OULUN AMMATTIKORKEAKOULU

Miikka Kortelainen

JATKUVAN PAIKANNUKSEN TOTEUTTAMINEN MAXTECH PRO -MOBIILISOVELLUKSEEN

JATKUVAN PAIKANNUKSEN TOTEUTTAMINEN MAXTECH PRO -MOBIILISOVELLUKSEEN

Miikka Kortelainen
Opinnäytetyö
Syksy 2021
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Miikka Kortelainen
Opinnäytetyön nimi: Jatkuvan paikannuksen toteuttaminen Maxtech Pro -mobiilisovellukseen
Opinnäytetyön englanninkielinen nimi: Implementing continuous location tracking to Maxtech Pro mobile application
Työn ohjaajat: Lasse Haverinen, Jenna Moisejeff
Työn valmistumislukukausi ja -vuosi: Syksy 2021
Sivumäärä: 41 + 1 liite

Opinnäytetyössä toteutettiin Max Technologies Oy:n toimeksianto. Työn aiheena oli jatkuvan paikannuksen toteuttaminen Maxtech Pro -mobiilisovellukseen. Tehtävä oli toteuttaa ja suunnitella paikannustoiminnallisuus, jolla laitteen sijaintia voidaan tarkkailla myös sovelluksen ollessa taustalla. Laitteen sijaintitietojen perusteella sovelluksen tuli muodostaa ajotapahtumia automaattisesti.

Työssä sovellukseen kehitettiin Android-alustalle natiivi palvelu laitteen sijainnin jatkuvalle paikanukselle ja alustariippumaton sijaintitietojen ja ajotapahtumien käsittelijä. Lisäksi sovelluksen karttanäkymän käyttöliittymää laajennettiin tukemaan ajotapahtumiin liittyviä toimintoja.

Työn tuloksena mobiilisovellukseen syntyi jatkuvan paikannuksen ominaisuus, joka tarkkailee laitteen sijaintia myös sovelluksen ollessa taustalla ja muodostaa ajotapahtumia automaattisesti.

Asiasanat: Mobiilisovellus, Android, ohjelmistokehitys

ABSTRACT

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

---

Author: Miikka Kortelainen
Title of thesis: Implementing continuous location tracking to Maxtech Pro mobile application
Supervisors: Lasse Haverinen, Jenna Moisejeff
Term and year when the thesis was submitted: Autumn 2021
Number of pages: 41 + 1 appendix

The purpose of this thesis was to design and implement background location tracking feature to Maxtech Pro mobile application. The feature was meant to be used to automatically construct driving sessions from the received location data.

In the thesis a native Foreground Service for tracking the device location was developed for Android platform version of the application and a cross-platform solution was developed for constructing driving sessions based on the received location data. The applications map screen was expanded to support driving session related functionalities.

The result was a functional feature for the mobile application, which tracks the devices' location continuously even when the application is in the background and constructs driving sessions automatically.

Keywords: Mobile application, Android, software development, React, React Native

SISÄLLYS

1 JOHDANTO

Työn tilaajana toimi Maxtech (Max Technologies Oy). Maxtech on suomalainen työnhallinta- ja paikannuspalveluita tarjoava yritys. Opinnäytetyön aiheena oli jatkuvan paikannuksen toteuttaminen Maxtech Pro -mobiilisovellukseen. Ominaisuutta on tarkoitus käyttää ajoneuvopaikannuksessa.

Opinnäytetyön tehtävänä oli suunnitella ja toteuttaa Maxtech Pro -mobiilisovellukseen laitteen paikannusominaisuus, joka toimisi luotettavasti myös sovelluksen ollessa taustalla. Paikannuksesta saatavan sijaintitietojen perusteella mobiilisovelluksen tuli muodostaa ajotapahtumia automaattisesti sekä synkronoida niiden tietoja web-järjestelmään rajapinnan kautta.

Tehtävä oli myös laajentaa sovelluksen karttanäkymän käyttöliittymää muodostettujen ajotapahtumien tarkkailua ja hallinnoimista varten. Karttanäkymään tuli lisätä valintakomponentti ajotapahtumissa käytettävälle ajoluokalle ja komponentti, josta ilmenee käynnissä olevan tapahtuman tiedot kuten, esimerkiksi kuljettu matka, kesto ja muodostunut kilometrikorvaus. Karttanäkymässä tuli näyttää käyttäjän päivän aikana ajamat reitit sekä käynnissä olevan ajotapahtuman reitti.

Maxtech Pro -mobiilisovelluksen kehitys aloitettiin keväällä 2019. Sovelluksen tarkoituksena on ajan saatossa korvata yrityksessä aikaisemmin kehityt Reporting Tool v3- ja TrackMyFleet-sovellukset. TrackMyFleet on vuosina 2011–2013 ajoneuvopaikannukseen kehitetty mobiilisovellus, jossa ajoneuvon reittiä seurataan karttanäkymästä ja annetaan käyttäjälle mahdollisuus raportoida työn etenemistä kentältä. Aikaisemmat sovellukset on kehitetty erikseen Android- ja iOS-alustalle. Maxtech Pro on puolestaan monialustainen React Native -viitekehyksellä toteutettu mobiilisovellus, jonka jatkokehitys on tehokkaampaa.

Tarve työlle tuli yrityksen suunnalta. TrackMyFleet -mobiilisovelluksen tärkeimmät ominaisuudet kuten, esimerkiksi jatkuva paikannus ja työn etenemisen raportointi on tarkoitus kehittää Maxtech Pro -mobiilisovellukseen, jotta asiakkuuksia voidaan siirtää uuden mobiilisovelluksen piiriin. Työn tulokset ovat oleellinen osa tätä projektia.

2 REACT

Tässä luvussa käydään lyhyesti läpi React-kirjaston oleellisimmat asiat, koska raportissa myöhemmin käsitelty React Native -viitekehys pohjautuu siihen. React on Facebookin ylläpitämä ja kehittämä avoimen lähdekoodin JavaScript-kirjasto, jolla voidaan luoda interaktiivisia, HTML-pohjaisia käyttöliittymiä JavaScriptillä (1).

Reactilla toteutetut käyttöliittymät koostuvat komponenteista (components). Komponentit ovat pohjimmiltaan JavaScript-funktioita, jotka palauttavat käyttöliittymän sisällön HTML-muodossa. Komponenteilla voi olla oma tila (state) ja niille voidaan antaa tietoa ylemmiltä komponenteilta (props). (Kuva 1.) (2; 3.)

```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

KUVA 1. Reactilla määritelty käyttöliittymäkomponentti (3)

Sovelluksen tietojen muuttuessa käyttöliittymästä päivitetään ainoastaan ne komponentit, joiden sisältö on muuttunut, ilman että koko käyttöliittymää tarvitsee päivittää. Komponentti päivitetään, jos siihen liittyvä tieto (state, props) muuttuu tai sen yläkomponentti (parent component) päivittyy. (4.)

Komponentit voidaan määritellä JSX-syntaksilla (JavaScript XML). JSX on JavaScriptin syntaksilaajennus, joka mahdollistaa HTML-elementtien käyttämisen JavaScript-koodissa, mikä nopeuttaa

käyttöliittymien luomista (kuva 2). Pohjimmiltaan JSX on kuitenkin vain nk. syntaktista sokeria (syntatic sugar), joka kääntyy normaaliksi JavaScriptiksi sovelluksen suorituksen aikana (kuva 3). (5.)

```
const jsxElement = <h1>This is a JSX element</h1>;

ReactDOM.render(jsxElement, document.getElementById('root'));
```

KUVA 2. Reactissa HTML-elementin luonti JSX:llä (5)

```
const element = React.createElement('h1', {}, 'This is not JSX!');

