Minttu Koponen

Mobiiliratkaisu teollisuuden laboratorioiden tiedonhallintajärjestelmässä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

23.9.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Minttu Koponen Mobiiliratkaisu teollisuuden laboratorioiden tiedonhallintajärjestelmässä 37 sivua 23.9.2018
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Ohjelmistotuotanto
Suuntautumisvaihtoehto	Tieto- ja viestintätekniikka
Ohjaajat	Lehtori Juha Kämäri Lehtori Jussi Alhorinne Tiiminvetäjä Antti Kauhanen

Insinöörityön tavoitteena oli kehittää Android-pohjainen mobiilisovellus osaksi yrityksen laboratorion tiedonhallintajärjestelmää (LIMS, Laboratory Information Management System) sekä tuottaa yritykselle mobiilisovelluksen kehittämiseen tarvittavaa tietoa. Työ tehtiin yrityksen Software Point Oy:n toimeksiannosta heidän asiakkaalleen, joka on iso suomalainen teollisuusalan laboratorio.

Osana insinöörityötä selvitettiin ääniohjauksen käyttämistä mobiilisovelluksessa. Tästä todettiin, että ääniohjauksen käyttäminen halutulla tavalla vaatisi paljon enemmän työtä, kuin mitä oletettiin, joten ääniohjaus jätettiin sovelluksesta pois. Merkittävimmät ääniohjauksen toteuttamista vaikeuttavat tekijät olivat sen käyttö ilman verkkoyhteyttä ja suomen kieli. Lisäksi ääniohjauksen haluttu toiminnallisuus ilman erillistä, sen aktivoivaa painiketta olisi lisännyt toteuttamiseen tarvittavaa työmäärää huomattavasti.

Työn teoriaosuudessa esitellään LIMS sekä käydään läpi Android-sovelluskehityksen ja ääniohjauksen teoriaa. Myös mobiilisovelluksen kehittämisen suunnittelun tulokset esitetään. Lopuksi käydään läpi toteutuksen vaiheita ratkaisuineen.

Insinöörityön tuloksena syntyi asiakkaan määritysten mukaisesti toimiva Android-pohjainen mobiilisovellus. Sovelluskehityksessä huomioitiin sen uudelleenkäytettävyys ja muokattavuus asiakkaan LIMS-järjestelmän kannalta. Projektin myötä yritykselle kertyi kokemusta Android-pohjaisten mobiilisovellusten kehitystyöstä.

Avainsanat	Android, LIMS, ääniohjaus

Author Title Number of Pages Date	Minttu Koponen Mobile Solution for Laboratory Information Management System in Industry 37 pages 23 September 2018
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Specialisation option	Software Engineering
Instructors	Juha Kämäri, Senior Lecturer Jussi Alhorinne, Senior Lecturer Antti Kauhanen, Team Leader

Objective of this thesis was to develop an Android based mobile application to be used as a part of industry's laboratory information management system (LIMS, Laboratory Information Management System). In addition, the objective was to increase the company's, Software Point Oy, knowledge of Android application development. The mobile application was developed for Software Point's customer, a big industry laboratory in Finland.

Part of this thesis was to investigate the possibility of using voice control in the mobile application. As an outcome, the voice control was left out because implementing it would have been more time consuming than was expected. The main reasons for the implementation difficulties were that it had to work in Finnish language in an offline environment. Furthermore, it had to be activated with a custom keyword, and should not need a button press to activate.

In the theory part of this thesis, LIMS, Android development and voice control are explained. After the theory part, the practical part explains the outcomes of the planning and the implementation of this Android application. A prototype of the application is presented.

As an outcome, an Android application was developed according to the customer's qualifications. During the development of this Android software, special attention was paid to the reutilization of the software and the modifiability of the software through the customer's LIMS. With this software development process, the company gained experience of the development of Android applications.

Keywords	Android, LIMS, voice recognition

Sisällys

Lyhenteet

LIMS	Laboratory Information Management System. Ohjelmisto, jonka avulla hallinnoidaan laboratorion tietoja.
API	Application Programming Interface. Ohjelmointirajapinta on määritelmä, jonka mukaan ohjelmat voivat keskustella keskenään.
XML	Extensible Markup Language. Merkintäkielen standardi.
SQL	Structured Query Language. Standardoidu kyselykieli, jolla voi tehdä relaatiotietokantaan hakuja, muutoksia ja lisäyksiä.

1 Johdanto

Androidin suosio on kasvanut viimeisen kymmenen vuoden aikana räjähdysmäisesti, ja nykypäivänä Android-mobiilisovelluksia käytetään yhä enemmän myös työelämässä. Yrityksessä Software Point Oy on tiedostettu asia ja lähdetty kehittämään mobiilisovellusta osaksi heidän laboratorion tiedonhallintajärjestelmää. Mobiilisovelluksen kehityksen ohella syntyi myös tämä insinöörityö.

