponentista toiseen. Joitain propseja tiedostossa ovat email, password ja handleAuthentication.

```

import React from 'react';
import { View, Text, TextInput, Button, StyleSheet, Image, StatusBar } from 'react-native';
import Banner from '../Banner';

const AuthScreen = ({ email, setEmail, password, setPassword, isLogin, setIsLogin, handleAuthentication }) => {
  return (
    <View style={styles.authContainer}>
      <StatusBar/>
      <Banner appName="Urheilusuoritukset"/>
      <Image
        source={require('../../assets/wörkout.png')}
        style={[styles.image, {height: "35%", width: "100%"}]}
      />
      <TextInput
        style={styles.input}
        value={email}
        onChangeText={setEmail}
        placeholder="Sähköposti"
      />
      <TextInput
        style={styles.input}
        value={password}
        onChangeText={setPassword}
        placeholder="Salasana"
        secureTextEntry
      />
      <View style={styles.buttonContainer}>
        <Button title={isLogin ? 'Kirjaudu' : 'Rekisteröidy'} onPress={handleAuthentication} color="black" />
      </View>
      <View style={styles.bottomContainer}>
        <Text style={styles.toggleText} onPress={() => setIsLogin(!isLogin)}>
          {isLogin ? 'Tarvitsetko tilin? Rekisteröidy' : 'Tili jo olemassa? Kirjaudu sisään'}
        </Text>
      </View>
    </View>
  );
};

```

KUVA 16. AuthScreen.js tiedoston koodi

Käyttöliittymässä käyttäjä joko rekisteröityy luomalla uuden käyttäjätilin syöttämällä salasanan ja sähköpostin, tai hän kirjautuu jo olemassaolevaan tiliin samoilla tiedoilla, joilla hän aikaisemmin on rekisteröinyt käyttäjätilin. Käyttöliittymän lopullinen malli oli mustavalkoinen (kuva 17.)



KUVA 17. *AuthScreen.js* käyttöliittymä

Projektissa käytettiin navigoimiseen kirjastoa `react-navigation/native-stack`. Stack-navigoinnissa asetetaan aloitusnäyttö, josta matkataan muualle sovelluksen sisälle (22). Aloitusnäytöksi asetettiin `AuthScreen.js` tiedosto. Kirjasto sisällytettiin tiedostoon nimeltä `AppNavigator.js` (kuva 18), joka jaettiin `App.js` tiedostolle, joka käynnistää käyttöliittymän suorittamisen. Kuvassa näkyvät näytöt `HomeScreen`, `StepCounter`, `MapScreen` sekä `HistoryScreen`. Nämä näytöt sisällytettiin `AppNavigator.js`-tiedostoon `components`-kansion sisältä olevasta `screens`-kansioista.

```

import React from 'react';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import HomeScreen from './screens/HomeScreen';
import AuthScreen from './screens/AuthScreen';
import StepCounter from './screens/StepCounter';
import MapScreen from './screens/MapScreen';
import HistoryScreen from './screens/HistoryScreen';

const Stack = createNativeStackNavigator();
console.log("App Navigator active");
const AppNavigator = ({ user, handleAuthentication, email, setEmail, password, setPassword, isLogin, setIsLogin }) => {
  return (
    <Stack.Navigator>
      {user ? (
        <>
          <Stack.Screen name="Home" options={{ headerShown: false }}>
            {props => <HomeScreen {...props} user={user} handleLogout={handleAuthentication} />}
          </Stack.Screen>
          <Stack.Screen name="Askelmittari" component={StepCounter} options={{ headerShown: true }} />
          <Stack.Screen name="Lenkkiseuranta" component={MapScreen} options={{headerShown: true}}/>
          <Stack.Screen name='Lenkkihistoria' component={HistoryScreen} options={{headerShown: true}}/>
        </>
      ) : (
        <Stack.Screen name="Auth" options={{ headerShown: false }}>
          {() => (
            <AuthScreen
              email={email}
              setEmail={setEmail}
              password={password}
              setPassword={setPassword}
              isLogin={isLogin}
              setIsLogin={setIsLogin}
              handleAuthentication={handleAuthentication}
            />
          )}
        </Stack.Screen>
      )}
    </Stack.Navigator>
  );
};
export default AppNavigator;