ReactDOM.render(element, document.getElementById('root'));
```

KUVA 3. Reactissa HTML-elementin luominen JavaScriptillä ilman JSX-syntaksia (5)

3 REACT NATIVE

3.1 Historia

React Native on Facebookin kehittämä avoimen lähdekoodin viitekehys sovelluskehitykseen, joka julkaistiin suurelle yleisölle vuona 2015. React Native kehitettiin aluksi Facebookin sisäisenä projektina ratkaisemaan mobiilisovelluskehityksessä ilmenneitä ongelmia, kun Facebook julistautui vuonna 2012 mobiilisovelluskehityspainotteiseksi yritykseksi. HTML-tekniikalla toteutetut mobiilisovellukset todettiin ongelmalliseksi niiden heikon suorituskyvyn vuoksi. (6.)

React Native sai alkunsa vuonna 2012, kun Facebookilla työskentelevä sovelluskehittäjä Jordan Walke kehitti prototyypin, jolla hän onnistui luomaan iOS-alustalla natiiveja käyttöliittymäelementtejä määriteltynä JavaScript-kielellä. Prototyypin jatkokehittämiseksi järjestettiin Facebookin sisäinen hackathon-tapahtuma. Tapahtuman lopputuloksena syntyi ensimmäinen versio React Nativesta, jonka jatkokehitykselle perustettiin yrityksen sisäinen tiimi. (6.)

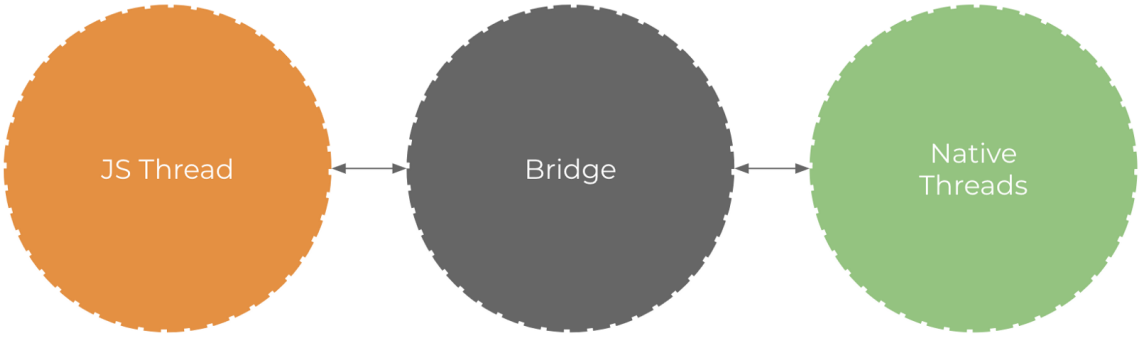
React Native oli alun perin kehitetty pelkästään mobiilisovelluskehitykseen, mutta nykyään sillä on mahdollista kehittää sovelluksia myös virtuaalitodellisuus- (virtual reality) ja työpöytäympäristöön. Viitekehyksen tulevaisuuden visio onkin, että sillä voisi kehittää sovelluksia mahdollisimman monelle alustalle. (7.)

3.2 Sovelluksen rakenne

React Native pohjautuu React-kirjastoon. Komponenttien päivityslogiikka ja muuttujat (props, state) toimivat samalla tavalla React Nativessa kuin Reactissa. React Native poikkeaa Reactista siten, ettei JSX:llä määriteltäviä käyttöliittymäkomponentteja muunneta HTML-elementeiksi, vaan alustan natiiveiksi käyttöliittymäkomponenteiksi (8).

React Native -sovellus koostuu natiiviosuudesta ja alustariippumattomasta JavaScript-osuudesta (8). Natiiviosuus on alustasta riippuen toteutettu eri ohjelmointikielellä, esimerkiksi iOS-alustalla Objective-C-kielellä tai Android-alustalla Java-kielellä.

Sovelluksen osuudet kommunikoivat keskenään niiden välille muodostetun sillan välityksellä, jota voisi luonnehtia eräänlaisena viestinvälittäjänä. Sillan kautta lähetettävät viestit ovat asynkronisia ja JSON (JavaScript Object Notation) -muodossa. JavaScript-säikeessä (JS Thread) JSX:llä määritelty käyttöliittymä lähetetään sillan kautta natiiville käyttöliittymäsäikeelle (UI Thread), jossa sovelluksen käyttöliittymä muodostetaan natiiveista käyttöliittymäkomponenteista. (Kuva 4.) (8.)



KUVA 4. Sovelluksen osien kommunikointi sillan (bridge) välityksellä (8)

React Nativessa on jo valmis tuki keskeisimmille käyttöliittymäkomponenteille Android- ja iOS-alustoille. Näitä komponentteja kutsutaan ydinkomponenteiksi (core components) (taulukko 1). Sovelluskehittäjät voivat myös tarpeen tullen tehdä omia natiiveja komponentteja (native components) tai käyttää valmiita komponenttikirjastoja. (9.)

TAULUKKO 1. Osa React Nativen tukemista ydinkomponenteista (9)

React Native	**Android**	**iOS**	**Kuvaus**
<View>	<ViewGroup>	<UIView>	Alue (container), tukee flexbox-tyylitelyä, kosketus- ja saatavuustoimintoja.
<Text>	<TextView>	<UITextView>	Näyttää tekstiä, tukee tyylittelyä ja kosketustoimintoja.
<Image>	<ImageView>	<UIImageView>	Näyttää erityyppisiä kuvia.
<ScrollView>	<ScrollView>	<UIScrollView>	Geneerinen vieritettävä alue, joka voi sisältää useita komponentteja ja alueita.
<TextInput>	<EditText>	<UITextField>	Sallii käyttäjän syöttää tekstiä.

3.3 Natiivit moduulit

Natiivit moduulit kytkeytyvät React Native -sovelluksien natiiviin osuuteen ja niitä voidaan käyttää sovelluksen alustariippumattomasta JavaScript-osuudesta. Tyypillinen käyttötarkoitus natiiville moduulille on paljon laskentatehoa vaativan ominaisuuden toteuttaminen tehokkaana monisäikeisenä koodina tai alustakohtaisen rajapinnan tuonti käytettäväksi sovelluksen JavaScript-osuuteen (esim. Googlen maksupalvelut). Sovelluskehittäjän kuitenkin harvemmin tarvitsee itse kehittää natiivia moduulia, sillä monessa tapauksessa vastaavaan käyttötarkoitukseen on jo tehty valmis moduuli, jonka kehittäjä voi ladata npm-pakettina. (10.)

4 TYÖN TAUSTA

4.1 Esiselvitys

Ennen työn käynnistämistä tehtiin esiselvitys, jossa tutkittiin, onko taustapaikannusta mahdollista toteuttaa tarpeeksi luotettavasti mobiilisovellukseen. Esiselvityksessä pyrittiin kartoittamaan mahdollisia ongelmia toteutuksessa ja miten niitä voitaisiin välttää. Tuloksena syntyi esiselvitysdokumentti (liite 1), jonka perusteella työ päätettiin toteuttaa.

Keskeisin aihe esiselvityksessä oli laitteen virransäästöasetusten vaikutus sijainnin hakemiseen taustalla ja GPS-pisteiden tarkkuuteen. Koska laitteiden virransäästöasetukset vaihtelevat paljon eri laitevalmistajien kesken, päätettiin, että ominaisuuden ensimmäistä versiota suositellaan käyttötapauksiin, joissa laite on jatkuvassa virransyötössä. Jatkuvalla virransyötöllä voidaan ennaltaehkäistä virransäästöasetuksista aiheutuvia ongelmia sijainnin hakemisessa taustalla suurella kirjolla eri laitteita. Tehtyjen rajauksien myötä suunnittelu- ja kehitystyön käynnistäminen oli helpompaa.

4.2 Vaatimusten määrittely

Vaatimusten määrittely tehtiin tilaajan kanssa ennen työn aloittamista. Ominaisuus pyrittiin suunnittelemaan asiakkaan näkökulmasta eli siten, että sen käyttäminen ei vaatisi ylimääräisiä toimenpiteitä käyttäjältä. Keskeisin vaatimus oli, että paikannuksen tulee toimia luotettavasti myös silloin, kun laitteen näyttö on kiinni ja sovellus on taustalla. Paikannuksesta saatavan sijaintitiedon tulee myös olla tarkkaa, jotta siitä muodostettavien ajotapahtumien reitti on todenmukainen. Ominaisuuden tulee toimia kaikilla sovelluksen tuetuilla alustoilla eli Android- ja iOS-alustalla.

Ominaisuuden käyttöönottoa varten sovelluksen käyttöliittymään tulee lisätä aktivointipainike. Aktivointipainike käynnistää paikannuksen, jos käyttäjä on kirjautuneena ajoneuvon kuljettajaksi ja sovellukselle on annettu luvat sijainnin käyttämiselle. Virhetilanteissa käyttäjälle tulee indikoida selvästi, miksi taustapaikannuksen käynnistäminen ei onnistunut.

Paikannuksesta saatavaa sijaintidataa tulee hyödyntää ajotapahtumien muodostamisessa. Sijainnin muutoksen perusteella sovelluksen tulee pystyä aloittamaan ja lopettamaan ajotapahtumia automaattisesti. Ajotapahtumien dataa tulee synkronoida web-järjestelmän rajapintaan säännöllisesti. Synkronointia varten tulee web-järjestelmän rajapintaa laajentaa tukemaan ajotapahtumien sijaintitietojen vastaanottamista.

Sovelluksen karttanäkymän käyttöliittymään tulee lisätä toiminnallisuuksia tukemaan taustapaikannusta. Karttanäkymässä tulee näyttää käynnissä olevan ajotapahtuman tiedot. Tapahtuman tiedoista tulee ilmetä kuljettu matka, matkan kesto ja ajoluokan perusteella muodostunut kilometrikorvaus. Päivän aikana ajetut reitit tulee piirtää karttanäkymään. Reitin väri määräyty ajoluokalle web-järjestelmässä asetetun värin mukaan.

Ajotapahtumissa käytettävää ajoluokkaa tulee pystyä vaihtamaan helposti karttanäkymästä. Valittavissa tulee olla kaikki ajoneuvolle web-järjestelmässä asetetut ajoluokat. Ajoluokan vaihtaminen aloittaa uuden ajotapahtuman, jolle valitun ajoluokan perusteella lasketaan kilometrikorvaus.

5 TOTEUTUS

Kehitystyössä pyrittiin käyttämään mahdollisimman vähän alustariippuvaista koodia, jotta samaa koodia voidaan hyödyntää kaikilla sovelluksen tuetuilla alustoilla. Android-alustalle kehitettiin Java-kielellä natiivi palvelu laitteen sijainnin hakemiselle. Palvelu lähettää laitteen sijaintia sovelluksen alustariippumattomalle JavaScript-osuudelle, jossa varsinainen sijaintitietojen käsittely tapahtuu.

Sovellukseen alustariippumattomaan JavaScript-osuuteen toteutettiin LocationHandler-luokka, joka käsittelee natiiveilta palveluilta saatavan laitteen sijainnin. Luokka aloittaa ja lopettaa ajotapahtumia automaattisesti vastaanotetun sijaintitietojen perusteella sekä synkronoi ajotapahtumat mobiilisovelluksesta web-järjestelmään. Web-järjestelmän rajapintaa laajennettiin tukemaan ajotapahtumien sijaintietojen lisäystä web-järjestelmän tietokantaan.

Ominaisuuden keskiössä on sovelluksen karttanäkymä, jonka käyttöliittymää laajennettiin toteutuksen yhteydessä hyödyntämään muodostettuja ajotapahtumia. Karttanäkymään lisättiin komponentti käynnissä olevan ajotapahtuman tietojen näyttämiselle ja komponentti, josta käyttäjä voi vaihtaa ajotapahtumissa käytettävää ajoluokkaa. Valitun ajoluokan perusteella määritellään ajotapahtuman ominaisuuksia, esimerkiksi tallennetaanko sen reitti web-järjestelmään ja paljonko kilometrikorvausta ajolle muodostuu.

Karttanäkymään lisättiin myös toiminnallisuus, josta käyttäjä näkee päivän aikana ajamansa reitit piirrettynä kartalle. Käynnissä olevan ajotapahtuman reittiä päivitetään sovelluksen karttanäkymään sen edetessä.

5.1 Natiivi palvelu laitteen sijainnin tarkkailulle Android-alustalla

Työssä selvitettiin mahdollisuutta käyttää jotain valmista kirjastoa laitteen sijainnin hakemiseen sovelluksen ollessa taustalla. Potentiaalisesti sopivia kirjastoja oli päivitetty viimeksi useita vuosia sitten eivätkä ne täysin soveltuneet suunniteltuun kokonaisuuteen. Parhaaksi ratkaisuksi koettiin toteuttaa itse tarvittavat natiivit palvelut sijainnin hakemiselle. Tämän päätöksen myötä myös sovelluksen jatkokehitys tulee todennäköisesti olemaan suoraviivaisempaa.

Työssä toteutettiin pelkästään tuki sijainnin hakemiselle Android-alustalla. Työn aikataulua arvioitiin tilaajan kanssa kesken toteutuksen ja iOS-alustan tuki päätettiin siirtää osaksi jatkokehitystä. Toteutus oli välissä tauolla kiireellisempien työtehtävien vuoksi, joka tiukensi opinnäytetyön aikataulua. Tilaajan kanssa arvioitiin, ettei iOS-alustan tukea ole mahdollista toteuttaa työn tavoiteajankohtaan mennessä. Päätöksen hetkellä kehitystyö oli muilta osin loppusuoralla, joten parhaaksi ratkaisuksi katsottiin siirtää iOS-alustan tuki osaksi jatkokehitystä ja saattaa opinnäytetyö valmiiksi.

Android-alustan taustalla olevat sovellukset voivat pyytää laitteen sijaintia käyttöjärjestelmältä vain muutaman kerran tunnissa. Rajoitus voidaan kiertää luomalla Foreground Service -tyyppinen palvelu, joka voi tarkkailla käyttäjän sijaintia aktiivisesti myös sovelluksen ollessa taustalla. Foreground Service -palveluiden on kuitenkin pakko näyttää käyttäjälle ilmoitus järjestelmän ilmoituspalkissa silloin, kun palvelu on käynnissä taustalla, jotta käyttäjä olisi tietoinen palvelun suorittamisesta ja sen kuluttamista resursseista. (11.)

Android-alustalle toteutettiin Java-ohjelmointikielellä AndroidLocationService-palvelu sijainnin hakemiselle, joka on tyypiltään Foreground Service. Palvelun ohjausta varten toteutettiin natiivi AndroidLocationServiceModule-moduuli, jolla sovelluksen alustariippumaton osuus voi kommunikoida palvelun kanssa komentojen välityksellä. Palvelu pyrittiin toteuttamaan siten, että sitä voidaan mukauttaa komentojen välityksellä sopivaksi erilaisiin käyttötarkoituksiin sovelluksen alustariippumattomasta JavaScript-osuudesta.

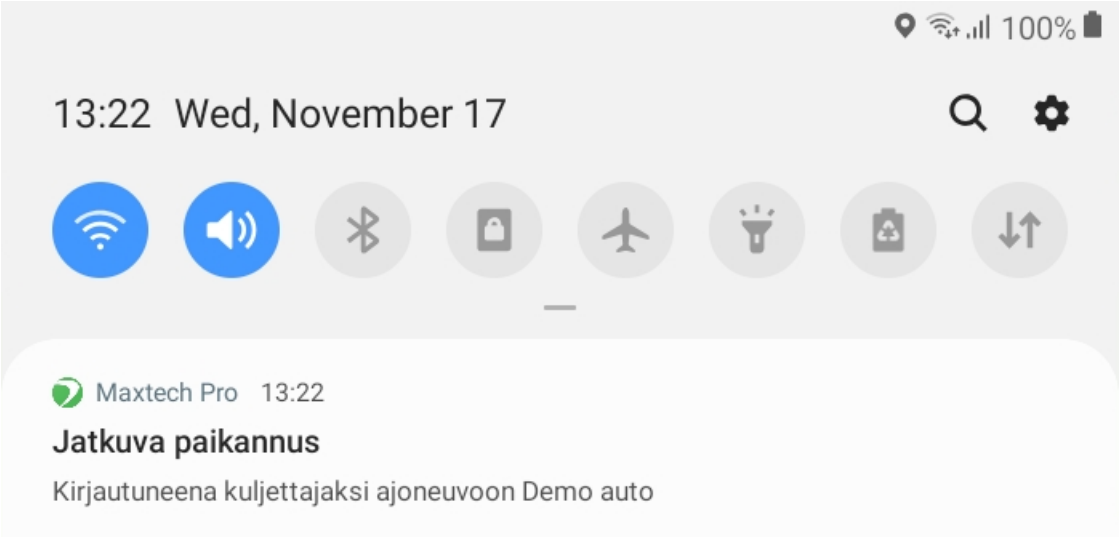
Virheiden käsittelyä varten moduulin komentoihin lisättiin parametriksi virhetilanteissa suoritettava errorCallback-funktio (kuva 5). Funktio suoritetaan, jos sijaintipalvelu kohtaa virheen kesken komennon suorittamisen. Toteutettu rakenne mahdollistaa palvelussa tapahtuvan virheen lähettämisen sovelluksen alustariippumattomalle osuudelle käsiteltäväksi, eikä palveluun tarvitse toteuttaa omaa logiikkaa virheidenkäsittelylle.