Software Point Oy on suomalainen yritys, yhdysvaltalaisen Labvantage-konsernin tytäryhtiö, joka myy ja räätälöi asiakkailleen laboratorion tiedonhallintajärjestelmää (LIMS). Yritys on toiminut jo vuodesta 1992 alkaen. Software Pointin LIMS toimii selainpohjaisesti edellyttäen toimiakseen verkkoyhteyden. Asiakkaalla, jolle mobiilisovellus kehitetään, on tarve kerätä näytetietoja ilman verkkoyhteyttä olevalla alueella. Ratkaisuna tähän on erillinen ilman verkkoyhteyttä toimiva mobiilisovellus, joka liitetään asiakkaan LIMS-järjestelmään.

Insinöörityön tavoitteena on kehittää Android-pohjainen mobiilisovellus osaksi yrityksen laboratorion tiedonhallintajärjestelmää. Kehitystyön lisäksi insinöörityöllä halutaan tuottaa yritykselle tietoa mobiilisovelluksen kehittämisestä, sillä tämä on yleisesti ottaen uusi asia Software Point Oy:ssä. Mobiilisovelluksen kehitys toteutetaan yrityksen Software Point Oy:n toimeksiannosta heidän asiakkaalleen, isolle suomalaiselle teollisuusalan laboratoriolle.

Työ alkaa teoriaosuudella, jossa esitetään projektiin liittyvien aihepiirien käsitteitä ja puheentunnistuksen teoriaa. Ensimmäisenä teoriaosuudessa esitellään LIMS, jotta lukija ymmärtää paremmin mobiilisovelluksen ympäristön. Toiseksi käydään läpi lyhyesti Android-sovelluskehityksen ominaisia peruspiirteitä. Yksi työn tavoitteista on selvittää ääniohjauksen soveltuvuutta mobiilisovelluksen yhteyteen, mikä parantaisi sovelluksen käytettävyyttä. Viimeisenä teoriaosuudessa esitellään ääniohjauksen teoriaa ja sen toiminnan vaiheita.

Teoriaosuuden jälkeen keskitytään mobiilisovelluksen kehityksen vaiheista suunnitteluun ja toteutukseen. Suunnittelussa esitetään koko sovelluksen suunniteltu toiminta-prosessi ja sovelluksen eri näkymien halutut toiminnot. Tässä osiossa esitellään lisäksi sovelluksen arkkitehtuuri sekä käytössä oleva teknologia.

Toteutuksessa käydään läpi erilaisia ongelmia ja ratkaisuja, joihin kehityksen aikana päädyttiin. Yksi näistä on ääniohjauksen käyttöönottamiseen vaativa työ sekä tähän liittyvät ratkaisut. Mobiilisovelluksesta esitellään tämänhetkinen prototyyppi. Tämän esittelyn yhteydessä käydään läpi erilaisia ratkaisuja, joihin muun muassa käyttöliittymän suunnittelussa päädyttiin.

Viimeisenä yhteenvetoluvussa tarkastellaan tuloksena saatua mobiilisovellusta. Kehitettyyn mobiilisovellukseen esitetään jatkokehitysajatuksia huolimatta siitä, että sovellus ei vielä kirjoitusvaiheessa ole täysin valmis.

2 Teoriaa

2.1 LIMS

Laboratorion tiedonhallintajärjestelmä LIMS on lyhenne englanninkielisistä sanoista Laboratory Information Management System. LIMS on ohjelmistopohjainen ratkaisu, johon kerätään tietoa muun muassa laboratorion henkilöstöstä, asiakkuuksista, näytteistä, laitteista, menetelmistä, laadunvarmistuksesta, raportoinneista ja laskutuksesta. LIMSin avulla voidaan hallinnoida kaikkea tätä tietoa.

Ensimmäiset LIMS-sovellukset kehitettiin 1960-luvun loppupuolella mutta lähinnä yrityksien omiin tarpeisiin. Tietotekniikan yleistyessä 1980-luvun alkupuolella kehitettiin ensimmäiset kaupalliset LIMS-tuotteet. Markkinoilla olikin vuonna 1984 jo 8 eri kaupallista LIMS-tuotetta tuoden LIMSin myös sellaisten yritysten ulottuville, joilla ei ollut halua kehittää ohjelmistoja omakustanteisesti. Ensimmäisillä LIMS-järjestelmillä hoidettiin lähinnä laboratorioiden laskentaa ja laitteiden automaatiota. Kummatkin ovat tarkkoja ja aikaa vieviä tehtäviä, jotka tietokone hoiti sekä nopeammin että tarkemmin. Tietotekniikan kehittyessä ja kustannusten pienentyessä LIMS-järjestelmät ovat yleistyneet laajalti laboratorioiden käyttöön maailmalla. LIMS-järjestelmän käyttö on myös laajentunut koko laboratorion tiedon hallintaan, raportointiin sekä laskutukseen. [1, s. 2–4.]

