OAMK OULUN AMMATTIKORKEAKOULU

Eero Vilkuna

SENSORIEN ASENNUSSOVELLUKSEN MODUULIN TOTEUTTAMINEN FLUTTERILLA

SENSORIEN ASENNUSSOVELLUKSEN MODUULIN TOTEUTTAMINEN FLUTTERILLA

Eero Vilkuna
Opinnäytetyö
Syksy 2022
Tietojenkäsittelyn tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tradenomi, Tietojenkäsittely

Tekijä: Eero Vilkuna
Opinnäytetyön nimi: Sensorien asennussovelluksen moduuliin toteuttaminen Flutterilla
Työn ohjaaja: Reima Riihimäki
Työn valmistumislukukausi ja -vuosi: Syksy 2022
Sivumäärä: 25

Tämän työn toimeksiantajana on Haltian Oy, joka on IoT-yritys Oulusta. Haltianin tuoteperheen sensoreita varten toteutettiin Thingsee Installer-sovellus. Sovelluksella käyttäjät pystyvät tarkistamaan ja ylläpitämään sensoreitaan.

Tässä työssä käsiteltiin sovelluksen laitetietosivun toteuttaminen käyttäen Flutteria. Sovelluksen laitetietosivulla käyttäjä pystyy tarkistamaan sensorin tietoja kuten paristotason ja asennustilan. Laitetietosivu sisältää myös osuuden missä näkyy sensorin viisikymmentä viimeisintä viestiä. Sovelluksen käyttöliittymässä näytettävien tietojen hakemiseen käytettiin Haltianin Thingsee Open services-rajapintaa. Rajapinnan datan haku ja käsittely on toteutettu sovelluksessa Bloc-tilanhallintapakettia käyttäen.

Työtä katsomalla voin sanoa, että sovelluksen toteutus onnistui hyvin ja vaivattomasti käyttäen Flutteria ja sen lisäosia. Laitetietosivu sisälsi kaikki suunnittelut ja ehdotetut osiot. Käyttäjä pystyy tarkistamaan laitteensa tilan ja viesteissä näkyy oleellisia tietoja. Tulevaisuudessa uusien sensorien implementointi sovelluksessa on helppoa lisäämällä vain tarvittavat tiedot omiin luokkiinsa ja sensorin viestit näkyvät sovelluksessa. Jatkokehittäessä olisi hyvä katsoa uudelleen rajapinnan käytön logiikkaa käyttöliittymässä tarvittavien tietojen latauksen nopeuttamiseksi.

Avainsanoja: Flutter, Tilanhallinta, Dart, Bloc, Thingsee

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Business information technology

Author: VILKUNA EERO
Title of thesis: Creating Installer tool's module with Flutter
Supervisor: Reima Riihimäki
Term and year when the thesis was submitted: Autumn 2022
Number of pages: 25

In this thesis, reader will learn about how an installer tools UI-module was created using Google Flutter SDK that uses Dart programming language also created by Google. Installer tool is used as an admin software and sensor installation tool for Haltian Thingsee sensors.

The goal and purpose of this thesis was to showcase how the module was made and how the components in the application were made to match the designer's plan. Information showcased in the user interface was acquired using Haltian open services API and state management for getting the data was made using bloc library that uses Google's BLoC architecture.

Purpose of the application user interface made in this thesis was to give user the ability to see important information from of their sensor. Device info contains devices status, battery level, installation status, firmware information and operating mode if the sensor has configuration option. In events page user may see the messages coming from sensor. These messages are opened so that the user sees measurements and values of sensors.

Thesis filled all the requirements that I wanted to bring out for this document. It tells the reader basic information about used technologies and bloc state management. All the user interface widgets match to the planned design and work as intended. In the future the user interface could be updated to show graphs of sensor data for more user-friendly experience on seeing the measurement data of the sensors.

Keywords: Flutter, State management, Dart, Bloc, Thingsee

SISÄLLYS

1 JOHDANTO 6
2 TEKNOLOGIAT JA TYÖKALUT 7
2.1 Flutter 7
2.2 Flutter Bloc 8
2.3 JSON-tiedostot 10
3 TOTEUTUS 11
3.1 Laitteen tunnistusnumero ja laitekuva 12
3.2 Bloc-tilanhallintakirjaston käyttäminen sovelluksessa 13
3.3 Laitteen tila 16
3.4 Paristo taso 17
3.5 Laitteen ohjelmistonversio ja laiteryhmä 17
3.6 Asennuksen tila 18
3.7 Laitteen toimintatila 19
3.8 Laiteviestit 20
4 POHDINTA 23
LÄHTEET 25

1 JOHDANTO

Tässä työssä toteutetaan sensorien asennussovelluksen käyttöliittymään laitetieto-osuus. Tämän osuuden avulla käyttäjä näkee sensorin tiedot ja viisikymmentä viimeisintä sensorin lähettämää viestiä. Sovellus helpottaa käyttäjän sensorien ylläpitoa. Opinnäytetyö tehdään Haltian Oy:lle (jatkossa Haltian) vuoden 2022 aikana.

Haltian on globaali IoT- ja tuotekehitystalo. Haltianin kehittämä, laajasti käytössä oleva IoT (Internet of Things) -teknologia-alusta Thingsee nopeuttaa laajoja IoT-käyttöönottoja. Thingsee on kasvava tuoteperhe, joka sisältää muun muassa sensoreita ja tukiasemia. Laitteet toimivat Thingsee-Cloud-pilvipalvelun kanssa. Asiakas voi käyttää Thingsee-perheen tuotteita itselleen soveltuvan älyratkaisun rakentamisessa. Haltianin Empathic Building -älytoimisto on tuote, jossa käyttäjä voi nähdä ympäristön lämpötilan, ilmanlaadun ja muita arvoja. Käyttäjä voi myös varata neuvotteluhuoneen tai vapaana olevan työpöydän toimistolta sovelluksen avulla. Haltian tarjoaa myös tuotekehitys- ja suunnittelupalveluja yrityksille. (10.)

