OULUN AMMATTIKORKEAKOULU

Herkko Karjalainen

HYBRIDISOVELLUKSEN LAAJAMITTAINEN KEHITTÄMINEN

HYBRIDISOVELLUKSEN LAAJAMITTAINEN KEHITTÄMINEN

Herkko Karjalainen
Opinnäytetyö
Syksy 2023
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehityksen suuntautumisvaihtoehto

---

Tekijä(t): Herkko Karjalainen
Opinnäytetyön nimi: Hybridisovelluksen laajamittainen kehittäminen
Työn ohjaaja(t): Eino Niemi
Työn valmistumislukukausi ja -vuosi: Syksy 2023 Sivumäärä: 38

Tässä opinnäytetyössä toteutettiin laajamittainen päivitys Anicare-yrityksen mobiilisovellukseen. Tavoitteena oli suunnitella asiakaspalautteesta kokoelma uusia ominaisuuksia sekä muutoksia sovellukseen. Päivityksessä ensimmäisenä muutettiin porojen paikannukseen tarkoitettua Paikkatieto kysyttäessä -ominaisuutta. Ominaisuuteen lisättiin uusia vaihtoehtoja paikannukseen sekä mahdollisuus muokata ja peruuttaa jo pyydettyjä paikannuksia. Porolistasta rakennettiin uudenlainen käyttöliittymä, joka asetettiin karttanäkymän sivuun. Uudessa versiossa käyttäjät voivat jaotella omaa porolistaa muodostamalla siitä ryhmiä. Karttanäkymään lisättiin mahdollisuus valita poroja kartalta toimintoja varten. Valinnan lisäksi karttanäkymään lisättiin uudelleenkohdistustoiminto sekä porojen nimet. Pääsääntöisenä kehitysympäristönä käytettiin hybridikehitystä. Kehitysalustana toimi Ionicin, Angularin ja Capacitorin yhdistelmä.

Työn lopputulokseksi saatiin toimiva modulaarinen sivupaneeli sekä paikannuksen päivitetty käyttöliittymä. Sivupaneeliin on listattu tiiviisti poro- sekä paikannusryhmät. Karttanäkymään saatiin lisättyä porojen valitsemis- ja rajaustoiminto, porojen nimet sekä manuaalinen kohdistus.

Kaikkia yksityiskohtia ja tavoitteita ei ehditty viedä loppuun, mutta koska työn osa-alueet suunniteltiin huolellisesti ja rakennettiin pala palalta, jatkokehitys ja loppujen tavoitteiden toteuttaminen on tulevaisuudessa helpompaa. Esimerkiksi porojen ryhmiä ei pysty luomaan eikä muokkaamaan, mutta ryhmän perusrakenne lisättiin sovellukseen.

Jatkokehityksenä on tarkoitus viedä keskeneräiset työt loppuun sekä kehittää kaikkia listattuja ominaisuuksia vielä pidemmälle. Sivupaneelista voi rakentaa automaattisesti liikkuvan riippuen käyttäjän vuorovaikutuksesta. Paikkatieto kysyttäessä -ominaisuuteen paikannuksiin voidaan lisätä enemmän vaihtoehtoja, kuten uusia paikannustiheyksiä. Tulevaisuudessa on myös tavoitteena kehittää uudella ryhmätyypillä jaettujen porojen ryhmiä, joissa ryhmän poroja pystyvät tarkastelemaan toiset käyttäjät, joille ryhmä on jaettu.

Asiasanat: Web-kehittäminen, hybridisovellukset, mobiilisovellukset, Ionic, Angular, Capacitor

ABSTRACT

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

---

Author(s): Herkko Karjalainen
Title of thesis: Widescale development of hybrid application
Supervisor(s): Eino Niemi
Term and year when the thesis was submitted: Autumn 2023
Number of pages: 38

The aim of this thesis was to develop a widescale update to Anicare mobile application. The idea for the update came from customer feedback. The update consists of few changes and new features for the Anicare application.

Location request feature was changed to better match the use. New options for the requests were added. Users can now modify or cancel ongoing requests. A whole new user interface was developed for the reindeer list. Reindeer list now works as a side panel next to the map view. Users can now create custom groups from their reindeer to better organize and manage large reindeer lists. In addition to location request and reindeer list changes, the map view had some changes too. Users can select reindeer from the map using a selection box. The view can also be focused manually to reindeer. The application was developed with a hybrid mobile development framework. As for this, Ionic, Angular and Capacitor were used as the framework.

The result of the work was a functional modular reindeer list with a group framework and an updated user interface for location requests. However, not all tasks were accomplished but the update was planned so that making new updates in the future and doing the rest of the changes won't be as difficult of a task.

Keywords: Web development, hybrid application, mobile application, Ionic, Angular, Capacitor

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön aiheena on Anicare-yrityksen porojen paikannukseen ja hallintaan tarkoitetun mobiilisovelluksen uusi päivitys. Työn idea lähti asiakaspalautteesta, jossa kuultiin erilaisista tarpeista, käyttötavoista sekä toiveista seuraavaan versioon. Sen jälkeen palautteesta tehtiin kokoelma uusia ominaisuuksia sekä muutoksia, joista lopulta muodostui päivityksen sisältö.

Ongelman pystyi kiteyttämään yhteen lauseeseen. Sovelluksen toiminta ei vastannut tarpeeksi hyvin käyttötarkoitukseen. Porojen paikannuksessa oli puutteita sekä sovelluksesta puuttui hallintaan liittyviä ominaisuuksia ja toimintoja, jotka ovat hyvin olennaisia porotaloudessa. Toimintojen lisäksi sovelluksen rakenne oli monimutkainen, mikä johti siihen, että toiminnot eivät olleet tarpeeksi esillä.

Työssä suunnitellaan ensiksi palautteen pohjalta kokoelma uusia ominaisuuksia ja muutoksia. Suunnittelussa jokaisen muutoksen ja ominaisuuden kohdalla mietitään, miten ongelma ratkaistaan, mitä uusia ongelmia ratkaisuun sisältyy ja miten niiltä vältytään. Uuteen päivitykseen tehdään muutoksia Paikkatieto kysyttäessä -ominaisuuteen, joka vastaa käyttöliittymässä porojen paikantamisesta. Paikannukseen lisätään uusia vaihtoehtoja sekä poistetaan rajoitteita. Pyydetyille paikannuksille rakennetaan uusi käyttöliittymä, jossa niitä pystyy tarkastelemaan, muokkaamaan tai peruuttamaan. Myös sovelluksen vanhasta porolistasta muodostetaan parempi käyttöliittymä. Sovelluksesta löytyy karttanäkymä, jonka sivuun rakennetaan sivupaneelin tyylinen ikkuna, johon porolista ja paikannuspyynnöt asetetaan. Käyttäjälle annetaan mahdollisuus muodostaa omista poroista ryhmiä ja järjestää niitä haluamallaan tavalla. Porolistan ja paikannusominaisuuden ohella karttanäkymään lisätään sovelluksen käyttämistä ja porojen hallintaa helpottavia ominaisuuksia. Esimerkiksi poroja pystyy rajaamaan kartalta toimintoa varten. Rajatut porot valitaan automaattisesti ja niille voidaan toteuttaa jokin toiminto, kuten paikannus tai jako. Valinnan lisäksi porojen nimet näkyvät kartalla ja kartan voi kohdistaa uudestaan omiin poroihin.

Työ toteutetaan hybridisovelluksena, jossa päärakenteena on Ionic sekä Angularin ohjelmistokehys. Jotta työ voidaan kääntää mobiilisovellukseksi esimerkiksi Androidille, tarvitaan työssä myös Capacitor-työkalua sekä Androidin virallista kehitysalustaa Android Studiota. Työn ohjelmointikielenä käytetään pääsääntöisesti selainohjelmoinnin alalajia TypeScriptiä, mutta työssä tulee esiin myös perinteiset Web-ohjelmointikielet kuten JavaScript, HTML sekä CSS.

2 ANICARE OY JA PAIKANNUSLAITTEEN KEHITYSTARPEET

Anicare Oy on yritys, joka valmistaa seurantalaitteita poroille. Rudolf on pieni ja kevyt seurantalaite, joka asennetaan poron korvaan asennuspihdeillä samalla tavalla kuin tavallinen korvamerkki. Seurantalaite seuraa aktiivisesti poron sijaintia. Perustoiminnassa laite lähettää sijaintitietonsa joka sunnuntai, mutta uudessa Paikkatieto kysyttäessä -ominaisuudessa käyttäjä saa itse pyytää laitteiltansa sijaintitietoa. Laite myös ilmoittaa poron kuolemasta. Seurantalaitteen paristo perustilassa kestää jopa kymmenen vuotta, joten laitetta ei tarvitse vaihtaa poron elinaikana. (Anicare 2023a.)

Rudolf-laitteen paikannus auttaa poronomistajaa kartoittamaan omia poroja. Sijaintitieto porosta vähentää reilusti sitä aikaa, joka porojen löytämiseen tai jäljittämiseen menee. Sen lisäksi, että porojen seuranta ja löytäminen on nopeampaa, se on myös tehokkaampaa, sillä useaa poroa voidaan seurata yhtä aikaa mobiilisovelluksen avulla. Paikantamisen päälle porojen tunnistaminen on helpompaa ja nopeampaa, sillä jokainen laite kantaa uniikkia sarjanumeroa, jolla poro on lisätty sovellukseen. (Anicare 2023a.)

Seurantalaitteen lisäksi Anicare kehittää mobiilisovellusta, jolla omia poroja voidaan hallita. Porojen viikoittaiset sijaintitiedot tulevat Anicare-sovellukseen. Porojen sijaintitietojen lisäksi sovelluksessa voi esimerkiksi kirjoittaa lisätietoa porosta, tarkastella aikaisempia sijainteja tai jakaa omia poroja toisille Anicare-tileille. Omia paikannuspyyntöjä voi lähettää sovelluksen Paikkatieto kysyttäessä -ominaisuuden avulla. Ilmoitus poron kuolemasta tulee puhelimeen, kun sovellus on asennettuna. (Anicare 2023a.)

Anicaren tulevaisuuden tavoitteena on lisätä seurantalaitteeseen myös terveydentilan seurantaa. Anicaren Nano Track -laite on saman mallinen laite kuin Rudolf-seurantalaite, mutta sen lisäksi että laite ilmoittaa poron sijainnista ja kuolemasta, se myös seuraa eläimen liikettä ja lämpötilaa. Näistä kahdesta uudesta datasta muodostuu aktiivisuusdataa, jolla voidaan tulevaisuudessa saada tarkempaa tietoa eläimen terveydentilasta. Näin eläinten terveydentilan muutoksiin voidaan varautua paljon aikaisemmin. Porotalouden lisäksi laitetta on testattu myös muissa eläinryhmissä, kuten esimerkiksi karjataloudessa. (Anicare 2023b.)

3 TEKNOLOGIAT

Tässä luvussa käydään läpi työkaluja ja teknologioita, joita opinnäytetyössä on käytetty. Työkalut ja teknologiat ovat valittu valmiiksi työtä varten, sillä Anicare-sovellus on ollut kehityksessä ennen opinnäytetyön alkua. Sovellus on hybridimobiilisovellus, joka on kirjoitettu TypeScript-pohjaisena Ionic Framework -projektina. Sen pääsääntöisenä ohjelmistokehyksenä käytetään Angularia.