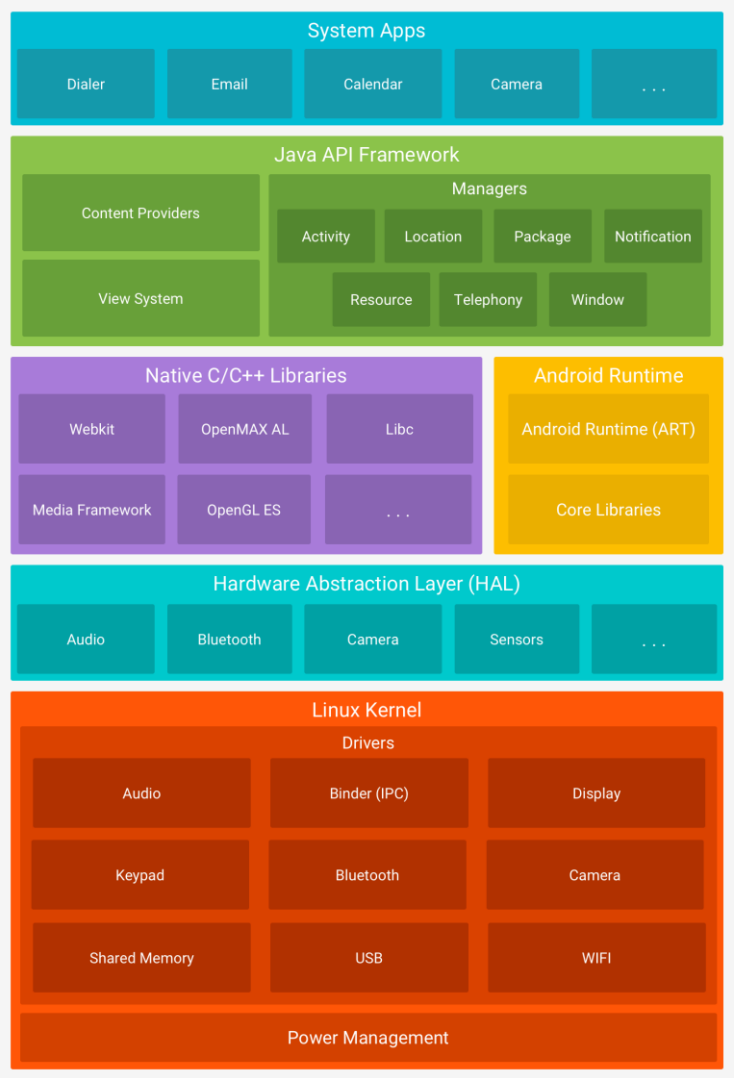
2.2 Android

Androidin historia alkaa vuodesta 2003, jolloin Android Inc. perustettiin. Google osti yrityksen vuonna 2005, ja ensimmäinen Android älypuhelin tuli markkinoille vuonna 2008. Androidin suosio kasvoi nopeasti ja Android onkin tällä hetkellä yleisin käyttöjärjestelmä älypuhelimissa. [2, s. 11–12.]

Android sovelluskehitys on laaja aihealue, josta tässä osiossa pyrin tiivistämään sen perusteet siltä osin, kun ne ovat tarpeen tämän projektin ymmärtämisen kannalta. Aluksi käyn läpi Android-sovelluksen arkkitehtuurin ja projektin rakenteen. Ne ovat yleisiä kaikilla Android-projekteilla. Seuraavaksi kerron Android-aktiviteeteista, jotka ovat keskeisessä osassa sovelluksen toiminnan hallinnassa. Lopuksi esittelen Androidin lokikirjoitusjärjestelmän ohjelmoinnin apuvälineenä.

2.2.1 Android-arkkitehtuuri

Googlen omistama Android-käyttöjärjestelmä on Linux Kernel -ytimen päälle rakennettu. Arkkitehtuuri on kokonaisuudessaan kuvattu kuvassa 1. Android tarjoaa kokonaan avoimen lähdekoodin kehitysalustan, jossa sovelluksia voidaan kirjoittaa Java ohjelmointikielellä. Suorittaakseen Java-luokkia Androidissa, ne täytyy esikääntää ART (Android Runtime) -virtuaalikoneella suoritettaviksi Dalvik executables -tiedostoiksi [3]. Android sisältää vakiona SQLite-tietokannan, ja sovelluksen toteutuskehyksessä kehittäjällä on käytössä täysi Android API (Application Programming Interface) [4]. API on ohjelmointirajapinnan määritelmä, jonka mukaan ohjelmat voivat keskustella keskenään. Sovelluksia käytetään pääsääntöisesti erilaisilla kannettavilla laitteilla kuten älypuhelimilla, tableteilla ja kämmentietokoneilla. [2, s.12–16.]