Opinnäytetyön aihe on osa sovellusta, minkä uudistamiselle tuli tarve, kun Haltianin tuotteita varten oli kehitetty asennustyökalu Thingsee Installer. Vanhan sovelluksen päivittäminen olisi ollut liian työlästä ja tulevaisuudessa sovelluksen ylläpito olisi ollut hankalaa. Toteutuksessa oli paljon toistuvaa koodia mikä vaikeutti lähdekoodin luettavuutta. Toteutuksessa oli myös käyttöliittymä ja sovelluslogiikkaa sekoitettuna. Sovellus päätettiin rakentaa uudelleen käyttäen Googlen kehittämää Flutter-ohjelmistokehityspakettia, jota käytettiin myös alkuperäisessä sovelluksessa. Uuden toteutuksen tarkoituksena on luoda käyttäjälle ohjelma, jonka avulla hän pystyisi tarkistelemaan laitteitaan ja muuttamaan laitteen asetuksia.

2 TEKNOLOGIAT JA TYÖKALUT

Sovellus toteutettiin Googlen Flutter-ohjelmistokehityspakettia käyttäen. Flutter-ohjelmistokehityspaketti käyttää Googlen kehittämää Dart-ohjelmointikieltä. Aiempi sovellus oli kehitetty Flutterilla, joten sovelluksen uudistaminen toteutettiin samaa teknologiaa käyttäen.

Ohjelmoinnissa käytettiin Microsoftin julkaisemaa avoimen lähdekoodin Visual Studio Code-ohjelmointiympäristöä. Visual Studio Code-ympäristössä on ohjelmointia varten otettu käyttöön Googlen tekemät viralliset lisäosat Dartillle ja Flutterille, jotka tarjoavat ohjelmointia helpottavia ominaisuuksia ja työkaluja. (4a.)

Versionhallinnassa käytettiin Git-versionhallintaohjelmaa, jolla tallennettiin lähdekoodin muutokset Gerrit-palveluun. Gerrit on webpohjainen tiimikoodityökalu. Gerrit-palvelun sisällä tiimin jäsenet voivat tarkistaa toistensa tekemät muutokset ja hyväksyä tai hylätä kyseiset muutokset. (2.)

2.1 Flutter

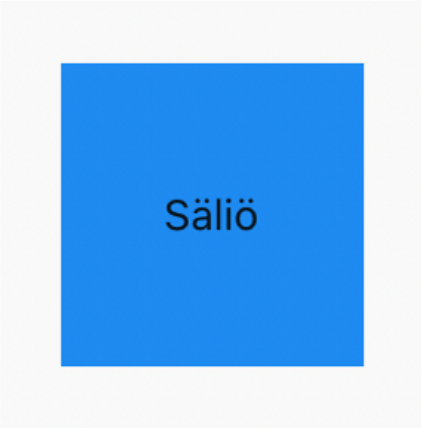
Flutter on Googlen kehittämä avoimen lähdekoodin ohjelmistokehityspaketti. Paketti julkaistiin vuonna 2018. Flutter soveltuu alustariippumattomien sovellusten kehittämiseen iOS-, Android-, Windows-, Linux-, macOS-, Web- ja sulautettuihin käyttöjärjestelmiin. Flutter-ohjelmoinnissa käytetään Googlen kehittämää Dart-ohjelmointikieltä. Dart-ohjelmointikieli käyttää tiukkaa tyyppijärjestelmää ohjelmoinnissa. Tästä johtuen tietotyypit pitää asettaa oikein jo ohjelman kehitysvaiheessa. (11.)

Flutterille ja Dartille on tarjolla laaja kirjasto erilaisia ohjelmapaketteja. Osa paketeista on Googlen kehittämiä, mutta suurin osa ohjelmapaketeista on muiden Flutter- ja Dart-ohjelmoijien kehittämiä paketteja. Paketti voi sisältää Dart-kirjastoja, sovelluksia, resursseja, testejä, kuvia ja esimerkkejä. Paketteja käyttämällä vähennetään ohjelmoinnin määrää ja nopeutetaan sovelluskehitystä. (6.)

Flutter sovelluksen toteutuksessa käytetään widgettejä eli ohjelmistokomponentteja. Sovellus rakennetaan widgettien avulla ryhmitellen sisältö tietyn ominaisuuden tai tarkoituksen mukaan.

Widgettiä pystytään käyttämään uudelleen, ja myös widgettiä pystyy käyttämään toisen widgetin sisällä. Kuvion 1. esimerkki sisältää tekstikomponentin keskityskomponentin sisällä. (5b.)
Kuviossa 1. on toteutettu esimerkki widgetti, missä on kaksi Container-widgettiä täytetty sinisellä värillä ja Text-widget on käytössä tekstin esittämistä varten. Molemmat komponenteista on keskitetty siihen keskitykseen tarkoitetulla Center-widgetillä. Koodi tuottaa sinisen nelikulmion, minkä sisällä on keskitettynä teksti: ”säiliö”.

```
1  import 'package:flutter/material.dart';
2
3  main() {
4    final container = Center(
5      child: Container(
6      width: 100,
7      height: 100,
8      color: Colors.blue,
9      child: Center(child: Text('Säliö')),
10   ));
11
12   final sovellusrunko = Scaffold(body: container);
13   final sovellus = MaterialApp(home: sovellusrunko);
14
15   runApp(sovellus);
16 }
17
```

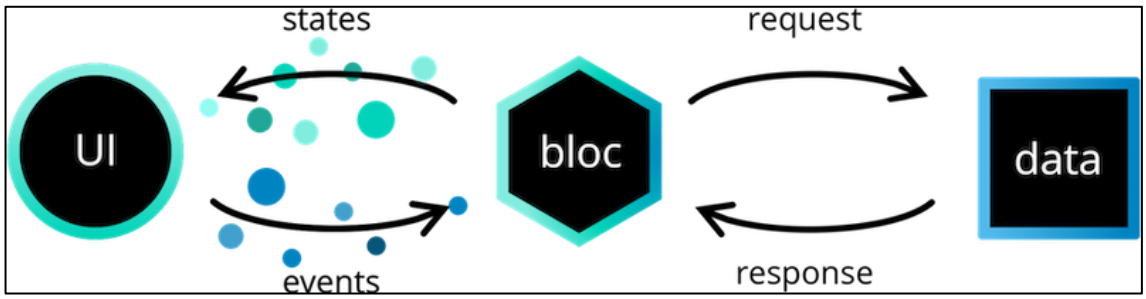


KUVIO 1 Esimerkki widgetin lähdekoodi ja tulostus

2.2 Flutter Bloc

Flutter Bloc on Felix Angelovin luoma ja ylläpitämä tilanhallintakirjasto. Flutter-Bloc-kirjaston toteutuksessa käytetään Business logic component-kaavaa, joka on Googlen kehittämä ja esiteltiin vuonna 2018 Google Developers tapahtumassa. (3.) Sovellukseen ohjelmapaketti tuodaan ohjelmapakettisivustolta. Ohjelmapaketin avulla eriytetään kaikki sovelluslogiikkaan liittyvä koodit omiin luokkiinsa.