```
/**
 * Start location service
 *
 * @param {String} notificationTitle Service notification title
 * @param {String} notificationText Service notification text
 * @param {Object} config Service config
 * @param {Function} success Success callback
 * @param {Function} error Error callback
 */
const startService = (notificationTitle :String , notificationText :String , config :Object , success, error) => {...};

/**
 * Stop location service
 *
 * @param {Function} callback Callback
 */
const stopService = (callback) => {...};

/**
 * Update location request config
 *
 * @param {Object} config Service config
 * @param {Function} errorCallback Callback on error
 */
const updateConfig = (config :Object , errorCallback) => {...};

/**
 * Update location service foreground notification content
 *
 * @param {String} title Service notification title
 * @param {String} text Service notification text
 * @param {Function} errorCallback Callback on error
 */
const updateNotification = (title :String , text :String , errorCallback) => {...};
```

KUVA 5. Moduulin komennot palvelun ohjausta varten

Palvelun sijainnin hakemisen asetukset, ilmoituksen otsikko ja sisältö voidaan myös määritellä palvelun käynnistyskomennon startService parametreinä: notificationTitle, notificationText ja config (kuva 5). Näin ollen palvelun käynnistyksen yhteydessä palvelulle ei tarvitse lähettää useampaa komentoa.

Palvelun ilmoituksen otsikkoa ja tekstiä (kuva 6) voidaan muuttaa moduuliin toteutetulla komennolla updateNotification sovelluksen alustariippumattomasta JavaScript-osuudesta (kuva 5). Palvelu tarkkailee laitteen sijaintia Androidin sisäisen FusedLocationProviderClient-luokan requestLocationUpdates-funktiota (kuva 7). Laitteen sijainnin tarkkaileminen aloitetaan heti palvelun käynnistyessä.



KUVA 6. Palvelun jatkuva ilmoitus laitteen ilmoituspalkissa

Sijainnin hakemisen asetukset määritellään funktion parametrissa LocationRequest-objektissa (kuva 7). Muuttamalla sen asetuksia voidaan sijainnin hakemista mukauttaa sopivaksi erilaisiin käyttötarkoituksiin. Asetuksia muuttamalla voidaan esimerkiksi asettaa 30 sekunnin aikaväli sijainnin hakemiselle tai määrittää sijainti haettavaksi 20 metrin välein. Asetuksia voidaan muuttaa moduulin updateConfig-komennolla sovelluksen JavaScript-osuudesta (kuva 5).

Funktion locationCallback-parametri suoritetaan, kun uusi sijaintitieto on saatavilla. Toteutuksessa locationCallback-parametri osoittaa AndroidLocationServiceModule-moduulin funktioon, jossa sijaintitiedot lähetetään sovelluksen JavaScript-osuuteen (kuva 7).

```
fusedLocationProviderClient.requestLocationUpdates(
        locationRequest,
        locationCallback,
        Looper.myLooper()
);
```

KUVA 7. Laitteen sijaintipäivityksien tarkkaileminen toteutetussa palvelussa

Jotta sijaintitiedot voidaan lähettää sovelluksen JavaScript-osuuteen, muunnetaan Androidin sisäinen Location-objekti getLocationResponse-funktiossa React Native bridgen tukemaksi WritableMap-objektiksi (kuva 8). Lähetettävä objekti sisältää seuraavat avain-arvoparit:

- Latitude: GPS-pisteen leveysaste
- Longitude: GPS-pisteen pituusaste
- Accuracy: GPS-pisteen halkaisija metreinä
- Speed: Nopeus GPS-pisteen hetkellä
- Occurred: Aikaleima GPS-pisteen hetkellä.

```
/**
 * Get location response
 *
 * @param location Location
 *
 * @return Location map for react
 */
private WritableMap getLocationResponse(Location location) {
    WritableMap locationMap = Arguments.createMap();
    WritableMap coordMap = Arguments.createMap();

    coordMap.putDouble("latitude", location.getLatitude());
    coordMap.putDouble("longitude", location.getLongitude());
    coordMap.putDouble("accuracy", location.getAccuracy());
    coordMap.putDouble("speed", location.getSpeed());

    locationMap.putMap("coords", coordMap);
    locationMap.putDouble("occurred", location.getTime());

    return locationMap;
}
```

KUVA 8. Sijaintietojen muuttaminen sovelluksen JavaScript-osuuden tukemaan muotoon

5.2 LocationHandler – sijaintitietojen ja ajotapahtumien käsittelijä

Sijaintitietojen käsittely pyrittiin toteuttamaan mahdollisimman alustariippumattomasti, jotta samaa toteutusta voidaan hyödyntää sovelluksen kaikkien tuettujen alustojen välillä. Laitteen sijainnin käsittelyä varten toteutettiin JavaScript-kielellä alustariippumaton LocationHandler-luokka. Luokka vastaanottaa sijaintitietoja luvussa 5.1 toteutetulta natiivilta palvelulta sekä lähettää komentoja sille

natiivin moduulin välityksellä. Vastaanotetun sijaintitiedon perusteella LocationHandler hallinnoi ajotapahtumien automaattista aloittamista ja lopettamista.

LocationHandler-luokan tiedoston vientilauseke toteutettiin siten, että siitä on aina olemassa yksi ilmentymä sovelluksen muistissa. Vientilausekkeella varmistetaan, ettei luokan ilmentymää siivota pois sovelluksen muistista missään vaiheessa. (Kuva 10.)

```
class LocationHandler {...}

export default new LocationHandler();
```

KUVA 10. LocationHandler-tiedoston vientilauseke

Tiedoston vientilauseke (kuva 10) mahdollistaa sen, että luokasta on aina olemassa yksi ilmentymä sovelluksen muistissa. Kun sovelluksesta käännetään julkaistava versio, käännöstyökalu käy kaikkien tiedostojen vientilausekkeet, jolloin luokasta luodaan ilmentymä sovelluksen muistiin (12). Tuontilauseke LocationHandler-tiedostosta (kuva 11) viittaa siis aina samaan ilmentymään luokasta, joka luodaan vientilausekkeessa.

```
import LocationHandler from './app/LocationHandler';
```

KUVA 11. LocationHandler-tiedoston tuontilauseke

Jotta sovelluksen alustariippumaton osuus olisi tietoinen natiivin palvelun lähettämästä sijaintitiedoista, sovelluksen käynnistyksen yhteydessä luodaan NativeEventEmitter, jolle annetaan parametriksi natiivin sijaintipalvelun moduuli (kuva 12). (13.)

```
/**
 * Add new event emitter
 *
 * @return {NativeEventEmitter<AndroidLocationService>}
 */
const addNewEventEmitter = () => new NativeEventEmitter(AndroidLocationService);
```

KUVA 12. AndroidLocationService-moduulin lähettämiä tapahtumia tarkkaileva NativeEventEmitter

NativeEventEmitterille luodaan tarkkailija addListener-funktiolla, jonka parametreissä määritellään tarkkailtavan tapahtuman tunniste ja funktio, joka suoritetaan, kun tapahtuma havaitaan (13). Natiivin palvelun lähettäessä sijaintitietoja sovellukselle se ohjataan LocationHandler-luokan handeLocationData-funktioon, jossa varsinainen sijaintitietojen käsittely tapahtuu. Nyt sovelluksen JavaScript-osuus on tietoinen natiivin palvelun lähettämistä sijaintitiedoista. (Kuva 13.)