```

KUVA 18. AppNavigator.js tiedoston koodi

Kirjautumisen jälkeen käyttäjä näkee HomeScreen.js käyttöliittymän. HomeScreen-komponentti toimii sisäänkirjautuneen käyttäjän kotisivuna, joka toivottaa hänet tervetulleeksi henkilökohtaisella viestillä ja tarjoaa navigoinnin sovelluksen eri ominaisuuksiin, kuten askelmittariin, lenkkiseurantaan ja lenkkihistoriaan. Noihin näkyymiin käyttäjä pääsee painamalla haluamansa näytön ikonia. Se sisältää myös uloskirjautumispainikkeen (kuva 19.)

Urheilusuoritukset

Tervetuloa testausta@gmail.com!



Askelmittari



Lenkkiseuranta



Lenkkihistoria



Kirjaudu ulos



KUVA 19. HomeScreen.js käyttöliittymä.

Tärkein näyttö projektissa on MapScreen.js, johon sisällytettiin projektin lenkkiseuranta. MapScreen-komponentti on suunniteltu seuraamaan käyttäjän kävelyä tai juoksua. Se sisältää useita kirjastoja ja palveluita. Näistä keskeisiä ovat karttatoiminnallisuus react-native-mapsin avulla, expo-location reaaliaikaiseen sijainnin seurantaan ja expo-sensors askelten laskemiseen askelmittarin avulla. Toiminnallisuus alkaa pyytämällä käyttäjältä sijaintilupia ja myönteisen luvan myötä sovellus saa suurimmalla mahdollisella tarkkuudella käyttäjän nykyisen sijainnin. Se määrittää tilamuuttujat, jotta sovellus voisi hallita reitin koordinaatteja, nykyistä sijaintia, kuljettua matkaa, ajastinta, seurannan tilaa, askelmäärää ja tilauksia paikka- ja askelmittaripäivityksiä varten.

Kun käyttäjä aloittaa seurannan, komponentti alkaa päivittää reitin koordinaatteja käyttäjän liikkeen perusteella, laskee kuljetun kokonaismatkan ja aloittaa sekuntikellon. Käyttäjän reitti piirretään kartalle Polyline-objektin avulla ja karttanäkymä seuraa käyttäjän nykyistä sijaintia. Komponentti ottaa myös kuvan viimeistellystä reitistä Polyline-viivan mukaisesti. Kun käyttäjä lopettaa