Käyttöliittymäosa ottaa vastaan tilan, minkä tilanhallintaosa on syöttänyt eteenpäin ja tilan perusteella toteuttaa käyttöliittymässä näkymän. Kolmas osa on informaatiota sisältävä osa, mistä tilanhallinta kysyy ja vastaanottaa dataa. Datan vastaanottamisen jälkeen tilanhallinta välittää tilan käyttöliittymälle. Käyttöliittymässä käsitellään myös käyttäjän ja sovelluksen tekemät tapahtumat. (KUVIO 2.)



KUVIO 2 Bloc-tilanhallinnan toimintakaava (Bloclibrary.dev 2022)

BlocProvider-widgetin tehtävänä on luoda rajapinnan tila ja myös tarjota tila eteenpäin käyttöliittymälle. Blocproviderilla varmistetaan, että vain tietty instanssi annetaan eteenpäin muille osille. Jos käyttöliittymän sivu vaati monta eri BlocProvideria, on käytössä MultiBlocProvider-widgetti, jota hyödyntämällä parannetaan koodin luettavuutta ja eliminoidaan tarve käyttää useaa BlocProvideria vähentäen toistuvan koodin määrää. Kuviossa 3. on toteutettu yksinkertainen BlocProvider. (1.)

```
BlocProvider(
  create: (BuildContext context) => BlocA(),
  child: ChildA(),
);
```

KUVIO 3 BlocProvider-widgetin esimerkki

BlocListener (KUVIO 4.) on widgetti, joka tarkkailee Bloc-tilan muutoksia ja toimii sen mukaan. BlocListenerin pitäisi olla käytössä vain kerran tilan muutoksen aikana. Esimerkiksi kun käyttäjä siirtyy laitesivulle, sivua ei näytetä ennen kuin BlocListener palauttaa arvon (tosi tai epätosi) ja varmistaa, että kaikki tiedot on ladattu. Myös BlocListeneristä on olemassa usean kuuntelijan käyttämistä varten tarkoitettu widgetti MultiBlocListener. Tämä widgetti vähentää toistuvaa koodia ja selkeyttää koodin lukemista. (1.)

```
BlocListener<BlocA, BlocAState>(
    listener: (context, state) {
      // Toteutus jossa reagoidaan BlocA:n tilan mukaan
    },
    child: Container(),
  )
```

KUVIO 4 BloxListener-widgetin esimerkki

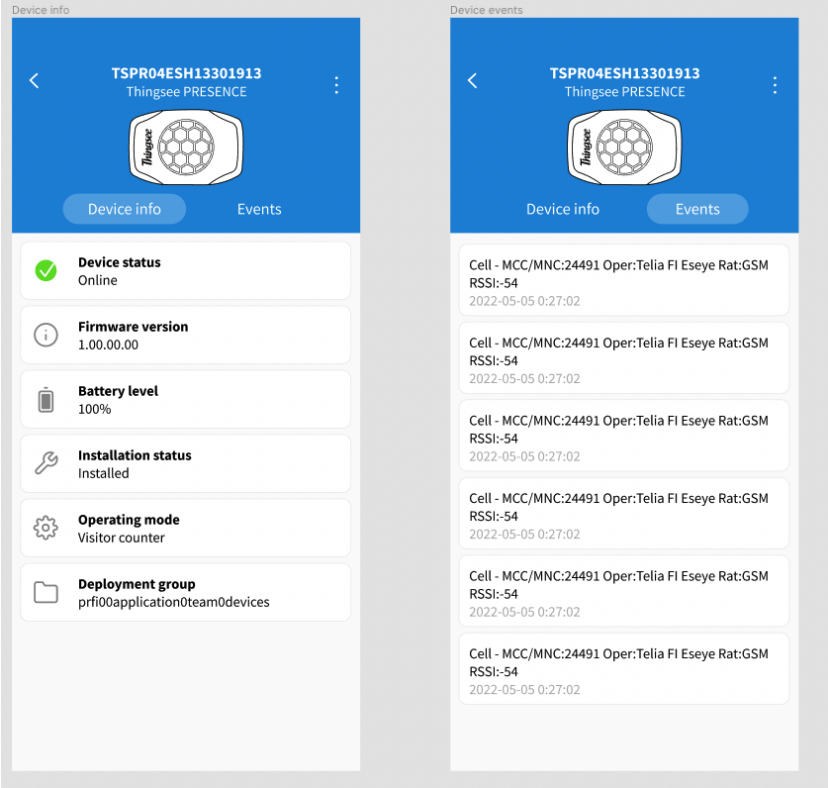
2.3 JSON-tiedostot

JSON (JavaScript Object Notation) on standardoitu strukturoidun datan esitysmuoto. JSON on nimensä mukaisesti alun perin JavaScript-ohjelmointikielestä muodostunut. Se on nykyjään laajalti käytössä oleva menetelmä tiedon vaihtamiseen järjestelmissä ja niiden välillä. JSON-tiedostot sisältävät kuusi erilaista tietotyyppiä:

1. Merkkijonot on suljettu lainausmerkein. Merkkijono voi sisältää kaikkia Unicode-merkkejä, kunhan ne ovat lainausmerkkien sisällä.
2. Numerot kirjoitetaan ilman lainausmerkkejä. Useimmat JSON-toteutukset olettavat numerot kokonaislukuina, koska desimaalipistettä ei ole käytössä.
3. Totuusarvomuuttuja eli boolean sisältää toteamuksen tosi tai epätosi.
4. Nolla eli null-literaaliarvo on käytössä ilmaisemaan tyhjää tai puuttuvaa arvoa.
5. Objekti on struktuuri, joka esitetään aaltosulkeilla sisältäen tekstiä tai arvoja lainausmerkkien sisällä. Objektin sisällä olevat arvot on eroteltu pilkulla, ja objektin sisällä olevien nimien tulee olla uniikkeja. Objektit, joiden nimet ovat ainutlaatuisia, ovat yhteentoimivia, että kaikki kyseisen objektin vastaanottavat ohjelmistototeutukset ovat yhtä mieltä nimi/arvo kuvauksista. Jos nimet eivät ole uniikkeja, on sovelluksen toiminta arvaamaton.
6. Taulukko on yksinkertainen luettelo, joka on suljettu hakasulkeisin. Taulukko tukee kaikkia tietotyyppejä ja sen sisällä olevat elementit on eroteltu pilkulla. Elementtien määrää taulukon sisällä ei ole erikseen rajoitettu. (12.)