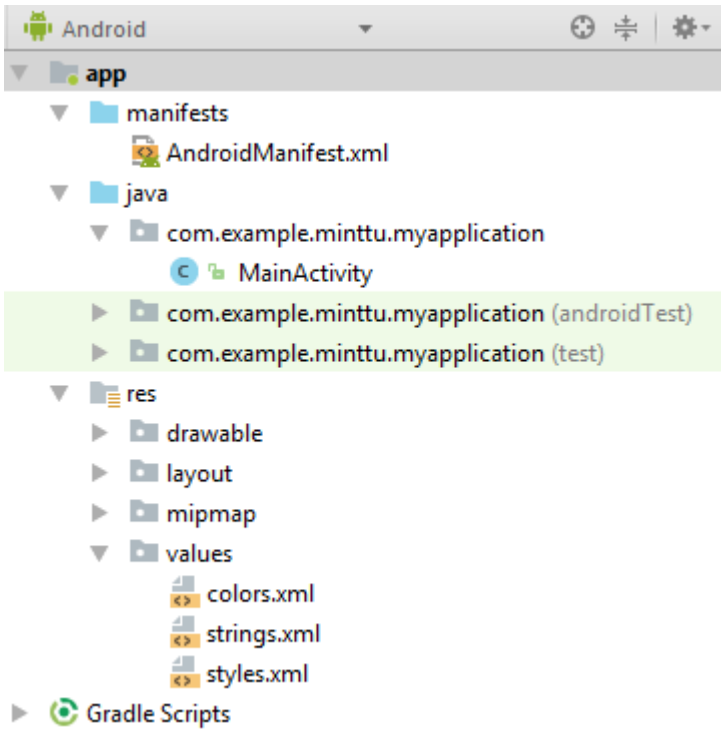


Kuva 1. Android-arkkitehtuuri [3].

Sovelluskehityksessä voidaan hyödyntää Androidin mukana tulevia perussovelluksia kuten Google Calendar, Google Maps ja Chrome sekä laitteen tehdasasennettuja ominaisuuksia kuten kameraa ja viivakoodinlukijaa. Näiden lisäksi voidaan hyödyntää käyttäjän itse asentamia sovelluksia. Niitä saa muun muassa Google Play -kaupasta. Android mahdollistaa sovellusten välisen tiedonvaihdon, joista osaan toki täytyy käyttäjältä pyytää lupa.

2.2.2 Androidin projektirakenne

Android-sovelluskehitykseen voidaan käyttää esimerkiksi Android Studiota, joka on Android-käyttöjärjestelmän virallinen ohjelmointiympäristö [5]. Luomalla Android Studiossa uuden Android-projektin saadaan projektille kuvassa 2 näkyvä hakemistorakenne. Projektin alihakemistot ovat nimeltään *manifests, java* ja *res.* Kyseinen hakemistorakenne on vakio kaikille Android-projekteille.



Kuva 2. Android-projektin hakemistorakenne.

Hakemiston *manifests* alta löytyy XML (Extensible Markup Language) -merkintäkielen muotoinen tiedosto AndroidManifest.xml, jossa määritellään kaikki sovelluslogiikan toteuttavat komponentit. Manifestitiedostossa määritellään muun muassa sovelluksen

käynnistävä luokka, API-versiot, käyttöoikeudet ja ulkoiset kirjastot. Android-sovelluksen käynnistävä luokka on mainittava manifestitiedostossa, sillä Androidissa ei ole vastaavaa aloittavaa kohtaa kuten Javan main-metodi.

Java-luokat sijoitetaan *java*-hakemiston alle. Luokat voivat Java-olio luokkien lisäksi olla neljää eri tyyppiä; aktiviteetti (Activity), palvelu (Service), sisällöntarjoaja (Content Provider) tai vastaanottaja (Broadcast receiver). Aktiviteetti määrittelee näkymän ja sen toiminnot. Taustalla suorittava komponentti, joka ei ole vuorovaikutuksessa käyttäjän kanssa, on palvelu. Sisällöntarjoajan avulla voidaan käsitellä jaettua sisältöä. Vastaanottaja välittää tietoa järjestelmätasoisesti, sovellusten välisesti sekä sovelluksen sisäisesti. Java-hakemiston alla olevien alihakemistojen ja Java-luokkien järjestys ei ole etukäteen määritelty, vaan kehittäjä voi päättää sen itse. [2, s. 33–36.]