3.1 Ionic-ohjelmistokehityskokoelma

Ionic Framework on avoimen lähdekoodin kokoelma erilaisia hybridisovelluskehitystyökaluja. Sen kehittäjä Drifty Co julkaisi ensimmäisen Ionic-version vuonna 2013. Ionic on alun perin rakennettu AngularJS-selainkehitysalustan päälle, mutta tukee nykyään muita kehitysalustoja, kuten Reactia ja Vueta. Ionicilla pystyy kehittämään mobiilisovelluksia, työpöytäsovelluksia sekä PWA- eli Progressive Web App -sovelluksia. Ionic on erikoistunut hybridikehitykseen, joka tarkoittaa yksinkertaisesti selainkehityksellä tuotettuja sovelluksia, jotka on käännetty monelle muulle alustalle samasta koodista. Esimerkiksi tavallinen web-sivu voi toimia mobiilisovelluksena iOS- tai Android-käyttöjärjestelmissä. (Mittal 2023; Ionic 2023a.)

Ionic on koko projektia ympäröivä työkalu, joten uuden projektin luominen ja kääntäminen tapahtuu pääosin Ionic-komennoilla komentorivissä. Uuden projektin voi luoda käyttämällä ionic start -komentoa. Komentoon voidaan lisätä myös tabs-, sidemenu- tai blank-liite. Liitteet lisäävät uuteen projektiin valmiiksi jonkun ominaisuuden. Esimerkiksi sidemenu-malli lisää projektiin valmiiksi sivupaneelin, jota voidaan käyttää mahdollisesti navigointipaneelina. Projektin voi kääntää lopulliseen muotoon ionic build -komennolla. Projektin koodi käännetään ja asetetaan www-nimisen kansion sisälle. Sen jälkeen valmis sovellus voidaan julkaista omaan serveriin tai kääntää mobiilisovellukseksi. Projektin testaamista ja muokkaamista varten voidaan käyttää ionic serve -komentoa. Komento kääntää projektin ja luo paikalliseen verkkoon serverin, johon projekti asetetaan. Näin projektia voi käydä kokeilemassa selaimella. Kun testiserveri on käynnissä, projektiin voi tehdä muokkauksia samaan aikaan ja serveri päivittää muutokset automaattisesti. (Ionic 2023b; Ionic 2023c.)

3.2 Angular-ohjelmistokehys

Angular on TypeScript-pohjainen avoimen lähdekoodin selainkehitykseen tarkoitettu ohjelmistokehys. Google julkaisi alustan vuonna 2016. Angular muodostaa pääsääntöisesti selaimen rakenteen kehyksen. Angular on komponenttipohjainen käyttöliittymäkirjasto, jonka avulla monimutkaisista funktioista voidaan muodostaa itsenäisiä kokonaisuuksia eli komponentteja ja lopuksi yhdistellä niitä toisiinsa. Näin ohjelman rakenne pysyy helppolukuisena sekä modulaarisena. (Fireship 2020.)

Angularin kehys koostuu kolmesta pääominaisuudesta. Tärkein ominaisuus on komponenttirakenne, joka sisältää TypeScript-luokan, HTML-pohjan sekä CSS-tyylit. Komponentit sisältävät siis oman muotoilun sekä funktiot. Esimerkiksi sovelluksen eri sivut ovat omia komponentteja. Jokaisella sivulla on oma lista tyyleistä, HTML-rakenteesta sekä funktioista. Toinen Angularin ominaisuus on Template. Template-ominaisuus mahdollistaa komponentin TypeScript-luokan muuttujia käytettävän HTML-koodissa. Näin esimerkiksi sivun tekstejä voidaan vaihtaa dynaamisesti funktioiden avulla. Kolmas ominaisuuksista on Dependency injection. Dependency injection -ominaisuudella pystyy yhdistämään TypeScript-luokkia toisiinsa. Ominaisuus pääosin helpottaa koodin kirjoittamista. Isompia projekteja voidaan hajauttaa useampaan tiedostoon, jotta koodin luettavuus pysyy hyvänä. Sen lisäksi yhtä luokkaa voidaan hyödyntää monessa muussa komponentissa, joten samaa koodia ei tarvitse toistuvasti kirjoittaa. (Angular 2023.)

3.3 Capacitor-työkalu

Capacitor on Ionicin valmistama työkalu, joka mahdollistaa selainkehitysalustan kääntämisen natiiviksi koodiksi Androidille sekä iOS-alustoille. Capacitor julkaistiin vuonna 2018. Web-koodin kääntämisen lisäksi Capacitorilla on työkaluja, joilla pystyy hyödyntämään omassa web-kehityksessä laitteen ominaisuuksia, esimerkiksi kameraa tai puhelimen taskulamppua. (Capacitor 2023a; Lynch 2023.)

Hybridisovelluskehityksen kannalta Capacitor on yksi tärkeimmistä elementeistä, sillä sen avulla Web-projektin pystyy kääntämään sellaiseen muotoon, että projektin voi avata toisten käyttöjärjestelmien niiden natiiveissa kehitysalustoissa kuten Android Studiossa. Esimerkiksi projektin kääntäminen Androidille tapahtuu seuraavasti. Komennolla ionic capacitor add android Capacitor luo käännetystä Ionic-projektista Android-projektin, jonka voi avata Android Studiossa. Komento toimii

vain silloin, kun Ionic-projektista ei ole vielä tehty Android-projektia. Komennolla ionic capacitor sync android Ionic kääntää projektin ensiksi uudestaan ja Capactior synkronoi lopuksi kaikki muutokset Android-projektiin. (Capacitor 2023b.)

3.4 Visual Studio Code -tekstieditori

Visual Studio Code tai VS Code on Microsoftin tuottama avoimen lähdekoodin tekstieditori. Visual Studio Code on yksi yleisemmistä ja suosituimmista tekstieditoreista. Editori julkaistiin vuonna 2015. VS Codessa on monelle editorille tuttu syntaksinkorostus sekä virheenkorjausominaisuudet. Vähemmän tunnettuna ominaisuutena on sisäänrakennettu Git-versiohallinta. Sen lisäksi editori tukee yli sataa eri ohjelmointikieltä. Kaiken tämän lisäksi VS Codessa on laajennushakemisto, josta löytyy tuhansia yhteisön tuottamia laajennuksia ja lisäosia, joilla voi muokata Visual Studio Codesta juuri sellaisen editorin kuin itse haluaa. (The VS Code Team 2016; Visual Studio 2023.)

Sisäänrakennettu komentorivi helpottaa hybridisovelluskehitystä, sillä esimerkiksi Ionicin testaamiset ja kääntämiset tapahtuvat komentorivin kautta eikä erillistä komentorivi-ikkunaa tarvitse avata. Sen lisäksi uusien lisäosien asentaminen onnistuu helposti samasta komentorivistä. Visual Studio Coden tiedostohakemisto on tehokas tapa löytää projektin eri tiedostot. Näppäinyhdistelmällä Ctrl + P voidaan hakea kansion tiedostoja nimellä. Etenkin isommissa projekteissa nimellä hakeminen on erittäin tehokasta. Hakemiston lisäksi avattujen tiedostojen välilehdet pitävät tiedostot ladattuina, joten tiedostojen avaamista ja lataamista ei tarvitse jatkuvasti odottaa.

3.5 Android Studio -ohjelmointiympäristö

Android Studio on Googlen oma virallinen mobiilisovelluskehitykseen tarkoitettu ohjelmointiympäristö eli IDE (Integrated development environment), jolla pystyy kehittämään Android-käyttöjärjestelmille suunnattuja sovelluksia. Ohjelmointiympäristöstä löytyy projektihakemisto, tekstieditori, kääntäjä, konsoli sekä paljon muuta. Android Studio käyttää Gradle-pohjaista kääntäjää, joka luo lopullisen apk-päätteisen tiedoston. Studiosta löytyy myös monien Android-laitteiden emulaattoreita, jotka mahdollistavat työskentelyn pelkästään tietokoneella, mutta myös omia fyysisiä Android-laitteita pystyy hyödyntämään. (Android Developers 2023.)

Hybridisovelluskehityksessä ei suuremmin käytetä Android Studiota, sillä Android Studio keskittyy laajalti natiiviin eli Android-keskeiseen koodiin. Capacitor kuitenkin kääntää web-projektin Android-projektiksi, joten testaaminen mobiililaitteella ja lopullinen kääntäminen mobiilisovellukseen tapahtuu Android Studiossa. Testaamista laitteen emulaattorilla tai fyysisellä laitteella auttaa Android Studion oma konsoli, johon projektin lopulliset tulostukset kirjoitetaan. Testaamisen lisäksi Android Studiossa voidaan asettaa Manifest-tiedoston kautta sovelluksen yleistiedot, kuten nimi, versionumero tai kuvake.

4 TOTEUTUS

Tässä osiossa läpikäydään koko työn prosessi suunnitteluvaiheesta konkreettiseen toteutukseen asti. Suunnitteluvaiheessa tutkitaan tarkemmin asiakkaiden palautetta sekä pohditaan mahdollisia ratkaisuja sovelluksen ongelmakohtiin. Pohdinnassa kerrotaan myös, miksi ja miten kyseisiin muutoksiin tai ominaisuuksiin päädyttiin. Suunnitteluvaiheen jälkeen ominaisuudet ja muutokset lisätään sovellukseen.

4.1 Suunnittelu

Työn ensimmäinen vaihe oli tutkia yleistä asiakaspalautetta, jota sovelluksen aikaisemmasta julkaisusta lähtien on vastaanotettu. Palautteesta selvitettiin sovelluksen hyödyllisyyttä ja toimivuutta päivittäisessä käytössä. Onko porojen hallinta helpompaa ja tehokkaampaa sovelluksella? Mikä sovelluksessa on onnistunut? Mikä vaatii muutosta? Lopuksi asiakaspalautteen pohjalta muodostettiin lista sovelluksen ominaisuuksista ja heikkouksista, joita lähdetään parantamaan.

Tärkeimpänä tarpeena nähtiin porojen tiheämpi paikannus, sillä laitetta pitäisi pystyä paikantamaan useammin tietyllä aikavälillä. Sovelluksen aikaisemmassa versiossa laitetta pystyi parhaimmillaan paikantamaan manuaalisesti päivittäin, mutta tämä ei täysin vastannut käyttötarkoitusta. Porot voivat liikkua pitkiäkin matkoja päivässä, joten aamulla saatu paikkatieto porosta saattaa olla jo reilusti väärässä iltapäivällä. Liian pitkän paikannustiheyden lisäksi laitetta pystyi paikantamaan enintään viidesti kuukaudessa. Asiakaspalautteesta selvisi, että poroja paikannetaan paljon enemmän ja tiheämmin tiettyinä aikoina vuodessa. Paikannuksia on eniten pääsääntöisesti keväällä ja syksyllä. Jos poroa ei ollut tarkoitus jäljittää tai seurata, laitteen viikoittainen tai päivittäinen paikannus oli tarpeeksi. Näin ollen viisi käyttäjän asettamaa paikannusta kuukauteen sekä liian pitkä paikannustiheys eivät täysin vastaa käyttäjän tarpeisiin.

Paikannuksen lisäksi sovelluksen käyttöliittymä vaati muutosta. Aiemmassa sovellusversiossa oli paljon hyviä ominaisuuksia, mutta niiden käyttäminen oli monimutkaista ja epäkäytännöllistä, kuten esimerkiksi aiemmin mainittu Paikkatieto kysyttäessä -ominaisuus. Sovelluksessa porosta pystyi pyytämään paikkatietoa ensiksi menemällä sovelluksen hakemistosta porolistan sivulle. Porolis-

tasta täytyi yksitellen valita jokainen poro, josta halusi paikkatietoa. Sitten poroille valittiin paikkatietotoiminto ja sen jälkeen käyttäjä pystyi asettamaan haluamansa päivän paikannukselle. Käyttäjän täytyi suorittaa kolmesta neljään eri vaihetta, ennen kuin hän pääsi asettamaan pelkästään paikannusta. Vaiheiden määrä kasvaa vielä enemmän riippuen käyttäjän tilanteesta. Esimerkiksi jos käyttäjä ei tiedä paikannettavien porojen nimiä, mutta tietää niiden sijainnit. Hänen täytyy ensiksi yksitellen selvittää jokaisen poron nimi kartalta tai listasta, että hän voisi pyytää juuri niiden porojen paikkatietoa. Ongelma esiintyi myös muissa toiminnoissa, kuten porojen jakamisessa tai tarkastelussa. Käyttöliittymän rakenne aiheutti sekavuutta ja monimutkaisuutta sovellukseen.