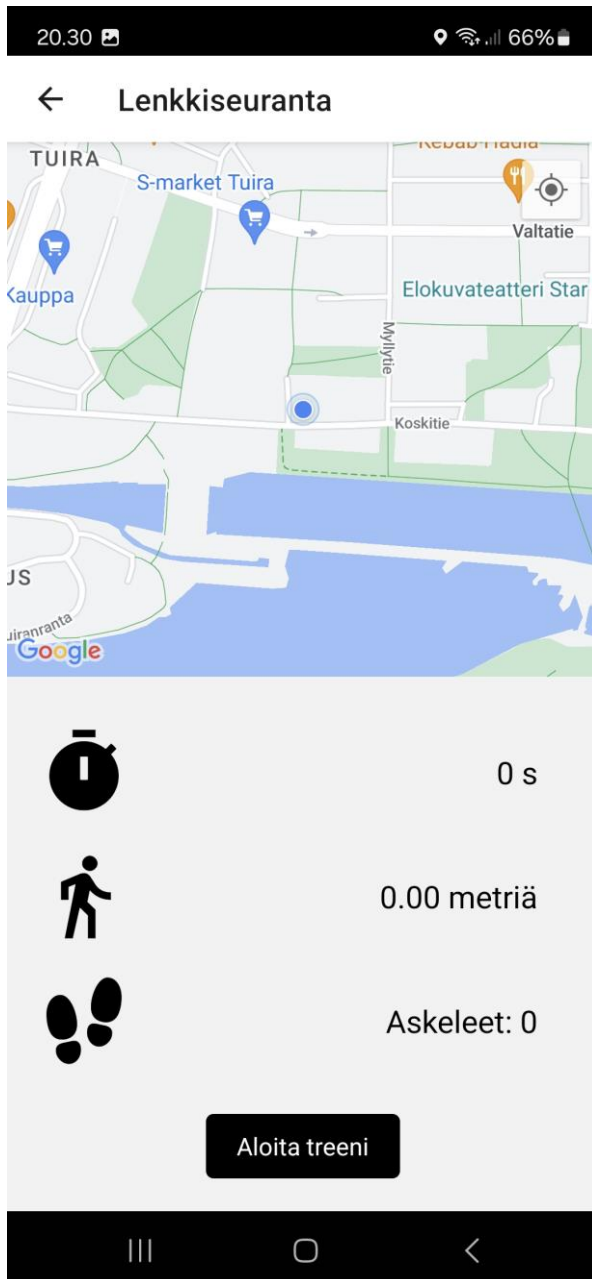
seurannan, ilmoitusikkuna kehottaa häntä vahvistamaan toimintansa tallentamisen. Jos hän vahvistaa tietojen tallennuksen, kartasta otetaan tilannekuva ja toimintatiedot, joihin lukeutuvat kuljettu matka, kokonaisaika ja askelmäärä. Nämä tiedot tallennetaan Firestoreen käyttäjän id-arvon kera.

MapScreen-komponentin käyttöliittymä sisältää karttanäkymän, joka näyttää käyttäjän reitin ja nykyisen sijainnin, jos hän on jo aloittanut lenkkiseurannan painamalla Aloita treeni-painiketta. Komponentti näyttää tietoruudut, jotka näyttävät kuluneen ajan, kuljetun matkan ja askelmäärän. Painikkeella käyttäjä voi aloittaa tai lopettaa seurantatoiminnon. Kuvassa 20 on käyttöliittymän koodi.

```
return (  
  <View style={styles.container}>  
    <MapView  
      ref={mapViewRef}  
      style={styles.map}  
      region={currentPosition}  
      showsUserLocation={true}  
      followsUserLocation={true}  
    >  
      {routeCoordinates.length > 0 && (  
        <Polyline  
          coordinates={routeCoordinates}  
          strokeWidth={3}  
          strokeColor='blue'  
        />  
      )}  
    </MapView>  
    <View style={styles.infoContainer}>  
      <View style={styles.infoBox}>  
        <MaterialIcons name="timer" size={70} color="black" />  
        <Text style={styles.infoText}>{formatTime(timer)}</Text>  
      </View>  
      <View style={styles.infoBox}>  
        <MaterialIcons name="directions-walk" size={70} color="black" />  
        <Text style={styles.infoText}>{distanceTravelled.toFixed(2)} metriä</Text>  
      </View>  
      <View style={styles.infoBox}>  
        <Ionicons name="footsteps" size={70} color="black" />  
        <Text style={styles.infoText}>{stepCount} askeleet</Text>  
      </View>  
      <TouchableOpacity onPress={isTracking ? stopTracking : startTracking} style={styles.button}>  
        <Text style={styles.buttonText}>{isTracking ? 'Lopeta treeni' : 'Aloita treeni'}</Text>  
      </TouchableOpacity>  
    </View>  
  </View>  
)  
);
```

KUVA 20. MapScreen.js tiedoston käyttöliittymän koodi

Käyttöliittymä sisältää mustavalkoisen värimaailman (kuva 21.) MapScreen.js sisältää 322 riviä koodia, joten sitä ei saa mahtumaan yhteen näyttökaappaukseen.