```
this.locationServiceEmitter = AndroidLocationService.addNewEventEmitter();

this.locationUpdateListener = this.locationServiceEmitter.addListener(
  EVENT_UPDATE_LOCATION,
  LocationHandler.handleLocationData
);
```

KUVA 13. Natiivin palvelun lähettämien sijaintietojen tarkkailu

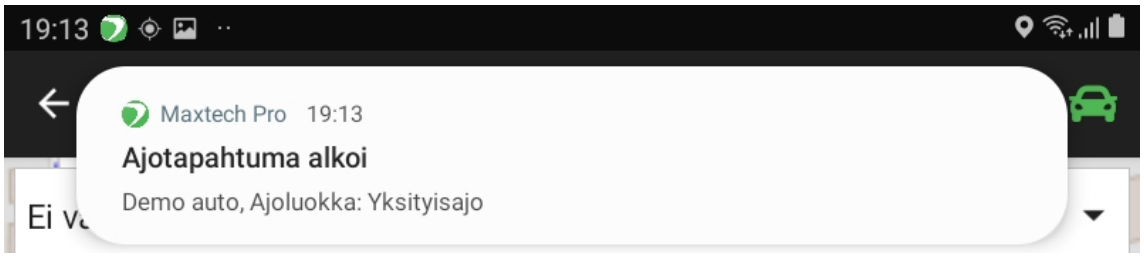
5.2.1 Ajotapahtumien automaattinen aloittaminen ja lopettaminen

Ajotapahtumien rakennetta oli jo tehty ennen työn aloittamista sovelluksen paikalliseen tietokantaan osana sovelluksen ajopäiväkirjaa, jonka avulla käyttäjä voi manuaalisesti syöttää ajettuja reittejä. Toteutuksen yhteydessä mobiilisovelluksen paikalliseen tietokantarakenteeseen lisättiin tuki GPS-pisteiden tallentamiselle osaksi ajotapahtumaa. Ajotapahtumien automaattisessa aloittamisessa ja lopettamisessa hyödynnettiin ajoneuvolle asetettavia matkan raja-arvoja, pienin sallittu liike ajotapahtuman aloittamiselle ja pienin sallittu liike GPS-pisteen lisäämiselle. Raja-arvoja voidaan muuttaa web-järjestelmän hallintakäyttöliittymästä, joka mahdollistaa ominaisuuden mukauttamisen erilaisiin olosuhteisiin.

Ajotapahtuma aloitetaan, kun laite on liikkunut yli asetetun matkan raja-arvon minuutin aikana. Raja-arvo ajotapahtuman aloittamiselle haetaan käytössä olevan ajoneuvon tiedoista. Oletuksena raja-arvo ajotapahtuman aloittamiselle on 50 metriä. Kuljettu matka lasketaan vertailemalla vastaanotetun GPS-pisteen etäisyyttä edelliseen vastaanotettuun GPS-pisteeseen käyttämällä avoimen lähdekoodin geolib-kirjastoa. Kuljettu matka nollataan, jos raja-arvoa ei ylitetä minuutin aikana.

Ajotapahtuman alkaessa LocationHandler luo sovelluksen paikalliseen tietokantaan uuden ajotapahtuman ja natiiville palvelulle lähetetään komento päivittää palvelun ilmoituksen sisältöä huomauttamaan käyttäjää alkaneesta ajotapahtumasta (kuva 14). Natiiville palvelulle lähetetään myös komento asettaa sijaintitietojen hakuväliksi käytössä olevan ajoneuvon raja-arvo ”pienin sallittu

matka GPS-pisteen lisäämiselle”. Oletuksena raja-arvo on 50 metriä. Kun ajotapahtuma on käynnissä, natiivilta palvelulta vastaanotettavat GPS-pisteet lisätään sen tietoihin paikallisessa tietokannassa.

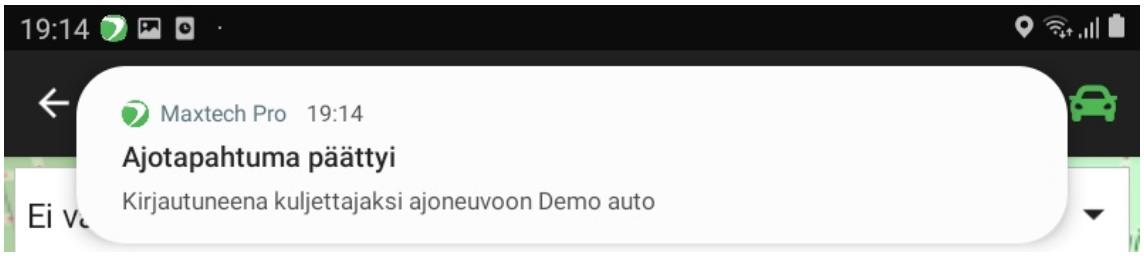


KUVA 14. Ilmoitus ajotapahtuman alkamisesta

Ominaisuudelle asetettujen vaatimusten mukaisesti ajotapahtumille tallennettavien sijaintietojen tulee olla tarpeeksi tarkkoja. Tietyissä olosuhteissa, esimerkiksi ajoneuvon ollessa parkkihallissa, natiivin palvelun lähettämä sijaintitieto voi olla hyvin epätarkkaa. Natiivin palvelun lähettämä sijaintitieto sisältää tarkkuusarvon (accuracy), joka määrittää GPS-pisteen halkaisijan metreinä. Jos vastaanotetun GPS-pisteen halkaisija on enemmän kuin 40 metriä, sitä ei liitetä osaksi ajotapahtumaa.

Ajotapahtuma päättyy, jos laite ei ole liikkunut 10 minuutin aikana. Ajotapahtumien lopettaminen toteutettiin ajastimella, joka nollataan, kun LocationHandlerin sijainnin käsittelyfunktio vastaanottaa uuden sijaintitiedon natiivilta palvelulta. Lopettaminen pystyttiin toteuttamaan melko yksinkertaisella ajastintoiminnallisuudella, koska ajotapahtuman alkaessa sijaintitietojen hakuväliksi asetetaan matkaan perustuva raja-arvo, jolloin palvelu ei lähetä yhtään uutta sijaintitietoa laitteen ollessa paikallaan.

Ajotapahtuman päättyessä natiiville palvelulle lähetetään komento päivittää ilmoituksen sisältöä huomauttamaan käyttäjää päättyneestä ajotapahtumasta (kuva 15). Palvelulle lähetetään myös komento poistaa sijainnin hakemisen asetuksista kuljettuun matkaan perustuva rajoitus, joka asetettiin tapahtuman aloittamisen yhteydessä, jonka jälkeen LocationHandler asetetaan taas tarkkailemaan ajotapahtumien automaattista aloitusta kuljetun matkan perusteella.



KUVA 15. Ilmoitus päättyneestä ajotapahtumasta

Ajotapahtuman tietojen lisäksi sijainnin käsittelyfunktiossa tallennetaan käyttäjän sijainti sovelluksen paikalliseen tietokantaan, jotta sitä voidaan käyttää sovelluksen muissa toiminnoissa. Tallennettua sijaintia hyödynnetään esimerkiksi käyttäjän siirtyessä sovelluksen karttanäkymään, jolloin kartta voidaan heti kohdistaa käyttäjän sijaintiin.

Toteutuksessa pyrittiin ottamaan huomioon mahdollisia virhetilanteita. Käyttäjä voi esimerkiksi kirjautua ulos ajoneuvosta kesken ajotapahtuman tai laitteen sijainti ei jostain syystä ole enää saatavilla. Virhetilanteissa natiiville palvelulle lähetetään pysäytyskomento ja mahdollisesti käynnissä oleva ajotapahtuma lopetetaan.

LocationHandler käsittelee myös natiiveissa palveluissa tapahtuvat virheet. Kaikissa natiivin palvelun komennoissa käytettävä errorCallback-parametri on asetettu osoittamaan LocationHandlerin virheidenkäsittelyfunktioon onServiceError. (Kuva 16.)

```
updateConfig = async (config) => {
    await AndroidLocationService.updateConfig(config, this.onServiceError);
}
```

KUVA 16. Natiiveissa palveluissa tapahtuvat virheet käsitellään sovelluksen JavaScript-osuudessa

5.2.2 Ajotapahtumien synkronointi web-järjestelmään

Toteutuksessa hyödynnettiin aikaisemmin tehtyjä rajapintoja ajotapahtumien aloittamiselle ja lopettamiselle. Web-järjestelmän rajapintaan lisättiin toiminnallisuus ajotapahtuman sijaintitietojen päivittämiselle. Rajapinta hyväksyy parametreinä päivitettävän ajotapahtuman tunnisteen ja sille lisättävät sijaintitiedot.

Ajotapahtuman sijaintitietoja synkronoidaan järjestelmään 1 minuutin välein. Rajapinnalle ei lähetetä yksittäisiä GPS-pisteitä, vaan synkronointiaikavälin sisällä vastaanotetut GPS-pisteet lähetetään samassa pyynnössä (kuva 17).


```
"driving_session_id": 225,
"gps_data": [
    {
        "latitude": 64.858078,
        "longitude": 25.514519,
        "occurred": "2021-11-17 20:25:39",
        "speed": 35
    },
    {
        "latitude": 64.858551,
        "longitude": 25.514887,
        "occurred": "2021-11-17 20:25:50",
        "speed": 25
    }
]
```


KUVA 17. Rajapinnalle lähettävät ajotapahtuman sijaintitiedot JSON-muodossa

Jotta rajapintaan ei lähettäisi jo synkronoituja GPS-pisteitä, pidetään sovelluksen paikallisessa tietokannassa kirjaa onnistuneesti lähetettyjen pisteiden määrästä. Rajapintaan lähetettävää pyyntöä muodostaessa siihen lisätään ainoastaan GPS-pisteet, joita ei ole vielä synkronoitu (kuva 18).