Uudessa päivityksessä pitää toteuttaa kattava muutos Paikkatieto kysyttäessä -ominaisuuteen sekä muuttaa sovelluksen rakennetta siten, että sitä on helpompaa ja tehokkaampaa käyttää. Erilaisten toimintojen pitää olla helposti tavoitettavissa sekä niitä pitää pystyä lähestymään riippumatta käyttäjän tilanteesta.

Miten Paikkatieto kysyttäessä -ominaisuutta sitten muutetaan? Tutkitaan ensiksi helpommin ratkaistavia ongelmia, kuten paikannusrajaa. Paikannuksen käytöstä ei ole tarpeeksi tietoa kuvaten esimerkiksi sitä, milloin, miten ja kuinka paljon paikannusta käytetään tietyin aikavälein. Silloin paikannusrajaa on vaikea asettaa mihinkään ajanjaksoon tai määrään luotettavasti. Siispä rajoite on järkevin ottaa pois käytöstä, jolloin käyttäjä voi paikantaa poroja rajattomasti milloin tahansa.

Koska poroa pitäisi pysytä jäljittämään tiheämmin, uudessa paikannusominaisuudessa laitetta pystyy paikantamaan jopa viidentoista minuutin paikannustiheydellä eli joka viidestoista minuutti laite lähettää sijaintinsa. Ongelmaksi syntyy tilanne, jossa käyttäjä asettaa tiheän paikannuksen ja mahdollisesti unohtaa paikannuksen päälle tai jollain muulla tavalla erittäin tiheä paikannus jää päälle. Tiheä paikannus ilman rajoituksia moninkertaistaa virrankulutuksen, joten laite ei kestä niin pitkään kuin sen pitäisi. Ylimääräisiä paikannuksia pitää vältellä mahdollisimman paljon, ja siksi jokaiseen paikannukseen pitää asettaa pakollinen lopetusaika. Käyttäjälle annetaan lopetusajan lisäksi myös mahdollisuus asettaa aloitusaika ja siten kaventaa paikannusta vielä enemmän.

Käyttäjän itse asettamalla päivämäärällä, aloitusajalla ja lopetusajalla annetaan jo hyvät mahdollisuudet vaikuttaa paikannusten määrään. Käyttäjälle annetaan myös vaihtoehtoja erimittaisiin paikannustiheyksiin. Voi olla tilanteita, joissa käyttäjän täytyy tietää poron sijainti koko päivältä, mutta hän ei tarvitse siltä äärimmäisen tarkkaa sijaintitietoa. Näin kahden tunnin paikannustiheys voisi

olla paljon parempi. Käyttäjälle annetaan 15 minuutin, puolen tunnin, tunnin sekä kahden tunnin vaihtoehdot paikannuksen tiheydelle.

Jo pyydettyjä paikannuksia pystyi tarkastelemaan aikaisemmassa versiossa. Ominaisuus oli hyvin yksinkertainen. Se ei ollut selvästi esillä eikä siihen pystynyt lisäämään uusia ominaisuuksia helposti. Uuteen versioon lisätään mahdollisuus peruuttaa paikannuspyynnöt ja tarvittaessa muokata pyyntöjä välttyäkseen turhilta paikannuksilta. Siksi paikannuksen uusien asetusten lisäksi käyttöliittymään pitää rakentaa uusi näkymä jo lähetetyille paikannuspyynnöille. Näkymästä käyttäjä voi tarkastella, muokata tai poistaa pyyntöjä tarvittaessa.

Sovelluksen yksinkertaistamista ja päivityksien lisäämistä lähestyttiin ensisijaisesti karttanäkymän kautta. Asiakkaan näkökulmasta porojen tarkastelu ja hallitseminen suoraan kartalta oli paljon luontevampaa kuin pelkästään listanäkymästä. Laite on pääsääntöisesti paikannuslaite, joten voidaan olettaa, että käyttäjä on karttanäkymässä suurimman osan ajasta, kun hän käyttää sovellusta. Siksi juuri karttaan on hyvä lisätä samat toiminnot, jotka löytyvät listanäkymästä, kuten porojen valitseminen, paikantaminen ja jakaminen.

Vaikka karttaan lisätyt toiminnot parantavat sovelluksen käyttöä, porot ovat edelleen yksi iso järjestyksetön lista. Porolistaan ehdotettiin järjestelyä aakkosten mukaan, lisäyspäivän mukaan tai viimeisen paikkatiedon mukaan, mutta yksikään näistä ei vaikuttanut käyttöön merkittävästi. Poroja kuitenkin jaetaan ryhmiin ja luokkiin käyttäjän oman hallinnan ja toimintatavan mukaan, joten on selvää, että käyttöliittymä tarvitsee tavan luoda poroista omia ryhmiä. Käyttäjä pystyisi valitsemaan porot, joista haluaa muodostaa oman ryhmänsä. Koska ryhmiä pitäisi pystyä myös lisäämään, poistamaan, tarkastelemaan sekä muokkaamaan, täytyy ryhmistä muodostaa oma listanäkymä sovellukseen.

Pororyhmistä sekä paikannuspyynnöistä piti siis luoda uudet käyttöliittymät. Niiden tuli olla selkeästi esillä ja olla helppokäyttöisiä. Jos molemmista ominaisuuksista luodaan erilliset sivut, tämä saattaisi lisätä sekavuutta ja monimutkaisuutta sovellukseen. Käyttäjän täytyisi hyppiä porolistan, paikannusten ja kartan välillä jatkuvasti, mikä ei ole hyvä asia.

Pororyhmissä ja paikannuspyynnöissä on paljon samankaltaisuuksia. Molemmissa on jokin tunniste tai nimi sekä lista poroista. Ryhmät poikkeavat toisistaan vain yhdellä tavalla. Pyynnöt ovat

väliaikaisia ryhmiä, kun pororyhmät ovat puolestaan pysyviä ryhmiä. Koska pororyhmät sekä paikannuspyynnöt ovat hyvin samankaltaisia, ne voidaan loogisesti asettaa samaan listanäkymään. Näin ei tarvitse rakentaa kahta erillistä käyttöliittymää.

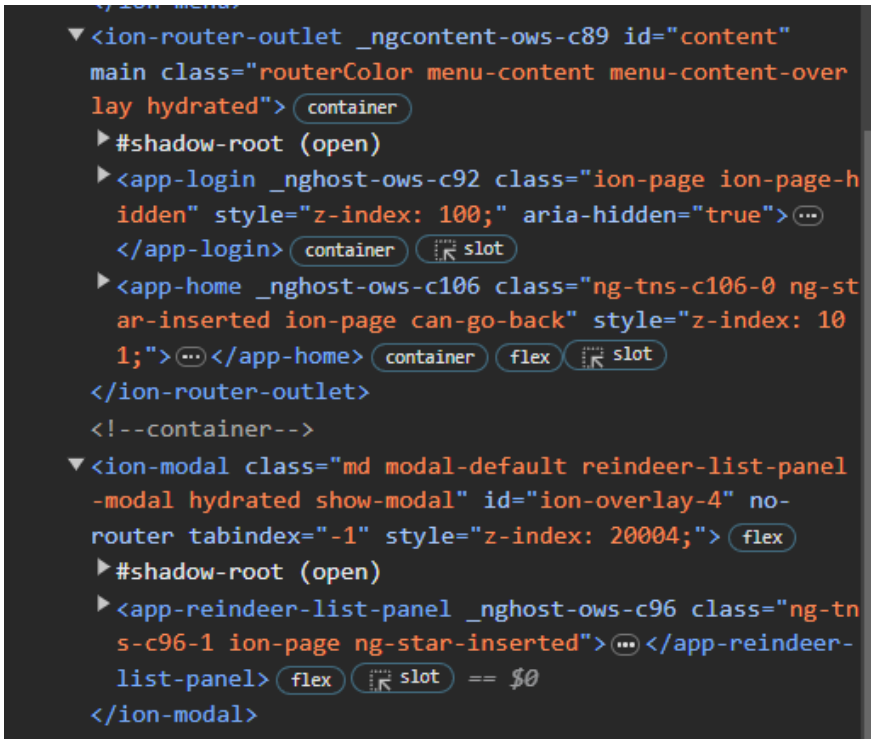
Samalla tavalla kuin porojen toiminnot voidaan lisätä karttanäkymään, voidaan listanäkymäkin lisätä karttaan. Poro- sekä pyyntöryhmät voitiin asettaa karttanäkymän kylkeen liikkuvan sivupaneelin tavalla. Sivupaneelin ja päänäkymän välinen vuorovaikutus sovelluksessa on hyvin yleistä muissa sovelluksissa. Sivupaneeliin lisättäisiin samat toiminnot kuin aiemmassa porolistassa, kuten paikannus tai jakaminen, mutta aiemman listan lisäksi se tukee myös ryhmiä.

Karttaan lisätyt toiminnot, poro- sekä paikannusryhmät ja sivupaneeli tukevat vahvasti sovelluksen helppokäyttöisyyttä sekä tehokkuutta. Sovelluksen vanhat toiminnot sekä porolista ovat nyt paljon lähempänä karttanäkymää, jota käyttäjä pääsääntöisesti käyttää. Päivityksen kokonaisuus ratkaisee myös ongelman porojen valitsemisessa. Aikaisemmin porot täytyi yksitellen poimia tai poissulkea valinnasta. Helppoa osittaista valintaa ei pystynyt toteuttamaan. Uusilla kartan rajaustoiminnoilla ja pororyhmillä osittainen valinta on nyt mahdollista. Käyttäjä pystyy rajaamaan kartalta alueen missä halutut porot sijaitsevat tai valita poroja ryhmittäin porolistasta. Poroja voi myös valita molemmilla tavoilla yhtä aikaa. Jos käyttäjä tietää porojen sijainnit, voidaan ne valita kartalta. Jos käyttäjä tietää porot nimeltä tai ne kuuluvat tiettyyn ryhmään, voidaan ne valita sivupaneelista.

4.2 Sivupaneeli

Sivupaneelin rakentaminen lähti ensin päättämällä, millä komponentilla paneeli luodaan. Sovelluksesta löytyi valmiiksi sivupaneeli, jota käytettiin sovelluksen navigointipaneelina. Paneeli oli rakennettu Ionicin omalla ion-menu-komponentilla. Paneelista pääsi sovelluksen muille sivuille, kuten tilin asetuksiin tai porolistaan. Työssä ei päädytty kuitenkaan käyttämään tätä komponenttia, sillä sen muokkaaminen oli rajallista eikä se tukenut ominaisuuksia, joita sivupaneeliin haluttiin lisätä.

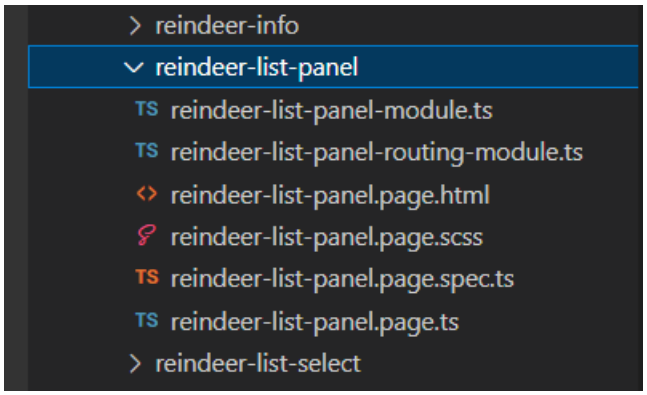
Sivupaneelin pohjana päädyttiin käyttämään ion-modal-komponenttia. Ion-modal on Ionicin oma komponentti, joka toimii hyvin samalla tavalla kuin sovelluksen sivukomponentit. Kun modaalikomponentti luodaan, se ei asetu sivujen kanssa saman HTML-elementin sisälle, vaan modaalikomponenteille on oma paikkansa, johon kaikki modaalit asettuvat (kuva 1). Tämä mahdollistaa sen, että oli käyttäjä millä sivulla tahansa, modaalikomponentit ovat aina sivun sisällön päällä.