KUVA 21. Käyttöliittymä MapScreen.js tiedostosta

MapScreen.js tiedostossa löytyvät funktiot `calculateDistance` ja `calculateTotalDistance` luotiin koonaan ChatGPT-palvelun avulla. `calculateDistance` laskee etäisyyden kahden maantieteellisen pisteen välillä Haversinen kaavaa käyttäen, ja `calculateTotalDistance` laskee kokonaismatkan pituuden laskemalla kahden peräkkäisen pisteen välisen matkan vaihtuvien koordinaattien perusteella (kuva 22).

```

const calculateTotalDistance = (coordinates) => {
  let totalDistance = 0;
  for (let i = 0; i < coordinates.length - 1; i++) {
    totalDistance += calculateDistance(coordinates[i], coordinates[i + 1]);
  }
  totalDistance = Math.round(totalDistance * 100) / 100;
  setDistanceTravelled(totalDistance);
  console.log(`Total distance travelled: ${totalDistance} meters`);
};

const calculateDistance = (from, to) => {
  const rad = x => x * Math.PI / 180;
  const R = 6378137;
  const dLat = rad(to.latitude - from.latitude);
  const dLong = rad(to.longitude - from.longitude);
  const a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
    Math.cos(rad(from.latitude)) * Math.cos(rad(to.latitude)) *
    Math.sin(dLong / 2) * Math.sin(dLong / 2);
  const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
  const distance = R * c;
  return distance;
};