```
/**
 * Format gps data for update request
 *
 * @param {Array<Location>} gpsData Driving session gps data
 * @param {int} startIndex Start index for gps data
 *
 * @return {Array} Gps data for gps update request
 */
const formatGpsData = (gpsData, startIndex) => {
  const gpsPoints = [];
  for (let i = startIndex + 1; i < gpsData.length; i += 1) {
    const gpsDataPoint = gpsData[i];
    if (gpsDataPoint != null) {
      gpsPoints.push({
        latitude: gpsDataPoint.latitude,
        longitude: gpsDataPoint.longitude,
        speed: gpsDataPoint.speed * 3.6, // meters per second -> kilometers per hour
        occurred: Moment(gpsDataPoint.occurred).format('YYYY-MM-DD HH:mm:ss', 'en-US')
      });
    }
  }
```

KUVA 18. Rajapintaan lähetettävien sijaintietojen muodostus

Mobiilisovelluksessa varmistetaan, että ajotapahtuma on luotu järjestelmään, ennen kuin sijaintitietoja voidaan lähettää vastaanottavalle rajapinnalle, koska on mahdollista, ettei ajotapahtumaa ole vielä synkronoitu web-järjestelmään ja se on olemassa ainoastaan sovelluksen paikallisessa tietokannassa.

Sovelluksen päädyssä ei myöskään voida sokeasti luottaa siihen, että rajapintaan lähetetty pyyntö on mennyt perille asti. Jos sijaintietojen lähetys palvelimelle ei onnistu, sovellus puskuroi lähetettäviä pyyntöjä sen paikalliseen tietokantaan ja ne lähetetään myöhempänä ajankohtana, jolloin yhteys web-järjestelmään on saatavilla.

5.3 Karttanäkymän toiminnallisuuksien laajentaminen

5.3.1 Paikannuksen aktivointi

Työlle asetettujen vaatimusten mukaan paikannuksen aktivointi täytyy olla mahdollisimman helppoa ominaisuuden käyttäjälle. Karttanäkymän yläpalkin oikeaan laitaan lisättiin aktivointipainike jatkuvalle paikannukselle (kuva 19), joka näkyy käyttäjälle vain, jos jatkuvan paikannuksen ominaisuus on kytkettynä päälle web-järjestelmässä sijaitsevasta käyttäjien hallinnasta.

KUVA 19. Karttanäkymän yläpalkin oikeaan laitaan lisätty aktivointipainike

Käyttäjä voi käynnistää painikkeesta paikannustoiminnallisuuden, jos sovellukselle on annettu lupa käyttää laitteen sijaintia taustalla ja sijainti on saatavilla. Paikannuksen aktivoimiseksi käyttäjän täytyy myös olla kirjautuneena ajoneuvon kuljettajaksi, jotta muodostettavalle ajotapahtumalle voidaan liittää ajoneuvo. Työssä hyödynnettiin sovellukseen aikaisemmin toteutettua kuljettajaksi kirjautumisen toimintoa. Kuljettajaksi kirjautuminen ajoneuvoon tapahtuu lukemalla web-järjestelmän kautta lisätty QR- tai NFC-tunniste, jonka toiminnoksi on asetettu kuljettajaksi kirjautuminen.

Paikannuksen aktivoinnin onnistuessa luvussa 5.2 käsitelty LocationHandler-luokka lähettää käynnistyskomennon natiiville palvelulle ja ajotapahtumien automaattinen muodostaminen aloitetaan (kuva 20). Paikannus voidaan kytkeä pois samasta painikkeesta, jolloin natiiville palvelulle lähetetään pysäytyskomento ja mahdollisesti käynnissä oleva ajotapahtuma lopetetaan.

KUVA 20. Jatkuvan paikannuksen onnistunut aktivointi

Jos paikannusta ei voida käynnistää, käyttäjälle näytetään virheilmoitus karttanäkymän alalaidassa ja aktivointipainikkeen kuvaketta muutetaan indikoimaan virhetilannetta. Virhetilanteissa käyttäjälle ilmoitetaan selkeästi, miksi paikannuksen käynnistäminen ei onnistu. (Kuva 21.)

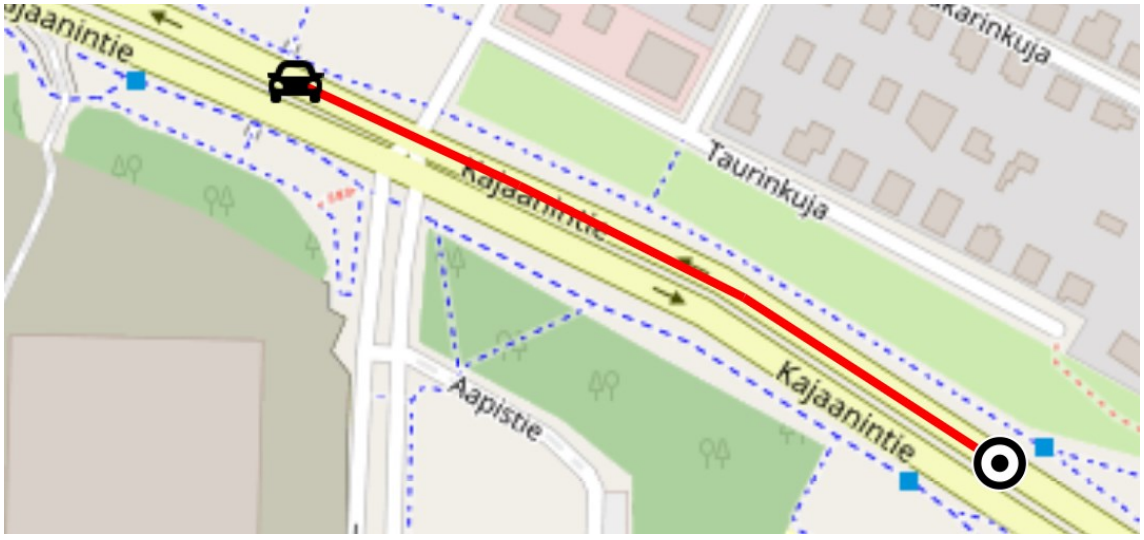
KUVA 21. Käyttäjälle näytettään virheilmoitus paikannuksen aktivoinnin epäonnistuessa

5.3.2 Ajotapahtumien tarkastelu

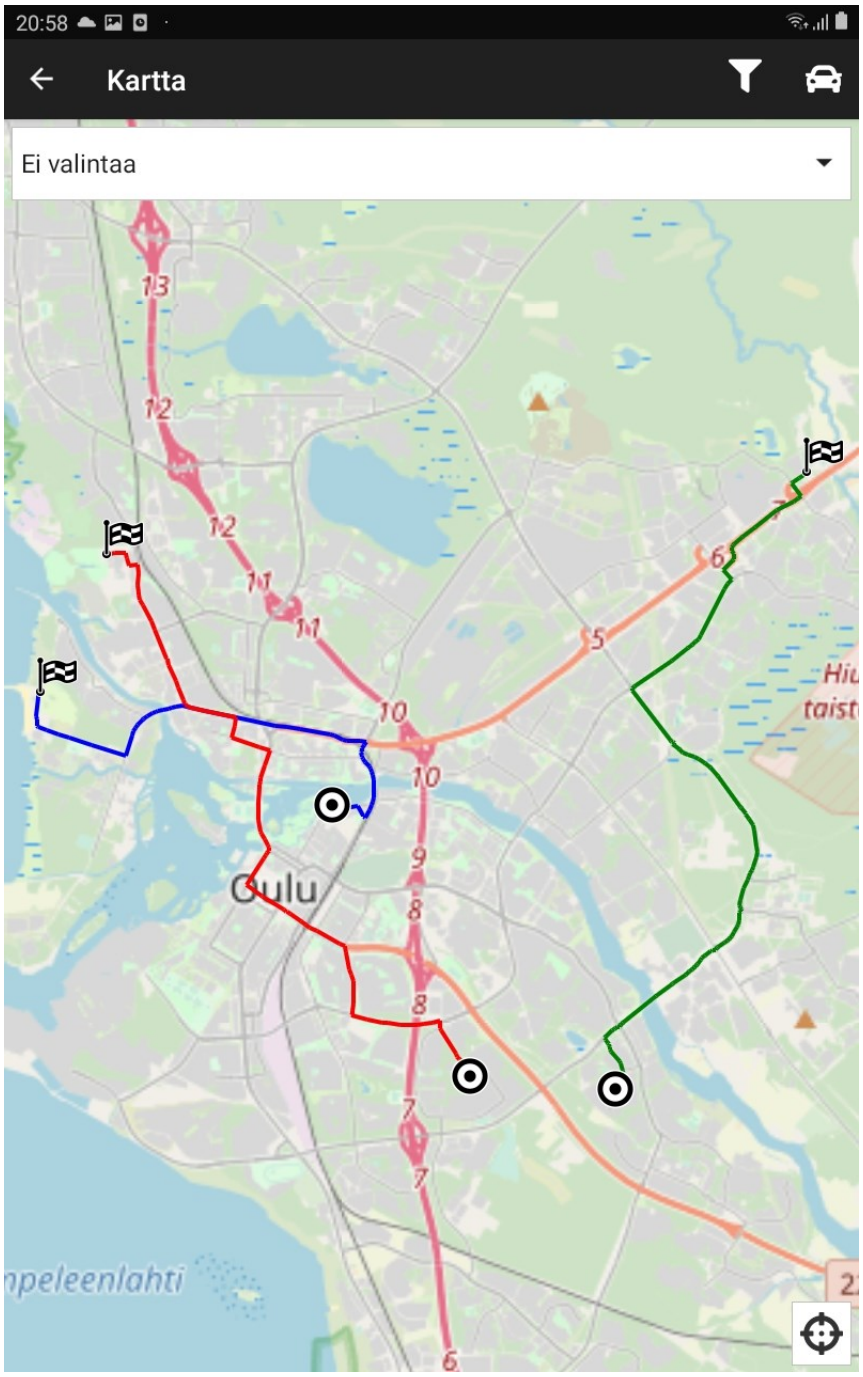
Karttanäkymään lisättiin komponentti, josta ilmenee käynnissä olevan ajotapahtuman tiedot. Komponentista ilmenee ajotapahtuman ajoneuvo, ajoluokka, kesto, kuljettu matka ja muodostunut kilometrikorvaus (kuva 22).

KUVA 22. Komponentti käynnissä olevan ajotapahtuman tietojen näyttämiselle

Karttanäkymään toteutettiin toiminnallisuus, josta käyttäjä näkee aktiivisen ajotapahtuman reitin (kuva 23) sekä päivän aikana ajamansa reitit kartalla (kuva 24). Reitin väri määräytyy ajotapahtumalle asetetun ajoluokan värin mukaan, jos ajoluokkaa ei ole asetettu, reitti piirretään mustana. Reitin muodostamisessa hyödynnettiin sovelluksen käytössä olevan karttakirjaston valmista Polyline-komponettia, jolle annettaan tiedoiksi ajotapahtuman GPS-pisteet ja ajoluokalle asetettu väri (kuva 25). Reitin alku- ja loppupisteeseen lisätään kuvakkeet, jotta siitä erottaisi loppu- ja alkupään. Jos ajotapahtuma on käynnissä, ei reitin loppuun lisätä lopetuskuvaketta.



KUVA 23. Käynnissä olevan ajotapahtuman reitti kartalla



KUVA 24. Käyttäjän päivän aikana ajamat reitit sovelluksen karttanäkymässä