KUVA 1. Sovelluksen HTML-rakenne, jossa sivujen ja modaalien sijainnit.

Modaalikomponentin pystyy luomaan kahdella eri tavalla. Komponentin pystyy kirjoittamaan suoraan pääsivun HTML-koodiin. Tätä kutsutaan Inline-modaaliksi. Komponentin voi myös kirjoittaa erillisenä tiedostona, jota kutsutaan erikseen funktion kautta. Inline-modaalin huonona puolena tässä tapauksessa on se, että koko sivupaneelin koodi pitäisi kirjoittaa pääsivun koodin sekaan. Jos varaudutaan tuleviin muutoksiin ja pidetään koodi helppolukuisena, on modaalikomponentti järkevää rakentaa erillisenä pääsivusta.

Sivupaneelista rakennetaan oma Angular-komponentti eli erillinen kansio, johon lisätään sivupaneelin rakenne, funktiot sekä tyylit (kuva 2). Jotta sivupaneelin voi avata kotisivulla, täytyy modaalikomponentin luomisfunktio asettaa sinne (kuva 3). Funktion lisäksi kotisivun HTML-rakenteeseen lisätään nappi, joka kutsuu modaalikomponentin funktiota (kuva 4). Nyt sivupaneeli avautuu nappia painamalla ja tuhoutuu, kun tummennettua taustaa painetaan.



KUVA 2. Sivupaneelin kansio, jonka sisällä paneelin rakenne, tyylit ja toiminnallisuus.

```
//   this.waitingPanel = false;
// } else {
  this.reindeerPanel = await this.modalCtrl.create(
    {
      component: ReindeerListPanelPage,
      componentProps: {},
      cssClass: 'reindeer-list-panel-modal',
      backdropDismiss: false
    });
  StaticVariablesService.isReindeerPanel = false;
  await this.reindeerPanel.present();

  // this.waitingPanel = false;
```

KUVA 3. Sivupaneelin luomisfunktio karttanäkymän TypeScript-tiedostossa.

```
<div
  [@fadeInOut]="!this.panelLoaded ? 'closed' : 'open'"
  [hidden]="!this.panelLoaded"
  id="sidepanelbutton"
  class="sidepanelButton"
  [ngClass]="{'sidepanelButtonOn': this.isPanelOn}"
  (click)="showReindeerListPanel()">
  <img src="assets/icon/sidepanel_button_1.svg">
  <ion-icon
    [@rotateFadeInOut]
    *ngIf="!this.isPanelOn"
    name="chevron-forward-outline">
  </ion-icon>
  <ion-icon
    [@rotateFadeInOut]
    *ngIf="this.isPanelOn"
    name="chevron-back-outline">
  </ion-icon>
</div>
```

KUVA 4. Sivupaneelin avausnappi karttanäkymän HTML-koodissa.

Normaalisti päällimmäinen modaalikomponentti täytyy tuhota, ennen kuin alempaa sivua pystyy käsittelemään. Tarkoituksena olisi, että käyttäjä pystyisi ohjaamaan karttaa sekä porolistaa yhtä aikaa. Lisäämällä backdropDismiss-asetuksen ion-modal-komponentin määrittelyihin sekä piilottamalla tummennetun taustan voidaan alla näkyvää sivua sekä sivupaneelia ohjata yhtä aikaa. BackdropDismiss-asetus on toiminto, jonka avulla käyttäjä tuhoaa modaalikomponentin painamalla tummennettua taustaa eli backdroppia. Tausta täytyy piilottaa, jotta käyttäjän syöttö osuu alla olevaan sivuun eikä tummennettuun taustaan. Sivupaneeli luodaan karttanäkymän päälle ja asetetaan se siten, että alempaa sivua pystyy käsittelemään samalla.

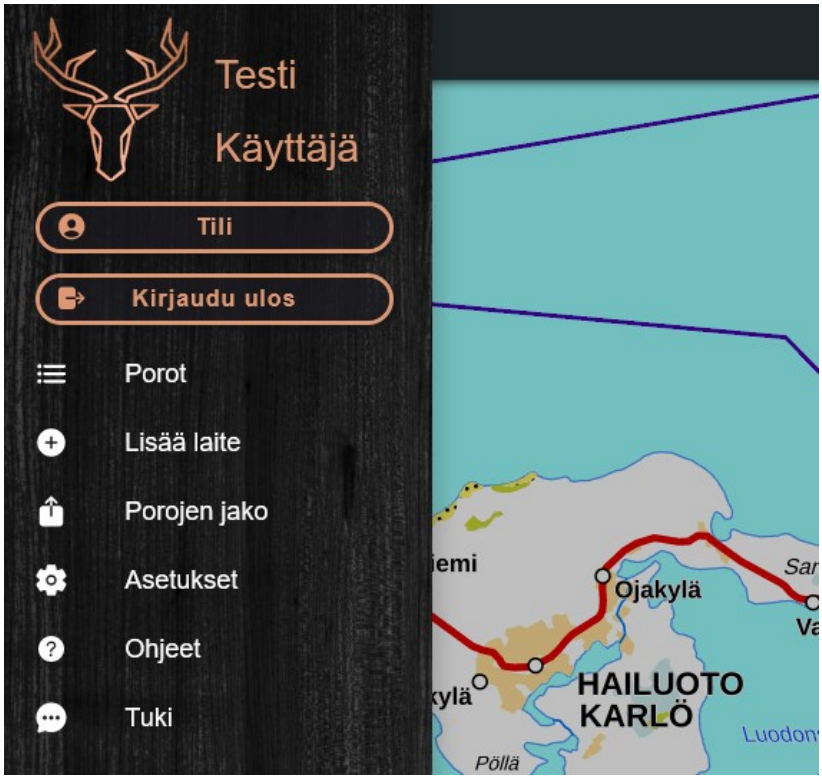
Koska sivupaneelin tuhoaminen estetään, täytyy paneeli piilottaa eri tavalla. Alun perin modaalikomponentti tuhotaan ja luodaan uudestaan avaamisessa ja sulkemisessa. Kun sivupaneeli tuhotaan, kaikki porovalinnat, ryhmien avaukset tai näkymän rullaus palautuvat. Toisin sanoen sivupaneeli latautuu uudelleen ja käyttäjän tekemät muutokset tai toiminnot poistetaan. Koska käyttäjän pitää pystyä ohjaamaan sivupaneelia ja karttaa yhtä aikaa sekä tekemään valintoja kartalta ja porolistasta, on luonnollisempaa, että sivupaneelin tila pysyy samana, kun se asetetaan piiloon. Sivupaneelin piilottaminen päädyttiin rakentamaan seuraavasti. Sivupaneeli luodaan vain kerran karttanäkymän avautuessa. Sen sijaan että sivupaneeli poistetaan sulkunapista, paneeli liu'utetaan ruudun ulkopuolelle animaatioiden avulla. Paneelin tila saadaan pysymään eikä paneeli ole karttanäkymän tiellä. Sivupaneeli kuitenkin tuhotaan joka kerta, kun karttanäkymästä poistutaan toiselle sivulle, jotta karttanäkymään palatessa ylimääräisiä sivupaneeleja ei synny.

Nyt sivupaneelin lisäämisen jälkeen sovelluksessa on päällekkäin navigointipaneeli sekä porolistapaneeli. Navigointipaneelista voidaan luopua kokonaan, kun sen toiminnot saadaan mahtumaan uuteen sivupaneeliin. Vanha lista navigointeja on turhan pitkä uuteen paneeliin, joten sitä pitää lyhentää. Listasta voidaan poistaa porolistanavigointi, sillä porolista löytyy suoraan sivupaneelista. Kun vertaa muita sovelluksia, joissa on useita navigointeja, toimintoja ja sivuja, huomaa, että monesti tilin profiilikuvakkeen alle sijoitetaan navigoinnit ponnahdusvalikkona. Ponnahdusvalikko on yleensä pieni erillinen ikkuna, joka sisältää listan toimintoja, jotka halutaan tiivistää yhden napin tai toiminnon alle. Käyttöliittymässä halutaan kuitenkin pitää suurin osa navigoinneista näkyvillä helpokäyttöisyyden vuoksi, joten vain tilin asetukset ja uloskirjautuminen asetettiin ponnahdusikkunaan. Ponnahdusikkunaan käytetään Ionicin komponenttia nimeltä ion-popover. Ion-popover on hyvin samankaltainen kuin ion-modal-komponentti, mutta se sisältää ponnahdusvalikkoon olennaisia ominaisuuksia. Koska kyseinen ponnahdusvalikko on hyvin pieni eikä sitä tarvitse avata muualla kuin sivupaneelissa, on ponnahdusvalikko järkevää rakentaa inline-komponenttina (kuva 5).

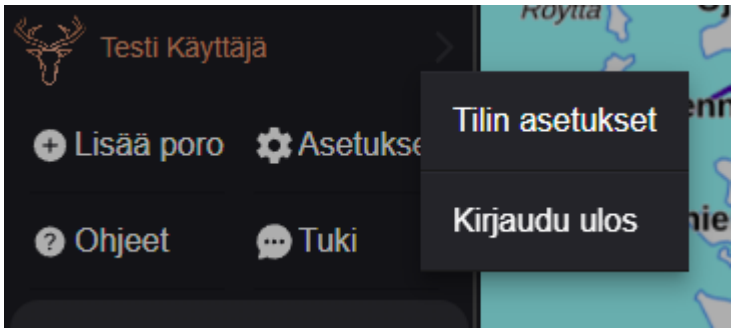
Lopuksi kaventamalla fonttikokoa ja kuvakkeita sekä asettamalla sivut kahteen pylvääseen saadaan hakemistosta hyvin pieni. (Kuva 6; kuva 7.)

```
<ion-popover #accountPopover [dismissOnSelect]="true" triggerAction="click"
class="popover-account-options" [isOpen]="isAccountOptionsOpen"
(didDismiss)="isAccountOptionsOpen = false">
  <!-- <ng-template> -->
    <!-- <ion-content class="ion-padding">Hello World!</ion-content> -->
  <!-- </ng-template> -->
  <ng-template style="width: 150px">
    <ion-item (click)="this.navigateToPage('/account-settings');
    checkAll(false)" class="reindeerOptionItem">
      <ion-label>Tilin asetukset</ion-label>
    </ion-item>
    <ion-item (click)="this.dismiss(); this.generalFunctions.logout(true);
    checkAll(false)" class="reindeerOptionItem">
      <ion-label>Kirjaudu ulos</ion-label>
    </ion-item>
  </ng-template>
</ion-popover>
```

KUVA 5. Ponnahdusvalikkokomponentti inline-tyylillä sivupaneelin HTML-koodissa

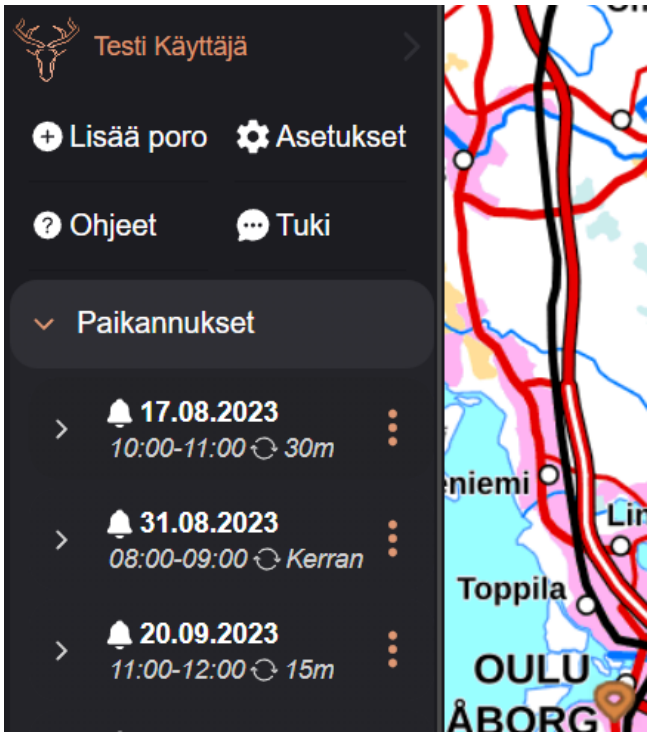


KUVA 6. Sovelluksen vanha navigointipaneeli.