3 TOTEUTUS

Sovelluksen käyttöliittymän ulkoasu tehdään suunnittelijan suunnitelman mukaisesti. Suunnitelmasta saadaan mallit, fontit ja mittasuhteet kaikille sovelluksen toteutukseen kuuluville osille. Molempien sivujen tietolaatikot rakennetaan erillisinä widgetteinä. (KUVIO 5.)



KUVIO 5 Käyttöliittymän suunnitelma

Laitesivulla näytetään käyttäjälle suunnittelussa päätetyt keskeisimmät tiedot, jotka käyttäjän halutaan näkevän ensimmäiseksi käyttäjän siirtyessä laitesivulle laiteryhmäosiosta. Sivun toteutus vaatii usean eri rajapinnan pyyntöä tarvittavien tietojen saamiseksi. Kun sovellus alkaa hakemaan tarvittavia tietoja sivulle, näkyy tiedonhaun ilmoittava pyörivä rengas, joka estää sivun näkymisen, ennen kuin kaikki tarvittavat tiedot on vastaanotettu rajapinnasta sovellukseen.

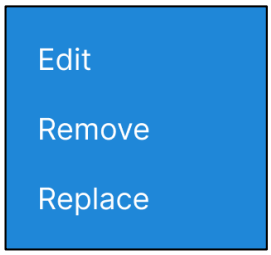
Toteutuksessa tarvittavat kyselyt rajapinnasta tarjoavat toteutuksen käyttöliittymässä käytettäville komponenteille aikaleiman, paristotason, laitteen ohjelmiston version, asennustilan, konfiguraation ja laiteviestit.

3.1 Laitteen tunnistusnumero ja laitekuva

Sivun yläosassa on sovelluspalkki sisältäen painikkeen takaisin siirtymistä varten. Sovelluspalkin keskelle on sijoitettu laitteen sarjanumero, jonka alapuolella on laitemalli. (KUVIO 6.) Laitteen malli saadaan näkymään käyttöliittymässä funktiolla, joka tarkistaa sarjanumeron alkuosuudesta osan ja vertaa sitä annettuihin arvoihin.

KUVIO 6. AppBar, laitekuva ja sivunilmaisin

Työkalupalkin oikealla on valikko, mistä käyttäjä voi siirtyä tarkasteltavan laitteen muokkaussivulle poistamaan laitteen tai korvaamaan laitteen järjestelmässä, esimerkiksi jos käyttäjän kyseinen laite on vioittunut. Laiteen asetuksia on myös mahdollisuus siirtyä muokkaamaan, jos jokin asetus on mennyt väärin. (KUVIO 7.)



KUVIO 7. Muokkausvalikko

Käyttöliittymän komponentin alempi osuus on Pageview-widgetti. Sen avulla luodaan käyttöliittymässä tarvittavat sivut, ja mukana tulee ominaisuus selata sivuja ja seurata, millä sivulla käyttäjä on. Sivujen vaihto tapahtuu hipaisulla tai käyttäen painikkeita laitekuvan alapuolella. Painikkeille on asetettu tyyli valmiiksi luomalla erillinen painike. Samaa painiketyyliä käytetään monessa eri painikkeessa sovelluksessa. Tällä tavoin saadaan helposti karsittua toistuvaa koodia

ja pidettyä käyttöliittymäkomponentit yhtenäisenä. Kun sivuja tarkkaillaan ja käyttäjä vaihtaa sivulta toiselle, muuttuu yläosioissa olevan painikkeen väri sinisestä vaaleaksi esittäen kummalla sivulla käyttäjä on. (KUVIO 8.)

```
Expanded(
  child: PageView(
  onPageChanged: (int index) {
    setState(() {
      _selectedPageIndex = index;
    });
  },
```

```
Color _buttonColor(int index) {
  return _selectedPageIndex == index
      ? InstallerColor.whiteColor.withOpacity(0.25)
      : InstallerColor.blueColor;
}
```

KUVIO 8. Sivun ilmaisimen värin muutoksen funktio lähdekoodissa

Laitetieto-sivulla näkyvät laitetietolaatikot näyttävät erilaisia mittaustietoja sensorista ja rajapinnasta haettuja laitteen tietoja. Laitetietolaatikoiden painaminen ei suorita sovelluksessa mitään tapahtumaa, laitetietolaatikot ovat käytössä vain tiedon esittämistä varten. Laitetietolaatikossa voidaan myös esittää vasemmalla puolella ikoni, kuten paristoikoni laitteen paristotieto-laatikon kohdalla.

3.2 Bloc-tilanhallintakirjaston käyttäminen sovelluksessa

Kun käyttäjä siirtyy laitetietosivulle, pitää laitetietosivun tila alustaa. Laitetietosivulla tarvittavia tietoja varten käytetään useita rajapintakyselyitä, joiden vastauksista kerätään tiedot näytettäväksi käyttöliittymässä laitetietolaatikoissa. Jotta Bloc-ohjelmakirjasto saadaan käyttöön, pitää sovelluksen riippuvuuksiin lisätä käytettävän paketin nimi. (KUVIO 9.)

```
dependencies:
  flutter:
    sdk: flutter
  flutter_localizations:
    sdk: flutter
  flutter_bloc: ^8.0.1
```