Resurssihakemisto *res* on jaettu edelleen alihakemistoihin. Hakemiston *drawable* ja *mipmap* alle tulevat kuvatiedostot. Kuvat voidaan määrittää usealle eri näytön tarkkuudelle erikseen. Itse kuvioita, kuten neliöitä tai kolmioita, voidaan määrittää myös XML-tiedostoina. Aktiviteettien käyttämät näkymät määritellään XML-tiedostoina *layout*-hakemiston alle. Jokaiseen näkymään määritellään sen rakenne ja sen käyttämät komponentit. Näkymien komponentteja on mahdollista myös lisätä ajonaikaisesti. Hakemiston *values* alta löytyy vakiona kolme XML-tiedostoa. Tiedostoon colors.xml määritellään ohjelman käyttämät värit, strings.xml-tiedostoon merkkijonot ja styles.xml-tiedostoon ohjelman tyylimääritykset.

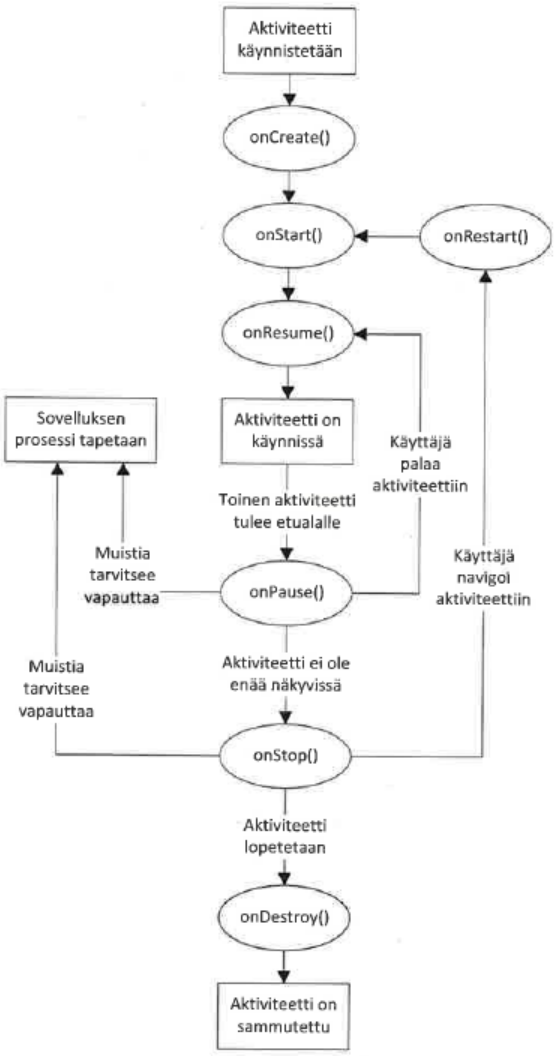
2.2.3 Aktiviteetti

Aktiviteetit ovat Android-sovelluksessa Java-luokkia, jotka vastaavat käyttöliittymän toiminnoista ja näkymistä. Aktiviteettien avulla hallitaan ohjelman kulkua näkymästä toiseen ja sovelluksesta toiseen. Aktiviteetti luokka perii luokan Activity, joten luokassa voidaan ylikirjoittaa Activity-luokan metodeja. Esimerkkikoodissa 1 on aktiviteettiluokan perusrakenne.

```
package example.demo;
import android.app.Activity;
import android.os.Bundle;

public class Aktiviteetti extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState){
          super.onCreate(savedInstanceState);
    }
}
```

Esimerkkikoodi 1. Aktiviteettiluokka.

Aktiviteetin elinkaari, joka näkyy kuvassa 3, alkaa aina onCreate-metodista. Metodia onCreate kutsutaan automaattisesti elinkaaren alussa. Sitä kutsutaan vain yhden kerran aktiviteetin ensimmäisen käynnistyksen yhteydessä. Metodissa voidaan luoda esimerkiksi tietoliikenneyhteyksiä ja näkymän komponentteja. Aktiviteetin elinkaari loppuu onDestroy-metodiin. Siinä voidaan sulkea luodut yhteydet, jotta ne eivät jää tarpeettomasti auki. Aktiviteetin elinkaaren aikana siirtyessä vaiheesta toiseen, kutsutaan kuvassa 3 näkyviä metodeja aina automaattisesti. [6.]



Kuva 3. Aktiviteetin elinkaari [2, s. 45].