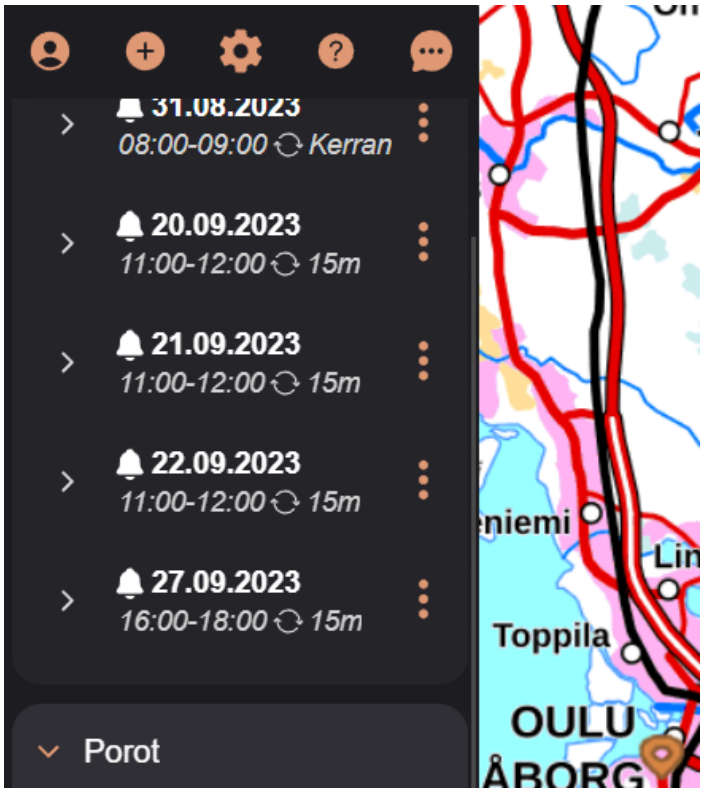


KUVA 7. Sovelluksen uusi navigointipaneeli, jossa avattu ponnahdusvalikko.

Navigointipaneeli uudessa käyttöliittymässä on silti hiukan pitkä, jos käyttäjän tavoitteena on vain tarkastella poroja. Ratkaisuksi käytettiin yhdenlaista keinoa, jota ilmenee myös monessa muussa sovelluksessa. Navigointilistasta muodostetaan kaksi erilaista näkymää. Ensimmäinen on laajennettu näkymä, jossa toimintojen nimet ja tekstit näkyvät. Toinen on tiivistetty näkymä, jossa vain toimintojen kuvakkeet näkyvät. Näitä kahta näkymää vaihdellaan riippuen siitä, mitä käyttäjä tekee sillä hetkellä. Käyttäjän tekeminen määritellään siten, miten käyttäjä on rullannut listanäkymää. Esimerkiksi jos käyttäjä on listan kärjessä eikä ole rullannut listaa alaspäin ollenkaan, voidaan olettaa, että käyttäjä haluaa navigoida toiselle sivulle. Silloin navigointipaneelista näytetään laajennettu näkymä. Jos käyttäjä on rullannut listaa alemmas, voidaan olettaa, että käyttäjä haluaa tarkastella poroja tai paikannuksia. Silloin näytetään tiivistetty navigointipaneeli. (Kuva 8; kuva 9.)



KUVA 8. Navigointipaneelin laajennettu näkymä sivupaneelissa.



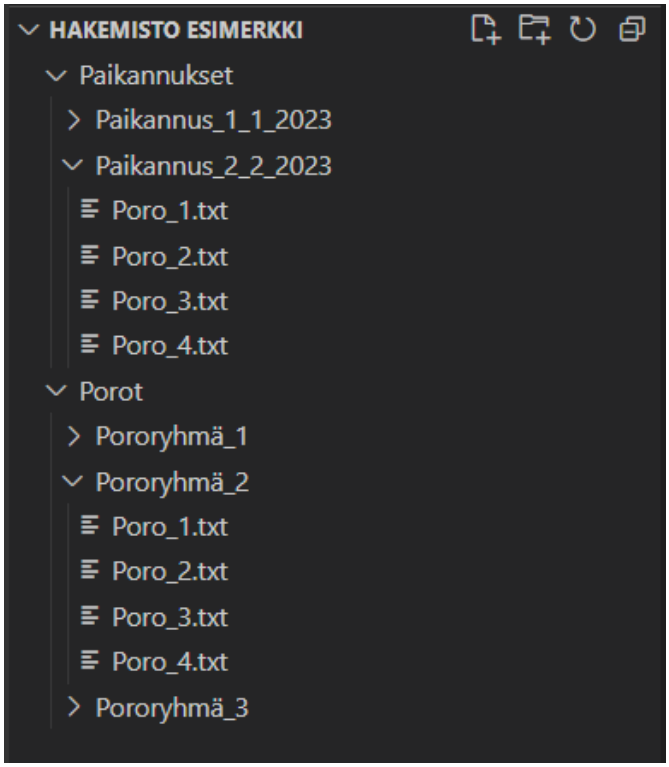
KUVA 9. Navigointipaneelin tiivistetty näkymä sivupaneelissa.

4.3 Poro- ja pyyntöryhmät

Porot, pororyhmät sekä paikannuspyynnöt täytyy lisätä uuteen sivupaneeliin. Ensiksi täytyy selvittää, miten listat rakenteellisesti asetetaan paneeliin. Listaan täytyy saada mahtumaan paikannuspyynnöt sekä pororyhmät ja niiden tulee erottua selvästi toisistaan. Ryhmien ja paikannuspyyntöjen lisäksi niiden sisältämiä poroja pitäisi pystyä tarkastelemaan. Jokaisella osalla tulisi olla myös lista tarvittavia toimintoja. Esimerkiksi paikannuspyynnöissä sekä pororyhmissä pitäisi olla muokkaus- ja poistotoiminnot ja poroilla pitäisi olla jakamis- ja paikantamistoiminnot.

Rakenteen tulisi toimia seuraavasti. Ylimpänä rakenteessa ovat paikannuspyynnöt sekä porot. Paikannuspyyntöjen sisälle kuuluu kaikki lähetetyt paikannuspyynnöt. Paikannuspyynnön sisällä on lista poroista, jotka kuuluvat siihen paikannukseen. Poro-osioon kuuluu kaikki käyttäjän omistamat porot sekä luodut pororyhmät. Pororyhmiin kuuluu kaikki siihen ryhmään kuuluvat porot. (Kuva 10.) Kun yllä kuvaillun rakenteen kaventaa tiiviiseen sivupaneeliin, saadaan hakemistoa tai sisällysluetteloa muistuttava tietorakenne. Ryhmät ovat kuin kansioita, joiden sisällä on tiedostoja eli tässä tapauksessa poroja. Hakemiston tyylinen tietorakenne antaa myös mahdollisuuden piilottaa ryhmän sisällön, jolloin lista pysyy tiiviinä ja siistinä. (Kuva 11.)

KUVA 10. Hahmotelma pororyhmien ja paikannuksien rakenteesta.



KUVA 11. Esimerkki hakemiston tyylisestä poro- ja paikannuslistasta.

Miten ryhmät sitten luodaan? Ionicilta löytyy ion-accordion niminen komponentti. Se on komponentti, jolla on otsikko ja sisältö. Annetun sisällön pystyy piilottamaan painamalla otsikon elementtiä. Ion-accordion-komponentilla pystyy rakentamaan hakemiston tietorakenteen ja sen etuna on

myös avaamis- ja sulkemistoiminnot sekä valmiit animaatiot. Suunnitellussa rakenteessa on pääosin kaksi sisäkkäistä avattavaa sisältöä, joten ion-accordion-komponentteja täytyy asettaa kaksi sisäkkäin. Esimerkiksi porot-osio muodostaa yhden ion-accordion-komponentin, jonka sisältö on lista ion-accordion-komponentteja, joiden sisällöt ovat listoja sen ryhmän poroista (kuva 10).

Rakenteen viimeisin osa on poro itse. Koska poroja tulee pystyä valitsemaan, täytyy valitsemista varten tehdä oma toiminto. Siksi porojen näyttämiseen käytetään ion-checkbox-nimistä komponenttia. Komponenttiin kuuluu otsikko sekä valintaruutu tai kuvake, joka ilmaisee, onko kyseinen poro valittu.

Sekä pororyhmistä että paikannuspyynnöistä muodostetaan omat luokat koodissa. Luokkaan sisällytetään sille ryhmälle ominaiset tunnisteet sekä lista sen ryhmän poroista. Poroista löytyy jo valmis luokka nimeltä IReindeer, joka sisältää poron tarvittavat tiedot, kuten nimen tai sijainnin (kuva 12). Esimerkiksi paikannuspyyntöryhmässä on identifiointinumero, paikannuksen tila, porolista, paikannustiheys, aloitusaika ja lopetusaika (kuva 13). Sivupaneelin koodissa luodaan listatyyppinen muuttuja luoduista luokista (kuva 14). Sen jälkeen lista asetetaan HTML-koodiin käyttäen ngFor-määritettä. NgFor on Angularin toiminto, joka toimii samalla tavalla kuin for-silmukka, mutta HTML-ympäristössä. Kun ngFor-määrite asetetaan esimerkiksi div-elementtiin ja annetaan sille viittaus listaan, ngFor luo niin monta div-elementtiä kuin alkioita on annetussa listassa. Paikannuspyyntöjen kohdalla ngFor asetetaan HTML-koodissa ylimpään elementtiin, joka sisältää yhdelle paikannuspyyntöryhmälle kaikki tarvittavat elementit, kuten ion-accordion-komponentti, otsikko tai kuvake. Listaksi annetaan aiemmin luotu lista paikannuspyyntöryhmistä. Sen lisäksi annetaan jokaiselle alkiolle requestGroup-tunniste, jotta jokaisen yksittäisen alkion muuttujia voidaan erikseen kutsua. Esimerkiksi requestGroup.reindeer palauttaa paikannuspyyntöryhmän porolistan. Pororyhmät toteutetaan samalla tavalla käyttäen sisäkkäisiä ngFor-määritteitä. (Kuva 15.)

```
export interface IReindeer {
  serialnumber: string;
  reindeerId: string;
  status: string;
  battery: number;
  lat: any;
  long: any;
  name: string;
```

KUVA 12. Interface-tyyppinen luokka poroista.

```
125
126   export interface IRequestGroup {
127     // received from API
128     id: number;
129     status: string;
130     devices: string[];
131     frequency: string;
132     first_req: string;
133     last_req: string;
134
135     // for visuals
136     date_formatted: string;
137     hourStart: string;
138     hourStop: string;
139     freq_formatted: string;
140     reindeer: IReindeer[];
141
```

KUVA 13. Interface-tyyppinen luokka paikannuksesta.

```
85
86     public requestGroups: IRequestGroup[] = [];
87
```

KUVA 14. Lista paikannusolioista sivupaneelin TypeScript-koodissa.