KUVIO 9 Riippuvuuden lisääminen sovellukseen

Sivulle siirtyessä ohjelma käynnistää laitetietojen saamiseksi tarkoitetun tilanhallintafunktion FetchDeviceInfo. Funktio sisältää kaikki käyttöliittymän moduulissa tarvittavien tietojen tilanhallintatoimittajat. Jokainen sivulla käytettävä tilanhallinta käynnistetään portaittain. Näin vältytään ongelmilta, joita voi esiintyä, jos kaikki tilanhallinnan toimittajat laitettaisiin käyntiin samaan aikaan. (KUVIO 10.)

```
Future _fetchDeviceInfo() async {
  BlocProvider.of<FetchDeviceInfoCubit>(context)
      .fetchDeviceInfo(
          stack: widget.arguments.stack, deviceID: widget.arguments.deviceID)
      .then((value) => BlocProvider.of<FetchInstallationStatusCubit>(context)
          .fetchInsatllationStatus(
              stack: widget.arguments.stack,
              deviceID: widget.arguments.deviceID))
      .then((value) => _setDefaults(context))
      .then((value) => BlocProvider.of<CommandsCubit>(context)
          .fetchAllCommands(
              stack: widget.arguments.stack,
              deviceId: widget.arguments.deviceID))
      .then((value) => BlocProvider.of<FetchDeviceMessagesCubit>(context)
          .FetchDeviceMessages(
              stack: widget.arguments.stack,
              deviceId: widget.arguments.deviceID,
              limit: 50))
      .then((value) => BlocProvider.of<FetchGroupCubit>(context).fetchGroup(
          stack: widget.arguments.stack,
          deviceID: widget.arguments.deviceID));
}
```

KUVIO 10. Tilanhallinnan toimittajien käynnistys

Kun tilanhallinta on käynnistetty, BlocListener-widgetit tarkkailevat tilanhallinnassa olevaa tilaa. Kun tilanhallinta pyytää rajapinnan kautta tietoa, on etukäteen määritelty, mitkä tiedot tarvitsemme rajapinnasta. Määrittely toteutetaan tehdasluokkaa käyttäen. Tehdasluokkaan lisätään haluttavien

tietojen tyypit ja nimet. Tyypin perään on laitettu kysymysmerkki: jos rajapinta palauttaisi arvon tyhjänä, se ei aiheuta ongelmia sovelluksessa. (KUVIO 11.)

```
import 'package:json_annotation/json_annotation.dart';
part 'device_info.g.dart';

@JsonSerializable()
class DeviceInfo {
  final int? battery_level;
  final int? timestamp;
  final String? gateway_tuid;
  final String? version;

  DeviceInfo(
      this.battery_level, this.timestamp, this.gateway_tuid, this.version);

  factory DeviceInfo.fromJson(Map<String, dynamic> json) =>
      _$DeviceInfoFromJson(json);

  Map<String, dynamic> toJson() => _$DeviceInfoToJson(this);
}
```

KUVIO 11. Tehdasluokka

Kun tilanhallinta hakee tarvittavia tietoja rajapinnasta käyttäjälle, käyttöliittymässä esitetään haun olevan vielä käynnissä käyttäen CircularProgressIndicator-widgettiä. Se luo käyttöliittymään ympyrän, joka pyörii siihen asti, kunnes jokainen tilanhallinta on palauttanut totuusarvon true onnistuneen tietojen haun rajapinnasta. Totuusarvon palauttaminen määritellään kuuntelijassa asettamalla totuusarvo muuttujalle. (KUVIO 12.) Käyttäjä voi myös vetää sivua alaspäin ja käynnistää sivun alustamisen uudelleen ja päivittää sivulla näkyvät tiedot.

```
BlocListener<FetchDeviceInfoCubit, FetchDeviceInfoState>(
  listener: (context, state) {
    if (state is FetchDeviceInfoInProgress) {
      setState(() {
        _loadingDeviceInfo = true;
      });
    } else if (state is FetchDeviceInfoFailed) {
      setState(() {
        _loadingDeviceInfo = false;
      });
    } else if (state is FetchDeviceInfoSuccess) {
      setState(() {
        _deviceInfo = state.info;
        _loadingDeviceInfo = false;
      });
    }
  },
), // BlocListener
```

KUVIO 12. Tilan kuuntelija ja latausindikaattori

Onnistuneen haun jälkeen syötetään tilasta etukäteen määritetyt tiedot deviceInfo-muuttujaan.

Nyt käyttöliittymäkomponentti näyttää tiedon kutsumalla esimerkiksi ”deviceinfo.timestamp”-komentoa, jolla tuodaan laitteen aikaleima esille sovelluksen käyttöliittymään.

3.3 Laitteen tila

Laitetieto sivulla käyttäjälle näytetään, onko kyseinen laite päällä vai ei. Tarkoituksena on ottaa yllä mainitusta rajapinnasta aikaleima numerosarja. Mikäli sensorin aikaleima ei ole päivittynyt kahteen tuntiin niin sensorin tilan näytetään olevan pois päältä. Laitteen tilan tarkastamiseksi kutsutaan käyttäen _isOnline funktiota. Funktiossa vertaillaan laitteen aikaleimaa nykyhetkeen sen sisällä. Ensiksi funktio hakee nykyhetken aikaleiman millisekunteina. Myös alkuperäinen aikaleima pitää muuttaa millisekunneiksi. Palautuslauseessa verrataan aikaleimojen eritystä kahteen tuntiin ja palautetaan käyttöliittymälle tosi, jos arvo on enemmän kuin 2 tuntia tai epätosi jos vähemmän. (KUVIO 13.)

```
bool isOnline(int timestamp) {
  timestamp = timestamp * 1000;
  int _now = DateTime.now().millisecondsSinceEpoch;
  return (_now - timestamp) >= 7200000 ? true : false;
}
```

KUVIO 13. Laitteen tilan tarkistus

Jos laite on pois päältä, näytetään laitetietosivulla käyttäjälle laitteen viimeisin aikaleima. Aikaleima on Unix-muodossa. Kyseinen muoto ei ole helposti luettavissa, joten se pitää formatoida käyttäjälle. Unix-aikaleima ilmaisee ajan sekunteina ajanhetkestä 1. tammikuuta 1970 kello 0.00.00. Funktiossa kutsutaan aikaleimaa millisekunteina. Funktiossa muutetaan laitteen tietokyselyssä annettu aikaleima. Flutter sisältää ominaisuuden, jota hyödyntäen pystytään formatoimaan aikaleima luettavaan muotoon ja muuttaa se käyttäjän laitteen aikavyöhykkeelle. (KUVIO 14.)

```
String formatTime(int timestamp) {
  DateTime date =
      DateTime.fromMillisecondsSinceEpoch((timestamp * 1000), isUtc: true);
  var formattedDate = DateFormat('d MMM yyyy HH:mm').format(date.toLocal());
  return formattedDate.toString();
}
```