```

KUVA 22. funktiot *calculateTotalDistance* ja *calculateDistance* koodissa

Lenkkiseurannan tulokset ovat nähtävissä HistoryScreen.js tiedoston (kuva 23) avulla. HistoryScreen-komponentti on suunniteltu näyttämään käyttäjän lenkkihistoria. Se hakee toimintatiedot Firebase Firestore-tietokannasta ja näyttää ne vieritettävänä kortteina, jotka voi poistaa. Koodi varmistaa sen, että tämänhetkinen kirjautuja näkee vain ne tiedot, jotka hänen käyttäjätilinsä on luonut.

```

const HistoryScreen = () => {
  const [activities, setActivities] = useState([]);
  const auth = getAuth(); // Get the current user
  const userId = auth.currentUser?.uid; // Get the user's ID

  useEffect(() => {
    const fetchActivities = async () => {
      try {
        const q = query(collection(db, 'activities'), where('userId', '==', userId));
        const querySnapshot = await getDocs(q);
        const activitiesData = querySnapshot.docs.map(doc => ({
          id: doc.id,
          ...doc.data()
        }));
        console.log("Fetched activities:", activitiesData);
        setActivities(activitiesData);
      } catch (error) {
        console.error("Failed to fetch activities:", error);
      }
    };

    if (userId) {
      fetchActivities();
    }
  }, [userId]);

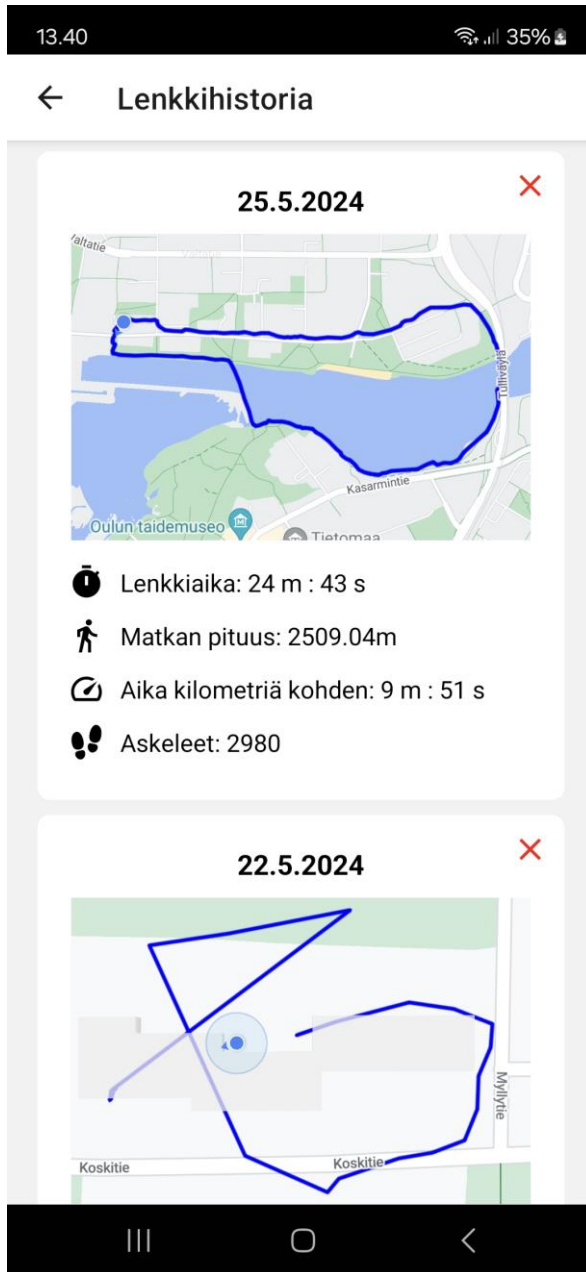
  const calculatePace = (seconds, distance) => {
    if (distance <= 0) {
      return "Ei määritely";
    }
    const pacePerKmInSeconds = seconds / (distance / 1000); // Time in seconds per km
    const minutes = Math.floor(pacePerKmInSeconds / 60);
    const secondsRemainder = Math.floor(pacePerKmInSeconds % 60);
    return `${minutes} m : ${secondsRemainder < 10 ? '0' : ''}${secondsRemainder} s`;
  };

  const handleDelete = (activityId) => {
    Alert.alert(
      "Poista treeni",
      "Haluatko varmasti poistaa tämän treenin?",
      [
        {
          text: "Ei",
          style: "cancel"
        },
        {
          text: "Kyllä",
          onPress: async () => {
            try {
              // Delete from Firestore
              await deleteDoc(doc(db, 'activities', activityId));
              // Delete from frontend
              setActivities(activities.filter(activity => activity.id !== activityId));
              console.log("Activity deleted successfully");
            } catch (error) {
              console.error("Failed to delete activity:", error);
            }
          },
          style: "destructive"
        }
      ],
      { cancelable: false }
    );
  };
};

```

KUVA 23. HistoryScreen.js tiedoston logiikka

Suoritushistorian käyttöliittymä on yksinkertainen ja mustavalkoinen (kuva 24). Historia näyttää päivämäärän, kuljetun reitin kartalla, kokonaisajan, kuljetun matkan, ajan kilometriä kohden ja askelmäärän.



KUVA 24. *Historyscreen.js* käyttöliittymä

4 POHDINTA

Urheilusuoritukset-sovellus pääsi osin alussa asetettuihin tavoitteisiin, muttei tuntunut täysin onnistuneelta projektilta. Projektissa oli alun perin tarkoitus julkistaa sovellus Play-kauppaan, mutta aikarajoitteiden vuoksi en saanut tuota tehtyä. Testaussovellukseksi jäänyt Urheilusuoritukset-sovellus oli mielestäni hyvin tehty, ja toiminnallisuudet olivat kaikki tarpeellisia hyvään lopputulokseen. Mielestäni pääsin sovelluksen osalta tyydyttävään lopputulokseen, sillä lopputuloksena oli käytäjäystävällinen sovellus, jonka keskeisin tehtävä on seurata lenkkipolkua GPS-paikannuksella. Reittikartan laatiminen, reitin ajastus, ajan laskeminen kilometriä kohden ja askeleet treenin aikana päätyivät lopullisiksi ominaisuuksiksi. Tietysti noiden asioiden kertaaminen lenkkihistorianäytön avulla päätyi myös viimeistelyyn projektiin.