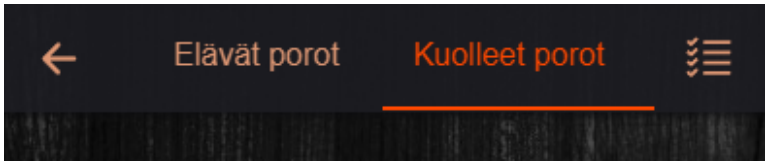
```
<ion-accordion class="requestList" style="border-radius: 15px; margin-top: 5px; margin-bottom:
  <ion-item slot="header" class="requestHeader">
    <ion-label style="margin-left:20px;" >Paikannukset</ion-label>
  </ion-item>
  <div class="requestListContent contentMargin" slot="content">

    <div class="noItemsDiv" *ngIf="this.requestGroups.length == 0">
      <ion-label>Sinulla ei ole yhtään paikannusta</ion-label>
    </div>

    <ion-accordion-group multiple="true">
      <div class="reindeerGroupWrapper" *ngFor="let requestGroup of requestGroups">
        <ion-button icon-only class="requestActionButton functionButtons" style="width: 40px; h
          <ion-icon class="reindeerImg" name="ellipsis-vertical" style="margin: 0px; position:
        </ion-button>
        <ion-accordion class="reindeerGroup" id="requestAccordion">
          <ion-item slot="header" class="requestGroupHeader">
            <ion-icon class="icon-request" mode="md" *ngIf="requestGroup.status =='pending'" name
            <ion-icon class="icon-request" mode="md" *ngIf="requestGroup.status =='today'" style=
            <ion-icon class="icon-request" mode="md" *ngIf="requestGroup.status =='running'" styl
            <div class="requestDateChip">
              <!-- <ion-icon name="calendar" style="font-size: medium; margin-right: 2px;"></ion-
              <ion-label>{{requestGroup.date_formatted}}</ion-label>
            </div>
```

KUVA 15. Paikannuslistan HTML-rakenne, jossa alkiot lisätään ngFor-määritteellä.

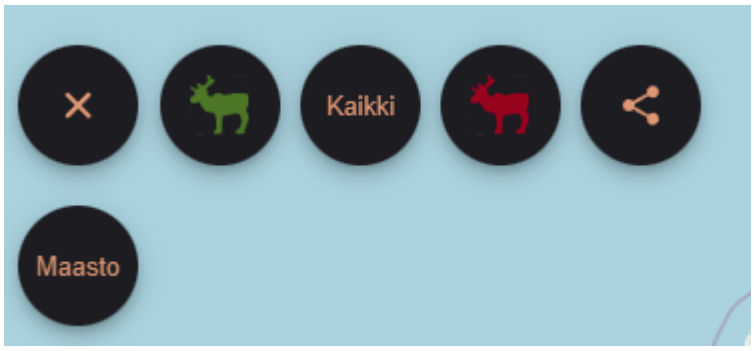
Paikannuspyyntöryhmät muotoillaan seuraavasti. Paikannuksen tilaa varten asetetaan kuvake, joka ilmoittaa värityksellään, onko paikannus tulossa, tänään käynnistyvä tai käynnistynyt. Kuvakkeen lisäksi aloitusajan ja lopetusajan voi kuvata päivämääränä sekä aikavälinä. Lopuksi lisätään paikannustiheys kavennettuna lyhenteisiin, kuten 15 min tai 1 h. Koska paikannuksia pitää pystyä muokkaamaan tai peruuttamaan, asetetaan paikannusalkion oikeaan laitaan kolmen pisteen kuvake. Kolmen pisteen kuvake yleensä ilmaisee painiketta, josta saadaan esiin enemmän toimintoja tai lisäasetuksia. Kaikki paikannuksen toiminnot asetetaan tämän napin alle ponnahdusikkunan tyyliin tilan puutteen vuoksi. Paikannusten porot eivät tarvitse erillisiä toimintoja, joten vain niiden nimet ja kuvakkeet näkyvät.

Pororyhmät muotoillaan hyvin samalla tavalla, mutta pienillä muutoksilla. Ryhmät ovat käyttäjän itse luomia, joten niitä pitäisi pystyä muokkaamaan monella eri tavalla. Ryhmille annetaan kuvake sekä nimi, mutta käyttäjä voi itse vaihtaa ne tulevaisuudessa. Poikkeavana toimintona on ryhmän valitseminen, joten ryhmä tarvitsee valintanapin. Valintanapista kaikki ryhmän porot valitaan kerralla. Valinnan lisäksi ryhmiä pitää pystyä muokkaamaan, joten ryhmät tarvitsevat myös kolmen pisteen kuvakkeen, johon kaikki toiminnot sisällytetään. Pororyhmän poroilla on yksittäisiä toimintoja, kuten poron poistaminen tai lisätietojen tarkasteleminen, joten porot tarvitsevat myös kolmen pisteen kuvakkeen. Valintakuvake poroille saadaan suoraan aiemmin mainitusta ion-checkbox-komponentista. Muuten poron alkioon tulee samat tiedot, mitä aikaisemmassa sovellusversiossa oli. Näitä ovat nimi, sarjanumero, poro- ja paristokuvake.

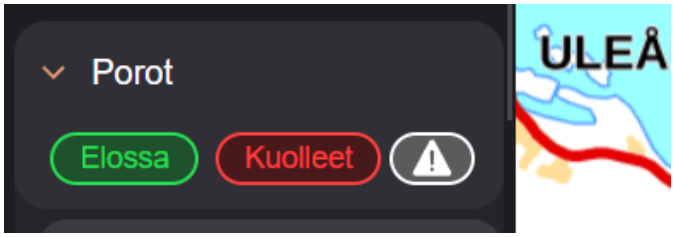
Aiemmassa versiossa porot ovat jaettu kahteen pääryhmään: eläviin ja kuolleisiin poroihin (kuva 16). Karttanäkymässä pystyi suodattamaan poroja ja näyttämään vain ne porot, jotka ovat valittu (kuva 17). Vanhassa porolistassa porot olivat pysyvästi jaettu näihin kahteen osaan. Sama suodatusominaisuus lisätään uuteen porolistaan, mutta yksinkertaistetummin. Koska karttanäkymä ja sivupaneeli toimivat yhdessä, voidaan toisesta suodatuksesta luopua kokonaan. Suodatinnapit lisätään porolistan alkuun. Nämä napit toimivat kiikkukytkiminä, joilla voidaan piilottaa tai näyttää halutut porot kartassa ja listassa. Nappeihin lisätään myös uusi ei dataa -nappi, joka sisältää kaikki porot, joilla ei ole sijaintitietoa tai seurantalaite ei ole käynnistynyt. (Kuva 18.)



KUVA 16. Sovelluksen vanha porolista, jossa välilehdet elävistä ja kuolleista poroista.



KUVA 17. Vanhan karttanäkymän suodatinnapit poroille.



KUVA 18. Porolistan uudet suodatinnapit sivupaneelissa.

4.4 Kartta

Suunnitteluvaiheessa todettiin, että käyttäjä viettää suurimman osan ajasta karttanäkymässä. Siispä karttanäkymään halutaan lisätä porojen toiminnot, kuten jakaminen ja paikannuksen pyytäminen. Porojen toimintojen ja valitsemisen lisäksi karttaan lisätään porojen tunnistamista ja tarkastelemista helpottavia ominaisuuksia, kuten porojen nimet sekä Cluster-ominaisuus. Kartan navigoinnin helppokäyttöisyyttä varten lisätään mahdollisuus kohdistaa näkymä omiin poroihin.

Kartta on rakennettu OpenLayers-kirjaston avulla. OpenLayers on avoimen lähdekoodin JavaScript -kirjasto, jolla pystyy näyttämään digitaalisia karttoja web-sovelluksessa. OpenLayers pystyy näyttämään myös karttasovellukselle tyypillisiä elementtejä, kuten karttamerkkejä tai etäisyysviivoja. (OpenLayers 2023.)

Porojen valitsemiseen käytetään OpenLayersin omaa DragBox-komponenttia. DragBox piirtää vektorityyppisen laatikon, jolla voi rajata kartalta alueen. Alueen sisältö palautetaan rajauksen loputtua, jotta sille voidaan toteuttaa haluttu toiminto. Rajaustoiminto lisätään helposti karttaan luomalla uusi olio DragBox-komponentista ja sisällyttämällä se pääkartan vuorovaikutuslistaan käyttämällä addInteraction-funktiota (kuva 19). Kartan vuorovaikutuslistaan kuuluu kartan käsittelemiseen ja käyttöön suunnattuja ominaisuuksia, esimerkiksi kartan kohdentaminen tai kallistaminen.

Rajaustoiminnon loputtua DragBox käynnistää on-funktion, johon voidaan kirjoittaa omat toiminnot. Tässä tapauksessa haetaan lista porokuvakkeista, jotka ovat alueen sisällä. Porojen identifiointi-numeroilla löydetään vastaavat porot virallisesta porolistasta ja asetetaan niiden checked-tila todeksi. Näin alueen porot ovat valittu. (Kuva 20.)

```
this.dragBoxOL = new DragBox({
  className: 'boxSelect',
  condition: function(e){
    if(StaticVariablesService.isBoxSelect){
      return true;
    } else {
      return false;
    }
  },
});

PARENT_MAP.addInteraction(this.dragBoxOL);
```

KUVA 19. OpenLayersin DragBox-komponentti karttanäkymän TypeScript-koodissa.

```
this.dragBoxOL.on('boxend', ()=>{
  console.log("boxend");
  const extent = this.dragBoxOL.getGeometry().getExtent();
  let boxFeatures;
  let reindeerIds = [];
  boxFeatures = REINDEER_LAYER.getSource().getFeaturesInExtent(extent).filter((feature)=>
  feature.getGeometry().intersectsExtent(extent));
  for(const feature of boxFeatures){
    let featureId = feature.values_.features[0].values_.id;
    reindeerIds.push(featureId);
  }
  for(const id of reindeerIds){
    this.reindeerService.setCheckedStatus(id,true);
  }
  this.addReindeerMapIcons(this.reindeerService.currentlyOnMap);
  if(reindeerIds.length > 0){
    if(!StaticVariablesService.isReindeerSelect){
      this.generalFunctions.ShowPopoverActions();

    }
  }
});
```

KUVA 20. DragBox-komponentin on-funktio, joka asettaa rajatut porot valituiksi.

Cluster-ominaisuus on OpenLayersin oma ominaisuus, jolla kartalla näkyvät kuvakkeet voidaan tiivistää yhdeksi riippuen niiden etäisyydestä toisiin. Cluster-ominaisuutta käytetään yleisesti sellaisissa karttasovelluksissa, joissa kuvakkeita on paljon kartalla. Ominaisuus parantaa kartan luettavuutta sekä sovelluksen suorituskykyä. Cluster-ominaisuus lisätään sovellukseen uutena karttakerroksena. Aikaisemmin lista poroista asetettiin VectorLayer nimiseen OpenLayers-komponenttiin. Nyt porot asetetaan Cluster-komponenttiin ja Cluster-komponentti asetetaan VectorLayer-komponenttiin source-ominaisuutena. Cluster-komponentille annetaan distance-muuttuja, joka määrittää millä etäisyydellä kuvakkeet yhdistetään. (Kuva 21.)

```
const REINDEER_CLUSTER = new Cluster({
  distance: 15,
  source: null, // list of reindeer
});

const REINDEER_LAYER = new VectorLayer({
  source: REINDEER_CLUSTER,
});

const MAPLAYER_GROUP = [
  MAP_LAYER,
  CLIPLAYER_COMBINED,
  LINE_LAYER,
  LOCATION_LAYER,
  REINDEER_LAYER
];

const PARENT_MAP = new Map({
  controls: [],
  interactions: map_interactions,
  layers: MAPLAYER_GROUP,
  view: new View({
    maxZoom: 17,
    center: fromLonLat([25.513321, 65.002794]),
    zoom: 10,
    projection: 'EPSG:3857',
  }),
});
```

KUVA 21. OpenLayers-kartan pääkarttakomponentti, joka sisältää Cluster-kerroksen, johon asetetaan lista poroista.