Käyttäjälle näkyvä osa aktiviteetista tapahtuu onStart- ja onStop-metodien välillä. Metodissa onStart päivitetään käyttöliittymän tiedot. onStop-metodissa keskeytetään tarpeeton toiminta sekä tallennetaan tilamuutokset. Käyttäjälle aktiivinen osa aktiviteetista on puolestaan onResume- ja onPause-metodien välillä. Metodeissa käynnistetään ja vastaavasti pysäytetään aktiviteetin toimintoja kuten käyttöliittymä, säikeet ja prosessit. [2, s. 43–44.]

Sovelluksessa aktiviteetista toiseen tai applikaatiosta toiseen voidaan siirtyä Intent-olion avulla. Intent-olio välittää tiedon käynnistettävästä aktiviteetista. Intent-olio luodaan ja aktivoidaan alla olevan esimerkkikoodin 2 mukaisesti. Intent-oliota luodessa sille annetaan parametrina aktiviteetti, joka halutaan käynnistää. Metodille startActivity annetaan parametrina kyseinen Intent-olio, jolloin aktiviteetti käynnistyy.

```
Intent intent = new Intent(this, Aktiviteetti.class);
startActivity(intent);
```

Esimerkkikoodi 2. Intent-olion luominen ja aktiviteetin käynnistys.

Intent-olion mukana voidaan siirtää tietoa aktiviteetille, ja aktiviteetilta voidaan palauttaa tietoa. Tiedon lähettäminen Intent-olion mukana toteutetaan putExtra-metodilla, jolloin parametriksi annetaan merkkijonot avain ja arvo. Esimerkkikoodissa 3 on lähetetty merkkijono *tieto* avaimella *lähtöavain*. Kun aktiviteetista halutaan palauttaa tietoa, aktiviteetti aloitetaan kutsumalla startActivityForResult-metodia. Sille annetaan parametriksi aktiviteetin koodi. Palautettu tieto käsitellään onActivityResult-metodilla, jossa verrataan aktiviteetilta palautuvaa koodia. Kun palautunut aktiviteetti on kohdistettu oikein, hasExtra-metodilla voidaan tarkistaa, onko aktiviteetilta palautunut haluttua tietoa.

```
final int AKTIVITEETTI_1 = 1;
Intent intent = new Intent(this, Aktiviteetti.class);
intent.putExtra("lähtöAvain", "tieto" );
startActivityForResult(intent, AKTIVITEETTI_1);

@Override
protected void onActivityResult(int requestCode,
int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode,
    data);
    if(resultCode == Activity.RESULT_CANCELED){
          //tee jotain
    }
    if(requestCode == AKTIVITEETTI_1) {
          if(data.hasExtra("paluuAvain")) {
             String paluuAvain =
             data.getStringExtra("paluuAvain");
          }
   }
}
```

Esimerkkikoodi 3. Tiedonsiirto aktiviteetistä toiseen.

Sovellus voi tarvita useaa eri aktiviteettia yhteen käyttäjän haluaman toiminnon suorittamiseen. Käytössä olevat aktiviteetit säilötään pinorakenteessa siinä järjestyksessä, kun ne on avattu. Käynnissä oleva, käyttäjälle näkyvä aktiviteetti, on aina pinon päällimmäisenä. Kun aktiviteetti suljetaan, se poistetaan pinosta, ja seuraavaksi päällimmäinen aktiviteetti palautetaan näkyväksi. Pinon päällimmäinen aktiviteetti poistuu myös käyttäjän painaessa laitteen takaisin (back) -painiketta, jolloin pinon toiseksi päällimmäinen aktiviteetti palautetaan näkyväksi. [7.]

2.2.4 Androidin loki

Sovelluksen virheenjäljittämistä helpottamaan on Androidissa sisäänrakennettu lokikirjoitusjärjestelmä nimeltä Logcat. Sovelluskehittimen Logcat-näytölle tulee ohjelman suorituksenaikaisia viestejä itse järjestelmästä sekä ohjelmoijan pyytämiä tulosteita. Tulosteet saadaan Log-luokan metodeilla määrittelemällä kaksi merkkijonoparametria; tagi ja viesti. Log-luokan metodeja on useita, joista taulukossa 1 luetelluilla metodeilla voidaan luokitella tulosteet eri tasoihin. [8.]

Taulukko 1. Android Log-luokan lokikirjoitusmetodit [8].

Metodi	Luokitus
Log.v(String tagi, String viesti)	Tekstiä (VERBOSE)
Log.d(String tagi, String viesti)	Virheenetsintä (DEBUG)
Log.i(String tagi, String viesti)	Tiedoksi (INFO)
Log.w(String tagi, String viesti)	Varoitus (WARN)
Log.e(String tagi, String viesti)	Virhe (ERROR)

Esimerkkikoodissa 4 on haluttu tulostaa ohjelmasta haettu arvo tiedoksi-luokituksella. Tiedoksi-luokitus saadaan käyttämällä Log-luokan metodia i. Tagiksi on määritelty WSP, ja viestin yhteyteen tulostetaan viestin lisäksi arvo.