```
/**
 * Render route
 *
 * @param {Object} route Route
 * @param {String} color Route color
 * @param {String} key Key
 * @param {Boolean} ongoing Is ongoing
 *
 * @return {Component} Polyline from route
 */
const renderRoute = (route, color, key, ongoing = false) => {
  if (route.length <= 1) {
    return null;
  }

  return (
    <Fragment key={key}>
      <Polyline
        coordinates={route}
        strokeColor={color ?? 'black'}
        strokeWidth={3}
      />
      <Marker
        image={startMarker}
        anchor={{ x: 0.5, y: 0.5 }}
        coordinate={getFirstCoordinate(route)}
      />
      { !ongoing && (
        getLastMarker( ongoing: false, route)
      )}
    </Fragment>
  );
};
```

KUVA 25. Reittikomponentin muodostusfunktio

5.3.3 Ajoluokan valinta

Karttanäkymään toteutettiin ajotapahtumissa käytettävän ajoluokan valintakomponentti. Komponentissa näytetään ajoneuvolle web-järjestelmässä asetut ajoluokat, jotka on synkronoitu mobiilisovelluksen paikalisen tietokantaan (kuva 26). Valittu ajoluokka tallennetaan sovelluksen paikalliseen tietokantaan ja sitä käytetään määrittämään aloitettavan ajotapahtuman ominaisuuksia, esimerkiksi sille muodostuva kilometrikorvaus. Jos ajoluokkaa vaihdetaan kesken ajotapahtuman, käynnissä oleva ajotapahtuma lopetetaan ja valitulla ajoluokalla aloitetaan uusi ajotapahtuma.

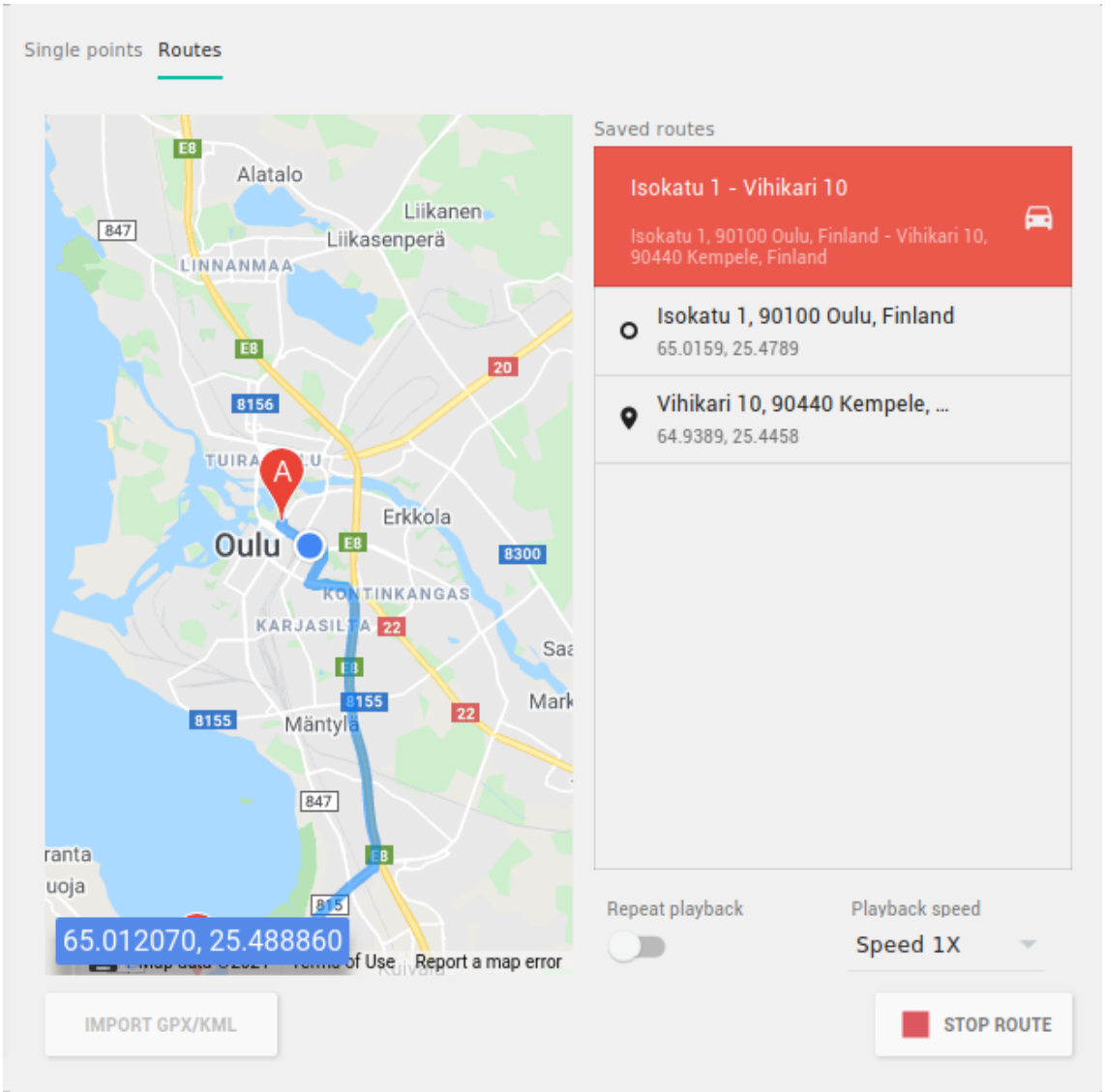
KUVA 26. Ajoluokan valintakomponentti karttanäkymän alalaidassa

Kun käyttäjä kirjautuu ulos ajoneuvosta, sovelluksen tietokannasta poistetaan aktiivisen ajoluokan valinta mahdollisten virhetilanteiden välttämiseksi. Toteutuksen aikana havaittiin, että käyttäjä voisi mahdollisesti kirjautua kuljettajaksi toiseen ajoneuvoon, jolle ei ole mahdollista valita samaa ajoluokkaa, ja aloittaa virheellisiä ajotapahtumia kyseisellä ajoluokka.

Tilanteissa, joissa ajoneuvolle ei ole asetettuna yhtään ajoluokkaa tai käyttäjä ei ole valinnut aktiivista ajoluokkaa, käytetään web-järjestelmässä käyttäjälle asetettua oletusajoluokkaa. Jos käyttäjälle ei ole asetettuna oletusajoluokkaa, ei ajotapahtumille aseteta mitään ajoluokkaa. Ajoluokan puuttuessa ajotapahtumalle ei voida muodostaa kilometrikorvausta. Jos käyttäjä haluaa asettaa ajotapahtumalle ajoluokan myöhemmin, voi hän muokata sen tietoja sovelluksen ajopäiväkirjanäkymän kautta.

5.4 Ominaisuuden testaaminen

Toteutuksen aikana ominaisuutta pääasiassa testattiin Android Studion emulaattorissa, jossa simuloitiin ajettavia reittejä (kuva 27). Emulaattorissa testaus mahdollisti tehokkaan kehitystyön, sillä muutoksien jälkeen ominaisuuden toimivuutta ei tarvinnut testata oikealla ajoneuvolla. Simuloidut reitit eivät kuitenkaan vastaa ominaisuuden oikeita käyttöolosuhteita, sillä niiden sijaintitiedot ovat tarkkuudeltaan ihanteellisia.



KUVA 27. Reitin simulointi Android Studion emulaattorissa

Toteutuksen loppupuolella ominaisuutta testattiin myös enemmän käyttötarkoitusta vastaavissa olosuhteissa eli tehtiin nk. kenttätestaus. Kenttätestauksen aikana henkilöautolla ajettiin Oulun keskustassa ja sen lähialueilla jatkuvan paikannuksen ollessa aktivoituna. Suurimman osan ajasta joko laitteen näyttö oli kiinni tai sovellus oli taustalla. Pääasiallinen tavoite oli varmistaa, että muodostettavien ajotapahtumien sijaintiedot ovat tarpeeksi todenmukaisia, myös sovelluksen ollessa taustalla. Kenttätestauksen tulokset käsitellään seuraavassa luvussa muiden työn tulosten ohessa.

Testilaitteena toimi Samsung Galaxy Tab Active 2 (Android 9 käyttöjärjestelmällä), josta oli kytkettynä pois sovelluskohtaiset virransäästöasetukset Maxtech Pro -mobiilisovelluksesta. Ajojen aikana testilaite oli ajoneuvon kojelaudan päällä, jotta laitteen GPS-signaali olisi mahdollisimman vahva. (Kuva 28.)

KUVA 28. Testilaite ajoneuvon kojelaudan päällä

6 TULOKSET

Opinnäytetyön tehtävänä oli suunnitella ja toteuttaa Maxtech Pro -mobiilisovellukseen taustapaikannus-toiminnallisuus, josta saatavilla sijaintitiedoilla mobiilisovellus osaa muodostaa ajotapahtumia automaattisesti. Osana työtä sovelluksen karttanäkymän käyttöliittymään lisättiin toiminnallisuuksia ajotapahtuman tietojen tarkastelua ja hallinnointia varten.

Työ aloitettiin esiselvityksellä, jossa tutkittiin, voidaanko paikannustoiminnallisuus toteuttaa sovellukseen tarpeeksi luotettavasti. Esiselvityksen tuloksena syntyi esiselvitysdokumentti (liite 1), jonka perusteella kehitystyö päätettiin käynnistää.

Android-alustalle toteutettiin natiivi Foreground Service -palvelu laitteen sijainnin hakemiselle, joka pystyy luotettavasti tarkkailemaan laitteen sijaintia myös sovelluksen ollessa taustalla. iOS-alustan tuki päätettiin jättää työn ulkopuolelle ja siirtää osaksi jatkokehitystä työn rajallisen aikataulun vuoksi.

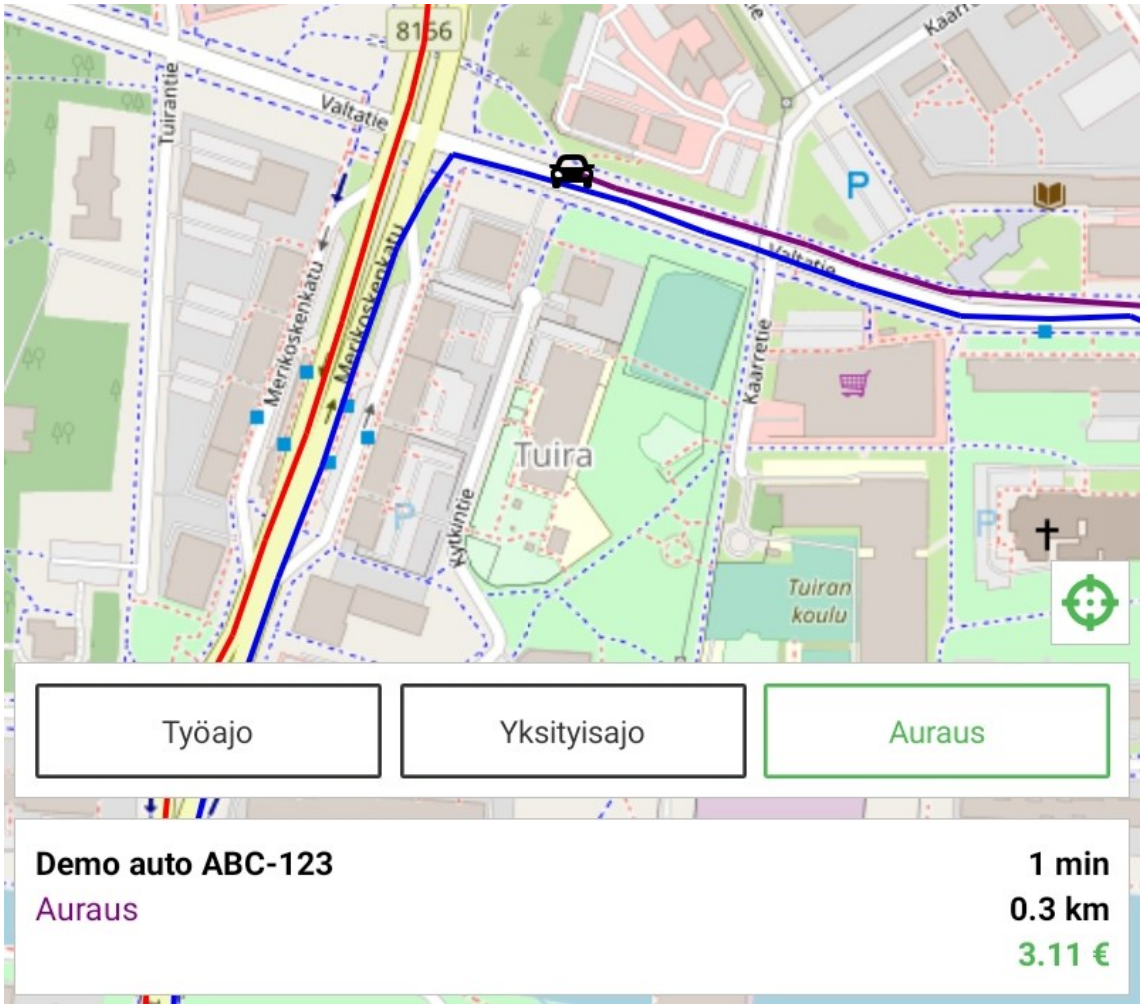
Ajotapahtumien muodostamiseen haluttiin mahdollisimman automaattinen ratkaisu, joka ei vaatisi ylimääräisiä toimenpiteitä sovelluksen käyttäjältä. Sijaintietojen ja ajotapahtumien käsittelyä varten sovelluksen alustariippumattomaan JavaScript-osuuteen toteutettiin LocationHandler-luokka, joka hallinnoi ajotapahtumien aloittamista ja lopettamista vastaanotetun sijaintitiedon perusteella.

Web-järjestelmässä olevaa rajapintaa laajennettiin lisäämällä tuki ajotapahtumien sijaintietietojen päivittämiselle järjestelmään. Mobiilisovellukseen toteutettiin taustasynkronointi, joka päivittää aktiivisen ajotapahtuman sijaintitietoja web-järjestelmään 1 minuutin välein.

Sovelluksen karttanäkymään lisättiin toiminnallisuuksia tukemaan muodostettuja ajotapahtumia. Karttanäkymään lisättiin komponentti, josta ilmenee käynnissä olevan ajotapahtuman tiedot: kuljettu matka, kesto ja muodostunut kilometrikorvaus. Aktiivisen ajoluokan valitsemiselle toteutettiin oma komponentti karttanäkymään. Ajoluokan vaihtaminen kesken käynnissä olevan ajotapahtuman aloittaa uuden ajotapahtuman. Karttanäkymään lisättiin myös ominaisuus, josta käyttäjä näkee päivän aikana ajamansa reitit. Piirrettävän reitin väri määräytyy ajoluokalle asetetun värin mukaan. Myös käynnissä olevan ajotapahtuman reittiä piirretään kartalle.