Sovelluksesta löytyy jo funktio, jolla kartta kohdistaa omiin poroihin. Funktiota kutsutaan aina sovelluksen käynnistyttyä. Uudessa versiossa halutaan, että käyttäjä voi itse toteuttaa kohdistuksen.

Käyttöliittymä pysyy siistinä eikä tukkiudu ylimääräisistä napeista, jos manuaalinen kohdistus toteutetaan kaksoisnapsautuksella. Karttakomponentti seuraa valmiiksi kaksoisnapsautusta on-funktiolla, joten sitä voidaan hyödyntää tässä tapauksessa. Funktion sisällä toteutetaan vain sama kohdistustoiminto, jota sovelluksen käynnistyksessä käytetään. (Kuva 22.)

```
PARENT_MAP.on('dblclick', (evt) =>{
  const selectedFeatures: any =
  PARENT_MAP.forEachFeatureAtPixel(evt.pixel, (thisFeature) => {
    return thisFeature.get("features");
  });
  if(selectedFeatures){
    // zoom more here
    console.log("zooming more...");
    const feature = selectedFeatures[0];
    const coord = feature.getGeometry().getCoordinates();
    const view = PARENT_MAP.getView();
    this.zoomLevel = view.getZoom();
    this.deer_clicked = true;
    this.animateCam(coord, 500, this.zoomLevel + 2);
    return;
  }
  this.zoomToBounds(this.currentReindeerSource);
});
```

KUVA 22. Pääkartan on-funktio, joka seuraa kaksoisnapsautusta ja kohdistaa kartan uudestaan poroihin.

Porojen tunnistamista varten porokuvakkeiden alle lisätään porojen nimet kartassa. Kuvakkeet muodostuvat OpenLayersin Style-komponentista. Style-komponentin sisällä on monia erilaisia määrittelyitä, mutta kaksi tärkeintä ovat image- ja text-määrittelyt. Porojen kohdalla image-tunnisteeseen asetetaan poron kuvake tiedostopolun avulla. Poron tyylin määrittelyyn lisätään vain text-määrite, johon lisätään poron nimi. Uusien ominaisuuksien ohella porokuvakkeilla on myös kaksi uutta tyyliä. Kun porot on asetettu yhteen Cluster-ominaisuudella, porokuvakkeen lisäksi asetetaan kuvakkeen sivuun numero, joka vastaa yhdistettyjen porojen määrää. Kun poro on valittu, lisätään kuvakkeeseen valintaa kuvaava tyyli. (Kuva 23.)

KUVA 23. Porokuvake kartalla ja sen eri tyylit, kun poro on valittu tai poroja on yhdistetty Cluster-ominaisuudella.

4.5 Paikkatieto kysyttäessä

Kuten suunnitteluvaiheessa todettiin, täytyy Paikkatieto kysyttäessä -ominaisuutta muokata sekä päivittää. Aiemmassa versiossa käyttäjä pystyi tekemään pyynnön vain yhdelle päivälle kerrallaan. Tämä tarkoitti, että kalenterinäkymästä pystyi valitsemaan vain yhden päivän. Nyt paikannuksia pystyy tekemään monelle päivälle kerralla, joten useampia päiviä tulisi pystyä valitsemaan kalenterista samaan aikaan. Kalenterinäkymänä käytetään kolmannen osapuolen rakentamaa kalenterikomponenttia, sillä Ionicin oma kalenterikomponentti ei ollut tarpeeksi joustava kaikkiin tarvittaviin muutoksiin. Valittujen päivien määrää pystyy vaihtamaan yksinkertaisesti kalenterikomponentin asetuksista. Koska samaa Paikkatieto kysyttäessä -näkymää käytetään myös paikannuspyyntöjen muokkaamiseen, päivän valitseminen pitää toimia kahdella eri tavalla. Kun uutta paikannuspyyntöä luodaan, päiviä saa valita useamman. Kun paikannuspyyntöä muokataan, päivää on mahdollista vain siirtää. Kalenterin pickMode-asetukseen liitetään muuttuja, joka on joko single tai multi riippuen, muokataanko vai luodaanko paikannuspyyntöä. Muuttujan arvo annetaan, kun Paikkatieto kysyttäessä -sivulle siirrytään. (Kuva 24.)

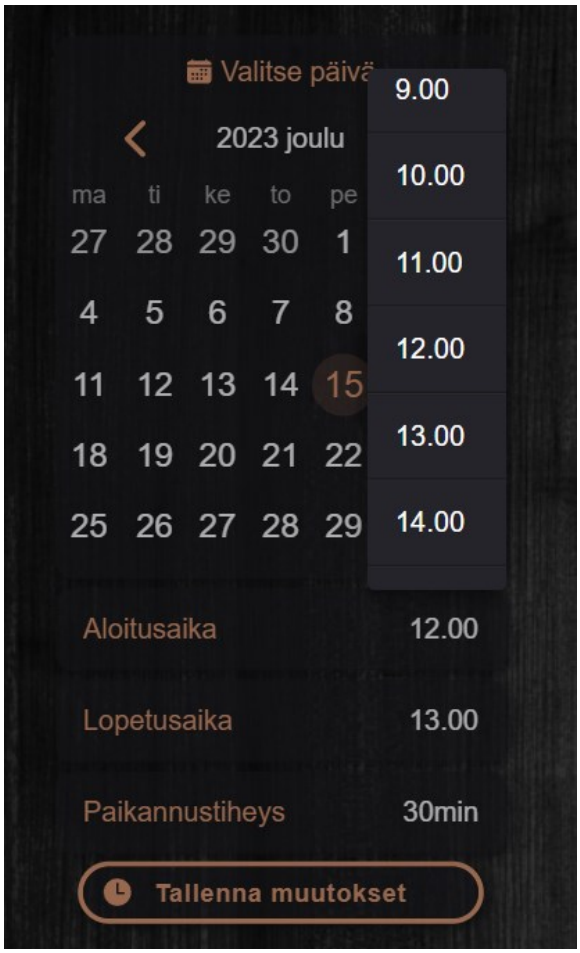
```
refreshCalendar(){
  this.calendarOptions = {
    pickMode: this.selectMode,
    showMonthPicker: false,
    from: this.minDate,
    // to: this.maxDate,
    weekStart: 1,
    // disableWeeks: [0],
    weekdays: [
      this.translateKey("PAGES.LOCATION_REQUESTS.sun"),
      this.translateKey("PAGES.LOCATION_REQUESTS.mon"),
      this.translateKey("PAGES.LOCATION_REQUESTS.tue"),
      this.translateKey("PAGES.LOCATION_REQUESTS.wed"),
      this.translateKey("PAGES.LOCATION_REQUESTS.thu"),
      this.translateKey("PAGES.LOCATION_REQUESTS.fri"),
      this.translateKey("PAGES.LOCATION_REQUESTS.sat")
    ],
    monthFormat: 'YYYY MMM ',
    daysConfig: this.reqDays
  };

  console.log("reqdays: " + JSON.stringify(this.reqDays));
  this.changeDetector.detectChanges();
}
```

KUVA 24. Kalenterikomponentin asetukset, jossa pickMode määrittää, pystyykö kalenterista valitsemaan yhden vai useamman päivän.

Paikannuspyyntöjen tarkastelu aikaisemmassa versiossa toimi seuraavasti. Paikkatieto kysyttäessä -käyttöliittymän yläosassa oli lista valituista poroista, joita haluttiin paikantaa. Kun kalenterista valitsi päivän, paikannettavien porojen lisäksi näkymään lisättiin porot, joille oli jo sinä päivänä asetettu paikannuspyyntö. Paikannettavat porot näkyivät valkoisella tekstillä ja porot, joille paikannuspyynnöt on tehty, sinisellä. Koska paikannuspyynnöt löytyvät nyt sivupaneelista, voidaan vanha pyyntöjen tarkastelu poistaa käyttöliittymästä kokonaan. Poiston lisäksi kalenteriin ei aseteta enää merkintöjä aikaisemmista paikannuksista.

Suunnitteluvaiheen mukaan uusina lisättävinä paikannusten vaihtoehtoina ovat aloitusaika, lopetusaika sekä paikannustiheys. Aikavälin asettamiseen mietittiin muutamaa erilaista näkymää. Ensimmäisessä versiossa pohdittiin, että eräänlainen kellonäkymä olisi helpoin ja kuvaavin tapa asettaa aikaväli. Kellosta voisi poimia kellonajat, jolloin paikannuksen haluaa tapahtuvan. Ongelmaksi syntyi kuitenkin kaksi asiaa. Kellonäkymä ei ole kovin yleinen käyttöliittymäkomponentti, joten siitä ei ole syntynyt yhdenmukaista toimintaperiaatetta. Tämä tarkoittaa sitä, että jokainen käytössä oleva kellonäkymä toimii usein erillä tavalla. Toiseksi aikavälin valitseminen kellosta on itsessään vaikea rakentaa yksinkertaiseksi, sillä aamun ja illan kellonajat menevät päällekkäin. Näin aikavälin asettamisesta syntyy helposti monimutkainen. Toisessa versiossa pohdittiin perinteisempää rullanäkymää, jossa valittavat kellonajat olisivat listassa ja näkymää rullaamalla voidaan valita haluttu kellonaika. Näkymä on paljon yleisempi ja Ionicilta löytyy tähän valmis komponentti. Lopuksi päädyttiin käyttämään yksinkertaista ponnahdusvalikkoa, jossa näkymän rullaaminen ei itsessään määritä valittua kellonaikaa.

Paikannustiheyden kohdalla ensimmäisessä versiossa vaihtoehdot olivat asetettu käyttöliittymään nappikomponentteina, sillä valintoja oli vain neljä ja tilaa riitti. Näin käyttäjän ei tarvinnut avata erillistä valikkoa, kuten ajan valitsemisessa, vaan paikannustiheyden voisi asettaa suoraan perusnäkymästä. Versio ei kuitenkaan mahdollistanut uusien paikannustiheyden lisäämistä tilan puutteen vuoksi, joten tulevaisuuden kannalta päädyttiin käyttämään samaa ponnahdusvalikkoa, jota aikavälin asettamisessa käytetään. (Kuva 25.)



KUVA 25. Paikkatieto kysyttäessä -ominaisuuden uusi käyttöliittymä, jossa aloitusajan ponnahdusvalikko auki.

Aloitusajan, lopetusajan sekä paikannustiheyden valikkoihin käytetään samaa ion-popover-komponenttia, jota esimerkiksi pororyhmien tai paikannuspyyntöjen toimintoihin käytetään. Kellonajoista ja paikannustiheyksistä muodostetaan listat TypeScript-luokassa ja aikaisemmin mainitulla ngFor-määritteellä luodaan jokaiselle listan alkiolle oma ion-item-komponentti. Valinnan lisäksi valintafunktion sisällä on pari yksinkertaista ehtoa. Jos käyttäjä valitsee tunnin mittaisen aikavälin, paikannustiheys asetetaan automaattisesti tunnin mittaiseksi. Kahden tunnin paikannustiheys ei toimi tunnin aikavälissä. Tiheyden lisäksi lopetusaika on aina myöhemmin kuin aloitusaika. Aloitusaikaa aikaisempien kellonaikojen valitseminen on estetty sekä lopetusaika siirtyy automaattisesti eteenpäin, kun aloitusaikaa muutetaan. (Kuva 26.)

```
checkStartTime(index: number){
  this.timeStart = this.startTimeHours[index];
  this.timeStartIndex = index;
  console.log("start time: " + this.timeStart);

  if(this.timeStop == null){
    this.timeStop = this.stopTimeHours[index];
    this.timeStopIndex = index;
  }
  if(this.timeStartIndex > this.timeStopIndex){
    this.timeStop = this.stopTimeHours[index];
    this.timeStopIndex = index;
  }
  this.timeFrame = (this.timeStopIndex + 1) - this.timeStartIndex;

  if(this.timeFrame < 2 && this.timeFrequency == 120){
    this.setTimeFrequency(60);
  }
}
```

KUVA 26. Aloitusajan asettamisen funktio.