Opinnäytetyön aikana opin lisää JavaScriptistä ja varmistin aikaisemmin oppimiani taitoja versionhallinnasta ja GitHub-palvelusta. JavaScriptin useState-hook jäi mieleen pysyvästi.

Opinnäytetyön kirjoittamiseen olisi tarvittu enemmän keskittymistä. Kirjoitukseen ei välttämättä käytetty tarpeeksi aikaa, ja pahimmillaan aikaa käytettiin asioiden murehtimiseen niiden tekemisen sijaan. Opinnäyte ei välttämättä saanut lisää kirjoituspuolen täytettä päiviin, vaan sovellusta tehtiin vajavaisilla taidoilla. Koin olevani hyvin hankalissa tilanteissa usein, ja tarvitsin usein apua ChatGPT-palvelulta koodaustietojeni varmistukseen. Monesti meni monta päivää, että sain kirjoitettua koodia joka oli merkityksellistä opinnäytetyön etenemiselle.

React Expon parissa oli helppo työskennellä. Expo Go -sovellus mahdollisti sen, että kehittämisen aloittamiseen vaativan Wifi-yhteyden alueelta pystyi poistua kokeilemaan GPS-paikannusta muualle. Sovellus oli ylipäätään ensisijaisen tärkeä, koska en saa jostain syystä kotikoneeni emulaattoria toimimaan. Ilman Expo Go-sovelluksen reaaliaikaisia päivityksiä en olisi varmaan saanut mitään tehtyä projektin aikana.

Opinnäytetyön tekeminen ei kuitenkaan ollut epämiellyttävä kokemus. Joka kerran, kun sain jotain toimimaan, tuntui mahtavalta. Kun sain ensimmäisen kerran karttaseurannan toimintaan, tuuletin hieman ja huomasin osaavani uusia asioita. Samalla, kun aloitin kirjoittamaan enemmän, tunsin, että tämä ei ole ylitsepääsemätön este.

Koin ongelmalliseksi opinnäytetyötä aloittaessa sen, että en löytänyt opinnäytetyön aihetta yrityksestä. Yrityksen toimeksiannosta olisin tehnyt jonkinlaisen työn, ja olisin keskittynyt vain lopputuloksen luomiseen. Ongelmalliseksi koin siis tarpeellisten ja tärkeiden aiheiden löytämisen, joista tekisin opinnäytetyöni. Lopulta päädyin aiheeseen, joka auttoi minua itseäni urheilemaan ja ulkoilemaan säännöllisemmin.

Pois jääneitä ominaisuuksia sovelluksesta olivat kuvaajan luominen, jossa näkyisi viikoittaiset suoritukset verrattuna omaan keskiarvoon, ja käyttäjätietojen laajentaminen kirjautumisesta omaan profiiliin, jossa olisi voinut määrittää omia urheilutottumuksia. Urheilusuorituksen aikana kartta ei myöskään automaattisesti seuraa kännykkää sen antaessa päivityksiä, vaan näkymä pitää päivittää itse, että kartta seuraa senhetkistä sijaintia. Nämä voivat olla jatkokehityksiä. Etäisyys- ja reitin seuranta kartalla viivan muodossa kuitenkin onnistuvat automaattisesti suorituksen aikana. Sovellus on helposti jatkokehitettävissä, ja enempiä toiminnallisuuksia voi keksiä tarpeiden tullen.