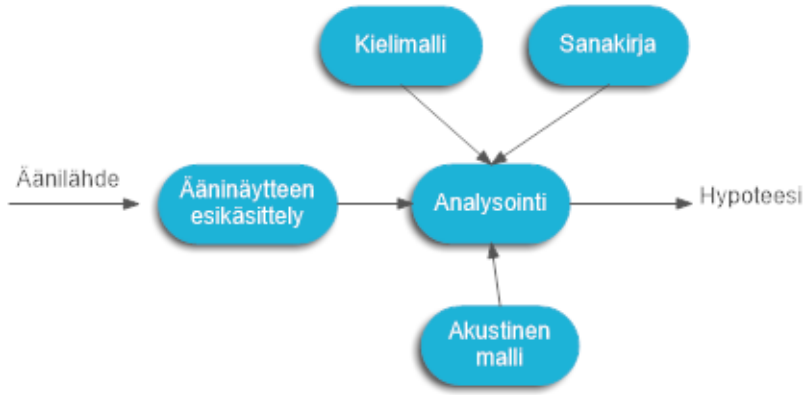
```
Log.i(”WSP”, ”haettu arvo on: ” + haeArvo());
```

Esimerkkikoodi 4. Android Log-luokan avulla tulosteen kirjoittaminen.

Sovelluskehittimen Logcat-näytöllä olevista viesteistä voidaan suodattaa näkyväksi vain halutut viestit. Suodatuksen kriteereinä voidaan käyttää viestin luokan tasoa ja viestissä esiintyvää tekstiä kuten tagin tekstiä. Lokiviestien suodattaminen helpottaa huomattavasti halutun viestin löytämistä ja nopeuttaa virheenetsintää.

2.3 Ääniohjaus

Ääniohjaus on nykypäivänä kehittyvää teknologiaa, jota tutkitaan myös tässä projektissa käytettäväksi. Sen toteuttamiseksi tarvitaan mahdollisimman luotettava puheen tunnistamisen prosessi, jonka vaiheet on esitelty kuvassa 4.



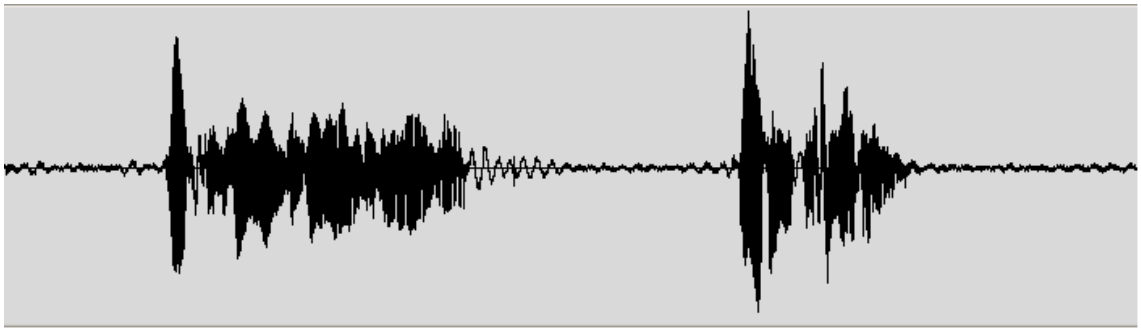
Kuva 4. Puheentunnistusvaiheet.

Puheentunnistus alkaa syntyneestä äänilähteestä, joka ensimmäisessä vaiheessa pilkotaan ja esikäsitellään. Seuraavaksi esikäsitelty äänilähde analysoidaan sanakirjan, kielimallin ja akustisen mallin avulla. Siitä lasketaan hypoteesi, joka on todennäköisin analyysin perusteella muodostettu sana tai lause. Puheentunnistus toteutetaan laskemalla todennäköisyyttä, eikä nykypäivän malleilla ole mahdollista tunnistaa puhetta 100 %:n varmuudella. [9.]

Äänilähteen, sen esikäsittelyn ja analysoinnin käsittelen tarkemmin seuraavissa osioissa. Tarkoitus on tuoda esille puheentunnistuksen vaativuutta ja siihen liittyvää epävarmuutta, jotta ääniohjauksen haasteellisuus on helpompi ymmärtää.