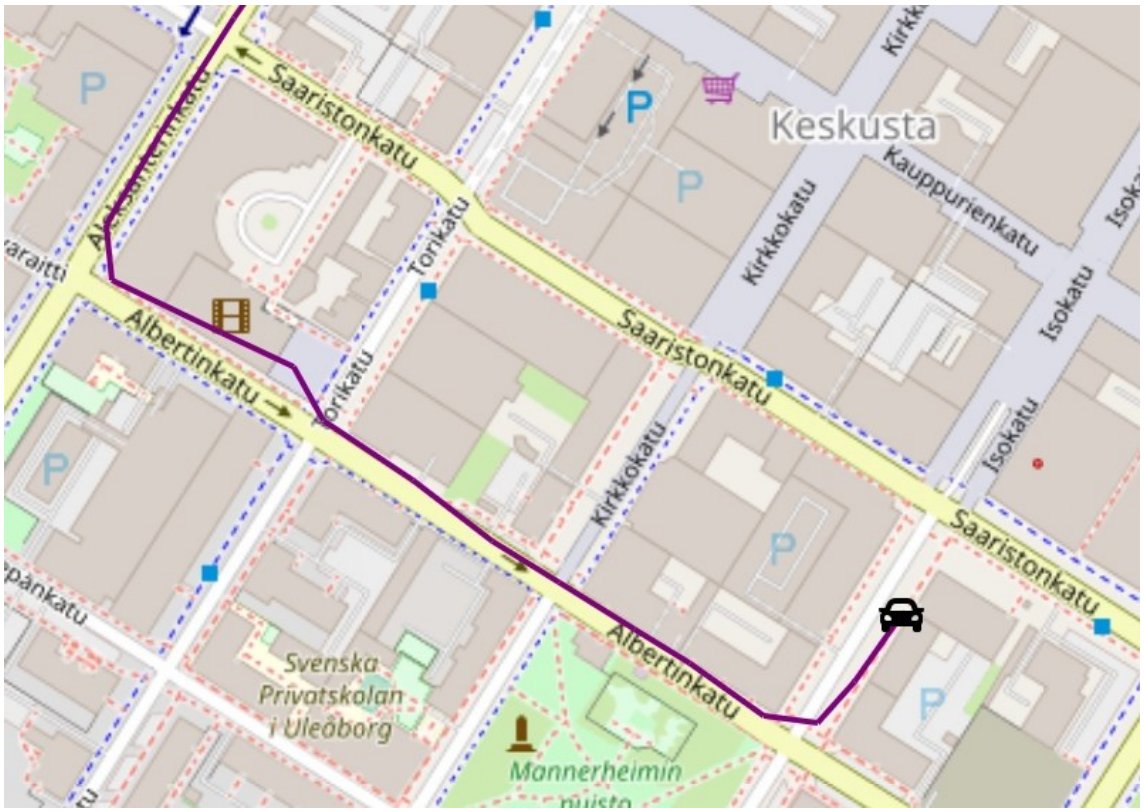
Alun perin karttanäkymään lisättyä aktivointipainiketta ei ollut tarkoitus toteuttaa, vaan ominaisuuden aktivointi olisi tapahtunut pelkästään web-järjestelmän kautta päälle kytkettävästä moduulista, jolloin paikannus aktivoituisi heti sovelluksen käynnistymisen yhteydessä. Kehitystyön edetessä ominaisuutta esiteltiin aktiivisesti yrityksen sisällä ja pelkkä moduulipohjainen aktivointi nähtiin ongelmalliseksi.

Tilaajan asettamiin vaatimuksiin päästiin, lukuun ottamatta iOS-alustan tukea ominaisuudelle. Työn loppuvaiheessa tehdyn kenttätestauksen perusteella ajotapahtumien automaattinen muodostaminen todettiin toimivaksi käyttötarkoitusta vastaavissa olosuhteissa. Testauksen aikana muodostuneet reitit olivat välillä jopa niin tarkkoja, että vierekkäisistä reiteistä pystyi hahmottamaan ajetun kaistan (kuva 29).



KUVA 29. Kenttätestauksen aikana ajettuja reittejä

Oulun keskustassa ajaessa muodostetun reitin tarkkuus kärsi ja se välillä oikaisi rakennuksien läpi (kuva 30). Epätarkkuudet reitissä todennäköisesti johtuvat korkeiden rakennusten aiheuttamista häiriöistä GPS-signaalissa.



KUVA 30. Kenttätestauksen aikana ajettua reittiä Oulun keskustassa

7 JATKOKEHITYS

Esiselvityksen aikana ominaisuuden käyttötapausta rajattiin siten, että sitä olisi aluksi tarkoitus käyttää pelkästään jatkuvassa virransyötössä. Parhaan mahdollisen sijaintitiedon saamiseksi käyttäjää pitäisi kehottaa kytkemään laitteen sovelluskohtaiset virransäästöasetukset pois. Virransäästöasetuksien poiskytkemiseen joudutaan tekemään laitekohtaisia ohjeita, koska Android-alustalla laitevalmistajilla on vapaus määrittää järjestelmässä käytettävät virransäästöasetukset. Laitevalmistajilla voi myös olla omia akunkäytön optimointisovelluksia.

Toteutuksen aikana päätettiin, että iOS-alustan tuki ominaisuudelle jätetään työn ulkopuolelle ja se tehdään osana jatkokehitystä. Työn aikana toteutettua LocationHandler-luokkaa voidaan hyödyntää iOS-alustalla ajotapahtumien käsittelyssä ja muodostamisessa, eikä vastaava toiminnallisuutta tarvitse toteuttaa uudestaan.

Toteutuksessa hyödynnettiin ajoneuvon tietokantarakenteeseen aiemmin lisättyjä matkan raja-arvoja ajotapahtumien automaattiseen aloittamisen ja GPS-pisteen lisäysvälin määrittelemiseen. Nykyisessä versiossa kaikkia LocationHandler-luokan käyttämiä raja-arvoja ei voida muuttaa web-järjestelmän kautta, joten ajotapahtumien GPS-datan synkronointiväli web-järjestelmään on aina 1 minuutti ja ajotapahtumien automaattinen lopetus 10 minuuttia. Näille raja-arvoille tulisi lisätä kentät ajoneuvon tietokantarakenteeseen ja mahdollistaa niiden muuttaminen web-järjestelmän hallinta-käyttöliittymästä, jotta ominaisuutta voidaan mukauttaa mahdollisimman monenlaisiin käyttötarpeisiin.

Karttanäkymän käyttöliittymän skaalausta on tarkoitus parantaa vaakatilassa olevilla näytöillä ja tablet-laitteilla. Nykyisessä versiossa käyttäjän voi olla hankala hahmottaa mihin ajoneuvoon hän on kirjautuneena kuljettajaksi ennen paikannuksen käynnistämistä, joten mielestäni käytössä olevan ajoneuvon tiedot olisi hyvä lisätä karttanäkymään. Työn rajallisen aikataulun vuoksi kehitystyössä keskityttiin laitteen sijainnin hakemiseen taustalla ja ajotapahtumien automaattiseen muodostamiseen, eikä käyttöliittymän hiomiseen jäänyt paljoa aikaa.

Toteutuksen alkuvaiheessa havaittiin, että ajotapahtumien reitti saattaa oikaista käännöksien aikana (kuva 31), koska nykyisessä versiossa GPS-pisteen lisääminen perustuu pelkästään ajoneuvolle asetetun matkan raja-arvon ylitykseen, joka on oletuksena 50 metriä. Tallennettavan reitin

todenmukaisuutta parannetaan jatkokehityksessä tarkkailemalla laitteen kiihtyvyysanturia ja hakemalla ylimääräinen GPS-piste, jos kiihtyvyyden raja-arvo ylitetään. Vaihtoehtoisesti reitin todenmukaisuutta voisi parantaa pienentämällä matkan raja-arvoa esimerkiksi 10 metriin, mutta tuolloin ominaisuus käyttäisi huomattavasti enemmän laitteen resursseja.

KUVA 31. Ajotapahtumien reitti saattaa oikaista käännöksissä

Android-alustan sijaintipalvelun ilmoitukseen voisi lisätä painikkeita pikatoiminnoille. Mahdollinen pikatoiminto voisi esimerkiksi olla ajoluokan vaihto. Pikatoiminnot tekisivät ominaisuuden käytöstä tehokkaampaa, sillä käyttäjän ei tarvitse avata sovellusta ja siirtyä sen karttanäkymään vaihtaakseen aktiivista ajoluokkaa. Toinen mahdollinen pikatoiminto voisi olla ajotapahtuman manuaalinen lopettaminen.

8 POHDINTA

Työ oli kohtuullisen haastava ja mielenkiintoinen toteuttaa. Taustapaikannuksen toteuttamisesta React Native -viitekehyksellä kehitettyyn sovellukseen oli haastava löytää suoria ohjeita, joten suunnitteluvaiheen aikana perehdyin Android-alustan ratkaisuihin sijainnin hakemiselle sovelluksen ollessa taustalla ja React Nativen natiiveihin moduuleihin. Suoran esimerkin tai ohjeiden puutteen vuoksi tietoa piti soveltaa useasta eri lähteestä kokonaisuuden muodostamiseksi.

Olen tyytyväinen varsinkin sijaintidatan käsittelyyn toteutettuun LocationHandler-luokkaan, jota voidaan hyödyntää kaikilla tuetuilla alustoilla. Mielestäni myös natiivi sijaintipalvelu Android-alustalle saatiin toteutettua siten, ettei se vaadi enää suurta jatkokehitystä. Arvioisin, että jatkokehityksessä voidaan pääasiassa keskittyä alustariippumattoman LocationHandler-luokan kehitykseen.

Työlle olisi voinut varata enemmän aikaa, jolloin esimerkiksi paremman käyttöliittymän suunnittelulle ja toteutukselle olisi jäänyt enemmän aikaa. Opinnäytetyö oli myös välillä tauolla kiireellisempien työtehtävien vuoksi. Kehitystyön ja raportin kirjoittamisen aikana tuli myös hyviä ideoita ominaisuuden jatkokehitystä varten.

Olen kuitenkin tyytyväinen saavutettuun kokonaisuuteen. Tuloksiin oltiin tyytyväisiä myös yrityksessä ja työn etenemistä seurattiin ja esiteltiin säännöllisin väliajoin yrityksen sisällä. Opinnäytetyön aikana ominaisuuden jatkokehitystä varten saatiin hyvä pohja. Jatkuvan paikannuksen siirtäminen tuotantoon vaati laajan testauksen suurella määrällä eri laiteitta, jotta ominaisuudesta saadaan mahdollisimman luotettava asiakaskäyttöön.