LÄHTEET

1. Roche David 2022. Trailrunner. Why Consistency, Not Intensity, Is the Key to Running Success. Trailrunner. Hakupäivä 7.5.2024. <https://www.trailrunnermag.com/training/trail-tips-training/consistency-not-intensity-key-running-success/>.
2. Mobulous Technologies 2023. How to Upload App to the Apple Store: A Step-by-Step Guide. LinkedIn. Hakupäivä 7.5.2024. <https://www.linkedin.com/pulse/how-upload-app-apple-store-step-by-step-guide-mobulous-elrif/>.
3. Codecademy Team 2021. What Is a Framework? Hakupäivä 23.5.2024. <https://www.codecademy.com/resources/blog/what-is-a-framework/>.
4. Panagia Sergio & Alonzi Giacomo 2022. Moze Studio. What is Expo and why it matters for app development. Moze. Hakupäivä 7.5.2024. <https://www.mozestudio.com/journal/what-is-expo-and-why-it-matters-for-app-development/>.
5. Netguru 2024. What is React Native? Complex Guide for 2024. Hakupäivä 22.5.2024. <https://www.netguru.com/glossary/react-native>.
6. Codecademy Team 2024. Introduction to JavaScript Runtime Environments. Codecademy. Hakupäivä 7.5.2024. <https://www.codecademy.com/article/introduction-to-javascript-runtime-environments>.
7. Simplilearn 2024. 10 Practical Applications of JavaScript & Tips for a Successful Career. Hakupäivä 22.5.2024. <https://www.simplilearn.com/applications-of-javascript-article>.
8. Expo 2024. Core concepts. Feature showcase. Hakupäivä 7.5.2024. <https://docs.expo.dev/core-concepts/>.
9. Expo 2024. Reference. Hakupäivä 22.5.2024. <https://docs.expo.dev/versions/latest/>.

10. Fanchi Christopher 2023. Backendless. What is Mobile Backend as a Service (MBaaS)? Backendless. Hakupäivä 7.5.2024. <https://backendless.com/what-is-mobile-backend-as-a-service-mbaas/>.
11. taglinej 2023. Medium. A comprehensive Guide to Backend Options for React Native Development. Medium. Hakupäivä 7.5.2024. <https://medium.com/@jinalg.tagline/a-comprehensive-guide-to-backend-options-for-react-native-development-8d3fb1e2da5c>.
12. DB-Engines 2024. DB-Engines Ranking. Hakupäivä 7.5.2024. <https://db-engines.com/en/ranking>.
13. Lee Wei-Meng 2024. Code Magazine. Introduction to Cloud Firestore. Hakupäivä 22.5.2024. <https://www.codemag.com/Article/1905071/Introduction-to-Cloud-Firestore>.
14. Expo 2024. Use Firebase. Hakupäivä 7.5.2024. <https://docs.expo.dev/guides/using-firebase/>.
15. Expo 2024. Expo Location. Hakupäivä 23.5.2024. <https://docs.expo.dev/versions/latest/sdk/location/>.
16. Expo 2024. MapView. Hakupäivä 23.5.2024. <https://docs.expo.dev/versions/latest/sdk/map-view/#installation>.
17. react-native-maps Github 2024. react-native-maps. Hakupäivä 23.5.2024. <https://github.com/react-native-maps/react-native-maps?tab=readme-ov-file>.
18. Expo 2024. Expo Sensors. Hakupäivä 23.5.2024. <https://docs.expo.dev/versions/latest/sdk/sensors/>.
19. Expo 2024. Pedometer. Hakupäivä 23.5.2024. <https://docs.expo.dev/versions/latest/sdk/pedometer/>.
20. Powell Zachid 2024. Kinsta. How to install Node.js and npm on Windows, macOS, and Linux. Hakupäivä 23.5.2024. <https://kinsta.com/blog/how-to-install-node-js/>.

21. Metwalli Sara 2024. BuiltIn. What Is NPM? Hakupäivä 23.5.2024. <https://builtin.com/software-engineering-perspectives/npm>.

22. React Navigation 2024. Native Stack Navigator. Hakupäivä 23.5.2024. <https://reactnavigation.org/docs/native-stack-navigator/>.