KUVIO 14. Aikaleiman muuttamisen funktio

3.4 Paristotaso

Sensorin pariston lataustason esittämiseksi otetaan rajapinnan palautuksesta paristodata elementti ”battery_level” ja muotoillaan se sovelluksessa oikean tulostuksen esittämiseksi. Mikäli kyseinen sensori toimii USB-virransyötöllä eikä paristoilla ilmoitetaan numeroarvon sijasta, että laite toimii USB-virransyötöllä. Muiden sensorien kohdalla otetaan palautuksesta paristotason numero arvo ja palautetaan se merkkijonona ja prosenttimerkki arvon perässä. Jos palautuksen arvo on null käyttöliittymä ilmoittaa, että ei ole paristo dataa saatavilla. (KUVIO 15.)

```
String batteryLevel(int? batteryLevel, BuildContext context) {
  if (batteryLevel == null) {
    return AppLocalizations.of(context)!.noBattery;
  } else {
    return batteryLevel.toString() + "%";
  }
}

String batteryStatus(DeviceModel _deviceModel, BuildContext context,
    ThingseeSensorType sensorType, DeviceInfo info) {
  if (sensorType == ThingseeSensorType.gatewayGlobal ||
      sensorType == ThingseeSensorType.gatewayLan ||
      sensorType == ThingseeSensorType.count) {
    return AppLocalizations.of(context)!.usbPowered;
  } else
    return batteryLevel(info.battery_level, context);
}
```

KUVIO 15. Pariston lataustason tarkistamiseen funktiot batterylevel ja batteryStatus

3.5 Laitteen ohjelmistonversio ja laiteryhmä

Ohjelmistoversio esitetään, jos rajapinnasta haettu laitetieto sisältää sen viestissään. Laitetiedon edessä funktio tarkistaa, että rajapinnan palautus ei ole tyhjä ja komponentti näytetään käyttöliittymässä. Mikäli palautus on onnistunut ja sisältää tarvittavan tiedon, muutetaan ohjelmiston versio palautus data merkkijonoksi lisäämällä datan perään. (KUVIO 16.)

```
if (_deviceInfo?.version != null)
  InstallerListTile(
      title: AppLocalizations.of(context)!.firmwareVersion,
      description: _deviceInfo?.version.toString(),
      onPressed: () {},
      icon: InstallerIcons.createInstallerAssetImage(
          assetPath: InstallerIcons.firmware, width: 30, height: 30),
      includeArrowIcon: false), // InstallerListTile
```

KUVIO 16. Ohjelmistoversion laitetietolaatikko lähdekoodissa

Käyttäjälle näytetään käyttöliittymässä laitteen laiteryhmä. (KUVIO 17.) Laiteryhmä näytetään koska, jos käyttäjä siirtyy laitteenhaku-sivulta, hän ei välttämättä tiedä laitteen ryhmää. Laiteryhmällä käyttäjä jakaa laitteet esimerkiksi asennuspaikkojen perusteella tai muulla haluamallaan muistisäännöllä. Jos laite on uusi tai sille ei ole määritetty erikseen luotua ryhmää laiteryhmänä on ”Unassigned”-ryhmä.

KUVIO 17. Laiteryhmä laitetietolaatikko käyttöliittymässä

3.6 Asennuksen tila

Haltianin Thingsee-sensoreilla on asennuksen tilatieto. Tiedolla seurataan, onko laite asennettu, vaihdettu tai poistettu käytöstä. Asennustiloja on 5 erilaista:

1. ”new" – Uusi laite tehtaalta.
2. ”installed” – Sensori on asennettu.
3. ”uninstalled” – Laite on poistettu käytöstä mutta se odottaa uudelleen asentamista.
4. ”quarantine” – Laite käyttäytyy erikoisesti ja on siirretty valvontaan.
5. ”retired” – Laite on otettu pois käytöstä eikä palaa enää käyttöön.

(9c.)

Asennuksen tilan noutamista varten haemme rajapinnan avulla viimeisimmän viestin rajapinnan palautuksesta rajaamalla haettavien viestien määrän yhteen.

Rajapinta palauttaa asennustilan alkukirjaimen pienaakkosisena. Tämän korjaamiseksi tehtiin funktio missä palautuksen merkkijonon ensimmäinen kirjain muutetaan suuraakkoseksi. (KUVIO 18.)

```
String _formattedInstallationStatus(String status) {
  return "${status[0].toUpperCase()}${status.substring(1)}";
}
```

KUVIO 18. Suuraakkosen muutos

3.7 Laitteen toimintatila

Thingsee Enviroment, Thingsee Enviroment rugged, Thingsee Presence ja Thingsee Count -sensoreilla on erillaisia toimintatiloja. Toimintatilojen muuttamista varten asiakkailla ei ole ollut ennen sovellusta ja uuteen sovellukseen haluttiin toteuttaa tapa näyttää käyttäjälle laitteiden toimintatilan muutos, ja laitetieto-sivulle mikä toimintatila sensorissa on käytössä.

Enviroment-sensoreilla on käytössä kaksi eri muutettavaa toimintatilaa: Hall ja Machine usage mode -tilat. Hall-tila seuraa sensorin ympärillä tapahtuvaa magneettista muutosta ja Machine usage mode seuraa laitteen tärinää ja suuntausta. Thingsee Enviroment sensorilla seurataan tärinää, magneettisen kentän muutoksia, laitteen suuntautumista, Ilmanpainetta, valonmäärää, ilmankosteutta ja lämpötilaa. (7a.)

Thingse Presence sensorin konfiguraatioina on vaihtoehtoina kävijä-, muutos- tai läsnäolotila. Pääasiassa näillä toimintatiloilla muutetaan laitteen tapaa lukea tilaa ja sitä, kuinka laite raportoi tilansa järjestelmälle. Thingsee Presence -sensorin monien käyttötarkoitusten takia sensorin toimintatilan muuttaminen on käyttäjälle tarpeellinen työkalun lisä. (8b.)