LÄHTEET

1. React 2021. Introducing JSX. React. Hakupäivä 22.11.2021. https://reactjs.org/docs/introducing-jsx.html

2. React 2021. Components and Props. React. Hakupäivä 16.11.2021. https://reactjs.org/docs/components-and-props.html

3. React 2021. State and Lifecycle. React. Hakupäivä 22.11.2021. https://reactjs.org/docs/state-and-lifecycle.html

4. Erikson, Mark 2020. Blogged Answers: A (Mostly) Complete Guide to React Rendering Behavior. Mark's Dev Blog. Hakupäivä 22.11.2021. https://blog.isquaredsoftware.com/2020/05/blogged-answers-a-mostly-complete-guide-to-react-rendering-behavior

5. Chandra, Lama 2021. What is JSX?. Medium. Hakupäivä 22.11.2021. https://medium.com/@iamme24cl/what-is-jsx-cf0d2dbde8da

6. TechAhead Team 2021. The history of React Native: Facebook's Open Source App Development Framework. Techahead. Hakupäivä 15.11.2021. https://www.techaheadcorp.com/blog/history-of-react-native/

7. Abernathy, Christian – White, Eli – Wei, Luna – Yung, Timothy 2021. React Native's Many Platform Vision. React Native. Hakupäivä 15.11.2021. https://reactnative.dev/blog/2021/08/26/many-platform-vision

8. Frachet, Marvin 2017. Understanding the React Native bridge concept. Hackernoon. Hakupäivä 18.11.2021. https://hackernoon.com/understanding-react-native-bridge-concept-e9526066ddb8

9. React Native 2021. Core Components and Native Components. React Native. Hakupäivä 22.11.2021. https://reactnative.dev/docs/intro-react-native-components

10. React Native 2021. Native Modules Intro. React Native. Hakupäivä 22.11.2021. https://react-native.dev/docs/native-modules-intro

11. Android 2021. Background Location Limits. Android. Hakupäivä 19.11.2021. https://developer.android.com/about/versions/oreo/background-location-limits

12. Yagnik, Arnav 2021. Singleton Object in React Native. Stack Overflow. Hakupäivä 21.11.2021. https://stackoverflow.com/a/44720030

13. React Native 2021. Android Native Modules. React Native. Hakupäivä 22.11.2021. https://reactnative.dev/docs/native-modules-android

MAXTECH

Esiselvitysdokumentti

Miikka Kortelainen 10.08.2021

Jatkuvan paikannuksen toteuttaminen Pro Mobileen - esiselvitysdokumentti

Tässä dokumentissa on kerättynä ajatuksia jatkuvan paikannuksen toteuttamisesta Pro Mobileen. Dokumentissa on pääasiassa keskitytty esiselvitystyön aikana ilmenneisiin ongelmiin jatkuvan paikannuksen toteuttamisessa, sekä esitetty ratkaisuja ja alustavia rajauksia toteutukseen, joilla jatkuvan paikannuksen data olisi mahdollisimman tarkkaa.

Kokonaisuus lyhyesti

Toteutuksen keskiössä on karttanäkymä, jossa näytetään käyttäjän sijainti ja ajoneuvon tiedot, johon käyttäjä on kirjautunut kuljettajaksi. Karttanäkymästä voi helposti vaihtaa ajoluokkaa. Ajoluokan vaihtaminen aloittaa uuden ajon ja ajolle muodostuu kilometrikorvaus ajoluokan mukaan.

Taustaprosessi kuuntelee käyttäjän sijaintia myös silloin kun laitteen näyttö on kiinni, sekä tunnistaa automaattisesti laitteen paikka- ja nopeusdatasta ajon alkamisen ja loppumisen.

Keskeiset ongelmat

Virransäästöasetukset

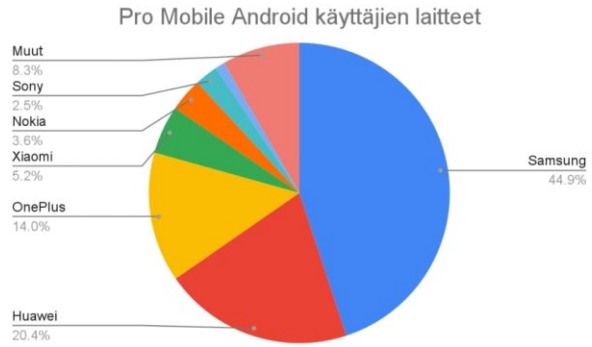
Suurin ongelma toteutuksessa on virransäästöasetusten vaikutus gps-datan tarkkuuteen. Kun laite on virransäästötilassa, gps-datan hakemista rajoitetaan ja sen tarkkuus kärsii.

Androidissa on eroa laitevalmistajien kesken virransäästöasetuksissa. Jotkut laitevalmistajat asettavat tiukempia rajoituksia taustaprosessien suorittamiselle sekä optimoivat sovellusten suorittamista taustalla. Laitevalmistajilla saattaa myös olla omia akunkäytön optimointisovelluksia.

Max Technologies Oy
2145296-2

Vihikari 10
90440 Kempele
Finland

+358 10 229 6200
www.maxtech.fi
info@maxtech.fi



Yllä olevassa kuvaajassa näkyy Pro Mobilen Android käyttäjien laitevalmistajien jakauma, suurimmilla laitevalmistajilla on useita kymmeniä eri laitemalleja. Suuresta määrästä erilaisia laitteita johtuen, mielestäni ei ole järkevää alkaa optimoimaan yksittäisiä laitemalleja.

Jatkuva sijainnin hakeminen myös kuluttaa paljon akkua, joten mielestäni paras ratkaisu alkuun olisi rajata ominaisuuden käyttöympäristöä tilanteisiin, joissa laitteella on jatkuva virransyöttö. Laite voisi esimerkiksi olla kytkettynä ajoneuvossa telineeseen ja sille olisi tarjolla jatkuva virransyöttö.

Yleispätevänä ratkaisuna käyttäjää voitaisiin kehottaa kytkemään pois sovelluskohtaiset virransäästöasetukset Pro Mobilen osalta. Tämä ei välttämättä kytke pois kaikkia laitevalmistajan omia virransäästöasetuksia, mutta auttaa kuitenkin asiassa.

iOS Taustaoperaatioiden keston rajoitus

iOS:ssa on rajoituksia taustaoperaatioiden kestossa. Laite saattaa lopettaa taustaoperaation, jos esimerkiksi datan lähetys web-järjestelmään kestää liian pitkään.

Max Technologies Oy
2145296-2

Vihikari 10
90440 Kempele
Finland

+358 10 229 6200
www.maxtech.fi
info@maxtech.fi

Tästä voi mahdollisesti seurata ongelmia, jos laitteen sijaintia halutaan seurata reaaliaikaisesti web-järjestelmästä, eli palvelimille pitäisi lähettää jatkuvasti dataa taustalta.

Johtopäätökset

Mielestäni jatkuva paikannus on mahdollista toteuttaa Pro Mobileen hyvällä tarkkuudella, mutta ominaisuuden käyttöympäristöä olisi aluksi syytä rajata tilanteisiin, joissa laitteella on jatkuva virransyöttö, esimerkiksi kytkettynä telineeseen ajoneuvossa. Rajaus voidaan poistaa, kunhan kehitystyö etenee ja saadaan tietoa, että paikannus toimii tarkasti myös ilman jatkuvaa virransyöttöä.

Aluksi toteutettaisiin gps-datan kuunteleminen ja ajojen tunnistaminen mahdollisimman tarkasti laitteen päädyssä. Jatkokehityksessä voidaan miettiä ajotietojen reaaliaikaista lähettämistä web-järjestelmään, jos sille on tarve.

Näillä rajauksilla kehitystyö olisi helpompi aloittaa. Vaikka ominaisuuden käyttötarkoitus olisikin aluksi rajattu pelkästään jatkuvaan virransyöttöön, kehityksessä kuitenkin pyritään pitämään taustaprosessin virrankulutus mahdollisimman vähäisenä.

Max Technologies Oy
2145296-2

Vihikari 10
90440 Kempele
Finland

+358 10 229 6200
www.maxtech.fi
info@maxtech.fi