Koska uuteen Paikkatieto kysyttäessä -versioon tehtiin muutoksia ohjelmointirajapintaan, täytyy rajapinnan kutsuja muuttaa sovelluksessa. Aikaisemmassa versiossa rajapinta toimi siten, että käyttäjä valitsi porot, joista halusi paikkatietoa. Seuraavaksi hän valitsi haluamansa päivän ja lopuksi lähetti pyynnön eteenpäin. Sovelluksessa muodostettiin jokaisesta valitusta porosta oma rajapintakutsunsa, ja tietokantaan lähti tieto poron identifiointinumerosta sekä päivämäärästä. Nyt uudessa rajapinnassa ei tarvita kuin yksi kutsu, johon liitetään aloitusaika, lopetusaika, paikannustiheys, sekä lista porojen identifiointinumeroista. Paikkatieto kysyttäessä -käyttöliittymässä luodaan samantyyppinen luokkaolio paikannuksesta, mistä Poro- ja pyyntöryhmät -luvussa kerrottiin. Olioon joko asetetaan uudet tai vaihdetaan vanhat paikannuksen asetukset riippuen, muokataanko vai luodaanko paikannusta. Sen jälkeen olio muotoillaan rajapintaa varten ja lähetetään eteenpäin palvelimelle.

5 TULOKSET JA JATKOKEHITYS

Sivupaneelista saatiin aikaiseksi toimiva avattava ikkuna, johon mahtuu selvästi kaikki tarvittavat ominaisuudet. Paneelista löytyy sulkemis- ja avaamistoiminnot sekä animaatiot. Paneeli ei myöskään sulkemisen yhteydessä palaudu alkutilaan. Vanha navigointipaneeli pystyttiin pienentämään ja asettamaan uuteen sivupaneeliin ilman ongelmia. Lisäksi navigointipaneelista pystyttiin luomaan dynaaminen, jossa käyttäjän toiminnan mukaan navigointipaneeli vaihtaa kokoa.

Pororyhmistä sekä paikannuspyynnöistä pystyttiin suunnittelemaan järkevä hakemiston tyylinen tietorakenne ja rakentamaan se käyttäen Ionicin valmiita komponentteja. Lisäksi valmiiden komponenttien avulla saatiin käyttöliittymään avaamis- ja sulkemisanimaatiot. Paikannuspyynnöille, pororyhmille sekä poroille pystyttiin lisäämään toimintonappi, joka käynnistää ponnahdusvalikon sisältäen monia erilaisia toimintoja. Uuteen sivupaneeliin voitiin asettaa aikaisemmin kehitetyt suodatinnapit ja hyödyntää niitä yhtä aikaa porolistassa sekä karttanäkymässä. Valitettavasti pororyhmien lisäämistä ja muokkaamista ei ehditty toteuttaa.

Karttanäkymään onnistuttiin lisäämään ohjaamista ja hallintaan vaikuttavia toimintoja. Käyttäjä pystyy nyt valitsemaan poroja kartalta ja asettamaan suoraan haluttuja toimintoja, kuten paikannus tai porojen jako toisille käyttäjille. Porojen nimet näkyvät nyt kartalla ja kartan pystyy kohdistamaan omiin poroihin manuaalisesti kaksoisnapsautuksella. Poron kuvakkeet pystyvät nyt sulautumaan yhteen Cluster-ominaisuudella, jotta sovelluksen suorituskyky sekä kartan luettavuus pysyy hyvänä.

Paikkatieto kysyttäessä -ominaisuuteen rakennettiin parempi paikannusten tarkasteluun tarkoitettu näkymä. Paikannuksiin annettiin mahdollisuus asettaa aikaväli sekä paikannustiheys, jotta paikannukset vastaisivat paremmin käyttötarkoitukseen samalla pitäen huolen siitä, että seurantalaitteen paristo ei pääse kulumaan nopeasti. Paikannuksen ohjelmointirajapintamuutosten ohella rajapintapyynnöistä pystyttiin luomaan yksinkertaisempia ja selkeämpiä. Paikannuspyynnöille olisi pitänyt rakentaa systeemi, jossa päällekkäiset ja samat paikannukset yhdistetään pitäen paikannuspyyntölista siistinä. Ikävä kyllä ominaisuutta ei ehditty viimeistellä.

Kokonaisuudessaan päivitys oli onnistunut. Suurin osa tarvittavista muutoksista saatiin tehtyä sekä uudet ominaisuudet pystyttiin lisäämään. Kaikkia yksityiskohtia ei valitettavasti ehtinyt toteuttaa,

mutta se mitä ehti tehdä, on rakennettu siten, että töiden loppuun vieminen ei vie paljoa aikaa. Päivityksessä pystyttiin tuomaan aikaisemmin piilossa olevia toimintoja enemmän esille sekä toimintojen suorittaminen on nyt paljon selkeämpää. Sen lisäksi sovelluksen ydin painottuu nyt enemmän yhdelle sivulle, joten sovelluksen käyttäminen on helpompaa.

Jatkokehityksenä on viedä päivityksen uusia ominaisuuksia eteenpäin ja rakentaa niiden päälle käyttämistä helpottavia toimintoja. Esimerkiksi sivupaneelista voidaan luoda enemmän reagoivampi. Paneeli voisi automaattisesti vaihtaa kokoa riippuen siitä, käsitteleekö käyttäjä karttaa vai itse sivupaneelia. Käyttäjälle annettaisiin myös enemmän vaihtoehtoja muokata pororyhmiä tai niiden toimintoja. Pororyhmistä voitaisiin tulevaisuudessa luoda automaattisesti poroja siirtäviä ryhmiä. Karttanäkymässä poroille annettaisiin tunniste omasta pororyhmästä tai kartassa olisi enemmän navigointityökaluja. Paikannuksille annettaisiin uusia vaihtoehtoja, kuten enemmän paikannustiheyksiä tai mahdollisuus tehdä paikannuksista toistuvia. Paikannuksista voitaisiin luoda myös valmiita malleja, joten käyttäjän ei tarvitsisi joka kerta asettaa paikannuksen valintoja uudestaan. Joka tapauksessa kaikkia ominaisuuksia voisi kehittää paljon pidemmälle.

6 YHTEENVETO

Työn tavoitteena oli toteuttaa laajamittainen päivitys Anicare-sovellukseen. Palautteesta saatiin koottua tarpeelliset muutokset ja uudet ominaisuudet, joista muodostui päivityksen kokonaisuus. Päivityksen osat suunniteltiin huolellisesti ja lisättiin sovellukseen. Kokonaisuudessaan työn toteutus onnistui ja päätavoitteet saavutettiin.

Opinnäytetyön työvaihe eteni tehokkaasti tasaiseen tahtiin ilman suurempia ongelmia. Dokumentointivaiheeseen meni huomattavasti enemmän aikaa, kuin oletettiin. Tähän suurin syy oli työvaiheen ja dokumentointivaiheen peräkkäinen jaksottaminen. Muistiinpanojen ja dokumentoinnin kirjoittamista työvaiheen aikana olisi pitänyt tehdä enemmän sekä aiheen rakennetta olisi ollut hyvä suunnitella aikaisemmin.

Itse työssä opittiin paljon hybridi- sekä frontend-kehityksestä, mitä käyttöliittymän rakentaminen vaatii ja kuinka paljon käyttäjän näkökulma vaikuttaa työhön. Työssä opittiin arvostamaan työn tarkkaa suunnittelemista ennen työn toteutusta. Huolellinen suunnitteleminen helpottaa työn kulkua, sillä monet tilanteet ja riskit työn toteutuksesta on monesti jo selvitetty suunnitteluvaiheessa. Työn eri vaiheet on jaksotettu selkeästi eikä missään vaiheessa tarvitse pysähtyä miettimään, mitä seuraavaksi täytyy tehdä.

Hyvin suunniteltu työ mahdollistaa myös asioita, joita ei alun perin ole suunniteltu. Esimerkiksi tässä työssä pororyhmät ja hakemiston tyylinen tietorakenne avaavat monia ovia uusille ominaisuuksille. Paikannuspyyntöjä ei alun perin mietitty lisättäväksi porolistaan, mutta hyvin suunniteltu ryhmärakenne pystyi muovautumaan pororyhmästä myös paikannuspyynnöksi. Uusi Paikkatieto kysyttäessä -ominaisuus kykenee niin moneen erilaiseen paikannukseen uusilla ominaisuuksilla. Periaatteessa uusi paikannus pystyisi korvaamaan vanhan viikoittaisen paikannuksen kokonaan. Esimerkiksi paikannuksen lopetusaikaa ei asetettaisi ollenkaan ja paikannustiheydeksi asetettaisiin viikko.

Kaiken kaikkiaan sovellukseen saatiin onnistunut toimiva päivitys, joka edesauttaa Anicare-sovelluksen käyttöä ja toimintaa porotaloudessa. Opinnäytetyöhön ja sen työtulokseen oltiin tyytyväisiä. Työssä myös opittiin paljon uutta suunnittelemisesta, sovelluskehityksestä ja dokumentoinnista, mistä varmasti on paljon hyötyä tulevaisuudessa.

LÄHTEET

Android Developers 2023. Meet Android Studio. Hakupäivä 24.10.2023. https://developer.android.com/studio/intro.

Angular 2023. What is Angular? Hakupäivä 24.10.2023. https://angular.io/guide/what-is-angular.

Anicare 2023a. Etusivu. Hakupäivä 2.10.2023. https://anicare.fi/.

Anicare 2023b. Tutkimus. Hakupäivä 2.10.2023. https://anicare.fi/tutkimus/.

Capacitor 2023a. A cross-platform native runtime for web apps. Hakupäivä 24.10.2023. https://capacitorjs.com/.

Capacitor 2023b. Capacitor Android Documentation. Hakupäivä 29.10.2023. https://capacitorjs.com/docs/android.

Fireship 2020. YouTube. Angular in 100 Seconds. Hakupäivä 7.10.2023. https://www.youtube.com/watch?v=Ata9cSC2WpM.

Ionic 2023a. Ionic Docs. Introduction to Ionic. Hakupäivä 24.10.2023. https://ionicframework.com/docs

Ionic 2023b. Ionic Docs. Installing Ionic. Hakupäivä 26.10.2023. https://ionicframework.com/docs/intro/cli

Ionic 2023c. Ionic Docs. Starting an App. Hakupäivä 26.10.2023. https://ionicframework.com/docs/developing/starting

Lynch Max 2023. Capacitor: Everything You've Ever Wanted to Know. Ionic. Hakupäivä 24.10.2023. https://ionic.io/blog/capacitor-everything-youve-ever-wanted-to-know.

Mittal Aashiya 2023. What is Ionic Framework? and Why Use it over Other Frameworks? hackr.io. Hakupäivä 24.10.2023. https://hackr.io/blog/ionic-framework.

OpenLayers 2023. OpenLayers. Hakupäivä 23.11.2023. https://openlayers.org/.

The VS Code Team 2016. Visual Studio Code 1.0! Hakupäivä 24.10.2023. https://code.visualstudio.com/blogs/2016/04/14/vscode-1.0.

Visual Studio 2023. Getting Started. Hakupäivä 24.10.2023. https://code.visualstudio.com/docs.