Thingsee Count-sensori on tuorein lisäys Haltianin Thingsee-tuoteperheeseen. Se antaa käyttäjälle tarkan lukeman kävijämäärästä. Käyttökohteina voivat olla esimerkiksi neuvotteluhuoneen kävijämäärän ja ajankohtaisen käyttöasteen seuranta. Count-sensorin konfiguraatiossa määritetään, onko sensori huoneen sisäpuolella vai ulkopuolella.

Konfiguraatio saadaan DeviceCommands-muuttujan avulla. Muuttujan tilanhallinta palauttaa listan, joka sisältää laitteelle lähetetyt konfiguraation muutoksen viestit. (KUVIO 19.) Toteutuksessa

tarvitaan vain sensorin uusin toimintatilaviesti. Tarvittava tieto saadaan ottamalla palautetun listan ensimmäinen osa. Mikäli laitteelle ei ole tehty konfiguraation muutosta käytetään laitteille etukäteen määriteltyjä oletustoimintatiloja.

```
void _setConfigurationsNames(DeviceCommand? command) {
  if (command!.mode != null) {
    setState(() {
      _selectedOperatingMode =
          installationModel.presenceModes(context).elementAt(command.mode!);
    });
  }
  if (command.peopleCountingOrientation != null) {
    setState(() {
      _selectedInstallationLocation = installationModel
          .installationLocations(context)
          .elementAt(command.peopleCountingOrientation!);
    });
  }
  if (command.accelerometerMode != null) {
    setState(() {
      _selectedAccelerometerMode = installationModel
          .accelerometerModes(context)
          .elementAt(command.accelerometerMode!);
    });
  }
  if (command.hallMode != null) {
    setState(() {
      _selectedHallMode =
          installationModel.hallModes(context).elementAt(command.hallMode!);
    });
  }
}
```

KUVIO 19. Konfiguraatioiden asettaminen

3.8 Laiteviestit

Sensorit lähettävät mittaustietoja, -arvoja ja laiteviestejä useita määriä päivän aikana. Laiteviestit-sivulla haetaan rajapinnan kautta viisikymmentä viimeisintä viestiä. Viestistä haluttu data esitetään InstallerListTile-widgetin otsikko kohdan sisällä ja viestin aikaleima-alaotsikossa. Jokainen viesti sisältää viestin tunnistenumeron. Tunnistenumeron perusteella voimme asettaa sensorin tietojen perusteella viestistä datan widgettiin näkyville. Kun viestit on haettu rajapinnan kautta pitää laiteen viestidata lista muuttaa laitetietolaatikoksi funktiolla _messageList. (KUVIO 20.) Sen sisällä kartoitetaan listalla olevat viestit ja viestien haluttu tieto lisätään laitetietolaatikkoon tuoden tiedon käyttöliittymään esille.

```
List<Widget> _messageList(
    List<DeviceMessages> _messageList, BuildContext context) {
  final List<Widget> items = [];
  items.add(const SizedBox(
    height: 9,
  ));
  _messages.map((DeviceMessages messages) {
    items.add(InstallerListTile(
      title: MessageModel.getMessage(messages.tsmId, context, messages),
      description: _deviceModel.formatTime(messages.tsmTs, context),
      onPressed: () {},
      includeArrowIcon: false,
      includeIconBox: false,
      subtitleColor: InstallerColor.darkGreyColor,
      titleStyle: InstallerTextStyles.bodyText,
    ));
    items.add(const SizedBox(height: 6));
  }).toList();

  return items;
}
```

KUVIO 20. Viestien muuttaminen rajapinnan palautuksesta laitetietolaatikoksi

Haltianin sensorit lähettävät tyyppinsä mukaan viestin tunnistenumeroa. Esimerkiksi viestit 1000 – 2000 ovat yleisiä viestejä mitä jokainen laite lähettää. Viestit 12000-12999 ovat Thingsee Enviroment-sensorin lähettämiä viestejä. Viestien numeroinnin avulla pystymme erottelemaan viestit omiin kategorioihinsa ja näin pystymme helpottamaan tulevaisuudessa laitteiden viestien lisäämistä sovelluksessa viesteihin. Viestien konteksti ja sisältö haetaan MessageModel-luokkaa käyttäen. Luokan sisällä olemme erotelleet laitteiden viestit ja suodatetaan listasta viestit mitä näytetään käyttöliittymässä.

Kuviossa 21. on Thingsee Enviroment-sensorin viestien erottelu: viesti 12100 ilmoittaa laitteen mittaamat ilmanlaatuarvot. Thingsee Enviroment-sensori lähettää arvon vain jos se on muuttunut mittausvälin aikana. Välttääksemme null-arvon näyttämistä viestissä toteutettiin weatherInfo-muuttuja viestin rakentamista varten. Jos arvo on null sitä ei lisätä viestiin vaan InstallerListTile-widgetissä näytetään vain viestissä tulleet arvot.

```
String enviromentProfile(
    int tsmId, BuildContext context, DeviceMessages messages) {
  if (tsmId == 12100) {
    String weatherInfo = "";
    if (messages.airp != null) {
      num? airp = (messages.airp! / 100);
      weatherInfo += AppLocalizations.of(context)!
          .weatherInfoAirp(airp.toStringAsFixed(2));
    }
    if (messages.humd != null) {
      weatherInfo +=
          AppLocalizations.of(context)!.weatherInfoHumd(messages.humd);
    }
    if (messages.lght != null) {
      weatherInfo +=
          AppLocalizations.of(context)!.weatherInfoLght(messages.lght);
    }
    if (messages.temp != null) {
      weatherInfo +=
          AppLocalizations.of(context)!.weatherInfoTemp(messages.temp);
    }
    return weatherInfo;
  } else if (tsmId == 12101) {
    return AppLocalizations.of(context)!
        .magnetoInfo(messages.hall, messages.hallCount);
  } else if (tsmId == 12102) {
    return AppLocalizations.of(context)!.leakageResistance(messages.resistance);
  } else if (tsmId >= 12200 && tsmId < 12300) {
    return AppLocalizations.of(context)!.sensorConfig(tsmId);
  } else {
    return tsmId.toString();
  }
}
```

KUVIO 21. Ilmanlaatusensorin viestien erottelu lähdekoodissa

Kun viestit on tuotu laitetietolaatikoihin ja viestit on avattu käyttöliittymän komponentteja varten. Tämän jälkeen saamme viestit-sivulle kuvion 22. näkymän: viesteissä näkyy Thingsee Environment-sensorin mittausarvoja. Sivun toisessa viestissä näkyy eriteltynä laitteen viesti, jossa on ilmoitettu Ilmanpaine, ilmankosteus, valonmäärä ja lämpötila. Käyttäjä pystyy selaamaan laitteen lähettämiä viestejä liikuttamalla sivua ylöspäin tai alaspäin.

KUVIO 22. Viestit-sivu käyttöliittymässä

4 POHDINTA

Tässä opinnäytetyössä tavoitteena oli esitellä, kuinka sensorien asennussovelluksen moduuli on toteutettu käyttäen Googlen Flutteria ja Bloc-tilanhallintakirjaston käyttöä tietojen pyytämiseen Thingsee-rajapinnasta. Työssä tuotiin esiin Bloc-tilanhallintapaketin käyttöä sovelluksen rajapinnan toteutuksessa ja tilanhallinnassa. Työssä myös pyrittiin näyttämään sovellukseen tuotettu käyttöliittymä suunnittelijan suunnitelman mukaan.

Laitetietosivusta tuli suunnittelijan toteutuksen mukainen, ja sivun toteutus onnistui ilman suurempia yllätyksiä. Sivun teosta sai myös hyvin materiaalia työhön. Lisäosan värit ja eri widgetit ovat yhtenäisiä sovelluksen muiden widgettien kanssa. Viestit-sivulla viestejä on avattu käyttäjälle näyttäen esimerkiksi ilmanlaatutiedot Thingsee Air -sensorista, tai tukiasema näyttää mobiiliverkon vahvuuden käyttäjälle. Sivun suurin heikkous on pitkä latausaika. Sivu käyttää tällä hetkellä neljää eri rajapinnan hakua saadakseen sivulla näytettävät tiedot tietokannasta. Latausaikaa sain optimoitua muuttamalla logiikkaa niin, että laitetietosivu voidaan näyttää ennen kuin viestit ovat ladanneet, ja tilanhallinta aloittaa viestien haun viimeisempänä. Viestejä haetaan viisikymmentä viimeisintä, joten se vie tilanhallinnassa eniten aikaa muihin verrattuna. Jos laitetietosivun tiedot saataisiin käyttämällä vain yhtä hakua, se parantaisi latausaikaa jo huomattavasti.

Laitesivun jatkokehittämisessä lisäisin datalle grafiikkaa joko erillisellä sivulla tai viestit-sivulla viestistä ponnahdusikkunalla avaamalla. Tällä hetkellä käyttäjä näkee vain viimeisimmät tiedot viidenkymmenen viestin joukosta, mutta graafeilla pystyttäisiin antamaan käyttäjälle selkeämpi kuva esimerkiksi Thingsee-Environment-sensorin lämpötilasta ja muista arvoista. Jatkokehityksessä olisi myös hyvä huomioida usean rajapinnan käyttö sivulla ja sen nopeuttaminen. Laitteen tilan logiikka on myös vähän yksinkertainen, koska poispäältä -tila tulee näkyviin vasta kaksi tuntia tilan muutoksen jälkeen. Logiikkaa olisi mahdollista parantaa esimerkiksi tekemällä tarkistuksen laitteelle, jos viimeisimmästä viestistä on kulunut tarpeeksi kauan.

Työtä tehdessä opin paljon. Pääsin ensimmäistä kertaa oikeaan sovelluskehitysprojektiin rakentamaan mobiilisovellusta ohjelmoijana. Sovelluksen teon aikana tutuksi tuli sovellusarkkitehtuuri ja sen rakentaminen. Tilanhallinnan käyttö ja Bloc-paketin lisääminen

sovellukseen oli todella mielenkiintoinen aihe ja niihin tutustuminen sekä kaavan opettelu tulevat olemaan tulevaisuuden projektien tekemiseen todella hyödyllisiä taitoja.

LÄHTEET

1. Felix Angelov. Core Concepts (package:flutter_bloc). Hakupäivä 10.10.2022 https://bloclibrary.dev/#/flutterbloccoreconcepts
2. Gerrit 2022. Gerrit Code Review Product Overview https://gerrit-documentation.storage.googleapis.com/Documentation/3.6.2/intro-quick.html
3. Google Developers 2018. Flutter / AngularDart – Code sharing, better together (DartConf 2018). Hakupäivä 29.10.2022. https://www.youtube.com/watch?v=PLHln7wHgPE
4. Flutter 2021 a. Flutter Development Tools Visual Studio Code. Hakupäivä 12.10.2022. https://docs.flutter.dev/development/tools/vs-code
5. Flutter 2021 b. Flutter Development Ui Introduction to widgets. Hakupäivä https://docs.flutter.dev/development/ui/widgets-intro
6. Flutter 2022. Flutter Development packages & plugins. Hakupäivä 10.8.2022. https://docs.flutter.dev/development/packages-and-plugins/developing-packages
7. Haltian 2021 a. Haltian Environment Profile (12000-12999). Hakupäivä 20.9.2022. https://support.haltian.com/knowledgebase/environment-profile-12000-12999/
8. Haltian 2021 b. Haltian Presence and Occupancy Profile (13000-13999). Hakupäivä 20.9.2022. https://support.haltian.com/knowledgebase-category/api/thingsee-iot-api/thingsee-messages/
9. Haltian 2021 c. Haltian support open services API. Hakupäivä 2.9.2022. https://support.haltian.com/knowledgebase/thingsee-open-service-api/
10. Haltian 2022. Haltian Oy. Hakupäivä 08.07.2022 https://haltian.com/resources/
11. Google 2022. The Dart type system. Hakupäivä 23.10.2022. https://dart.dev/guides/language/type-system
12. T. Bray, Ed. 2017. The JavaScript Object Notation (JSON) Data Interchange Format. Hakupäivä 2.9.2022. https://www.rfc-editor.org/rfc/rfc8259
13. Dan's tools 2022. What is the unix time stamp? Hakupäivä 13.10.2022. https://www.unixtimestamp.com/