2.3.1 Ääninäytteen esikäsittely

Puheentunnistaminen alkaa äänilähteestä. Ääni on ilmanpaineen nopeata värähtelyä, joka voidaan graafisesti kuvata ilmanpaineen vaihteluna ajan funktiona kuten kuvassa 5. Samat äänteet ääntyvät eri puhujilla ja eri kerroilla eri tavalla, aiheuttaen äänisignaaliin merkittävää vaihtelua. Saman sanan graafinen kuva voi olla puhujasta riippuen hyvinkin erinäköinen. Vaihtelu riippuu muun muassa puhujan hengityksestä ja puhujan fyysisistä ominaisuuksista, kuten kurkunpään, kielen, huulien ja muiden äänen muodostumiseen vaikuttavista tekijöistä. [10, s. 105–106.]



Kuva 5. Nauhoitettu äänisignaali. Kuvassa näkyy ilmanpaineen vaihtelu ajan funktiona. [9.]

Äänilähteen esikäsittelyssä ääni pilkotaan pienempiin osiin ja niiden ominaisuus vektoreihin. Puhe on jatkuvaa äänivirtaa, joka koostuu tasaisista ja epätasaisista äänivirran tiloista. Epätasaiset tilat koostuvat peräkkäisistä äänteistä ja äänteiden välissä olevista siirtymäjaksoista. Puheentunnistuksessa puheen äänivirta jaetaan pienempiin, 20–30 millisekunnin aikaikkunoihin esikäsiteltäväksi. Äänisignaalin aikaikkunoista lasketaan sen eri taajuuksien suhteita eli spektrejä, joiden ominaisuus vektorit siirretään analysoitavaksi. [11, s. 122.]

2.3.2 Ääninäytteen analysointi

Ääninäytteen analysoimiseen tarvitaan kolme mallia: sanakirja, kielimalli ja akustinen malli. Näiden kolmen mallin avulla ääninäytteestä muodostetaan sanoiksi muutettu hypoteesi.

Sanakirja määrittää sanaston sanat ja niiden ääntämisen mallin. Yhdellä sanakirjan sanalla voi olla monta ääntämisen mallia, koska sama sana voidaan ääntää hyvinkin

eri lailla riippuen esimerkiksi puhujan murteesta. Esimerkkiote sanakirjasta on kuvassa 6, jossa on ääntämisen mallit englanninkieliselle sanalle *the*. Eri ääntämisen malleille voidaan lisätä painoarvo vastaamaan niiden esiintymistiheyttä. [11, s. 123.]

```
the TH IH
the(2) TH AH
```

Kuva 6. Ote sanakirjan englanninkielisestä sanasta the, ja sen ääntämisen mallit [12].

Kielimalleja voi olla monen tyylisiä. Yhdessä mallissa voidaan rajoittaa tunnistus esimerkiksi tiettyihin lauseisiin, jos sovelluksessa käyttäjältä odotetaan vain tarkoin määriteltyjä lausevaihtoehtoja. Toisessa mallissa voidaan määrittää todennäköisyyksiä tiettyjen sana yhdistelmien esiintyvyydelle kielessä, jolloin kielimalli laskee ehdollisen todennäköisyyden seuraavalle odotettavissa olevalle sanalle, kun edelliset sanat tunnetaan. Kielimallin valintaan vaikuttaa puheentunnistuksen haluttu toiminta itse sovelluksessa: halutaanko sovelluksen kuuntelevan koko ajan puhujaa vai aktivoidaanko sovellus vain tiettyjen käskyjen kuuntelemiseksi. [11, s. 124–125.]

Akustinen malli on matemaattinen malli, joka laskee todennäköisyyttä sille, että äänifragmentti vastaa tiettyä foneemia tai sanaa [13]. Kyseistä todennäköisyyttä lasketaan yleisemmin Markovin piilomallilla HMM (Hidden Markov's Model). Markovin piilomalli on tilastollinen malli. Se perustuu äännefragmentin analysointiin kontekstiriippuvaisesti. Yleisesti Markovin piilomallista käytetään kolmen foneemin mallia, jolloin mallissa otetaan huomioon äänneyksikön edeltävä ja sitä seuraava äänne. Markovin piilomalli analysoi äänifragmentin kahdella tasolla. Ensimmäinen on piilotaso, jossa äänilähteen ominaisuusvektoreiden aika dynaamiset ominaisuudet määritellään Markovin ketjumallilla. Toisella tasolla määritetään äänilähteen ominaisuusvektoreiden tilastollinen vaihtelu tilariippuvaisella todennäköisyyden tiheysfunktiolla. Yleisemmin todennäköisyyden tiheysfunktion laskemiseen käytetään Gaussian mixture model (GMM) -mallia. [11, s. 123–124.]

Hyvän puheentunnistusmallin täytyy tunnistaa puhetta riittävällä tarkkuudella ja riittävän nopeasti [11, s. 125]. Kolmen mallin yhteenlaskettu tulos on äänilähteelle muodostettu hypoteesi, joka on todennäköisin tunnistus äänilähteestä. Koska virheellisiä tunnistuk-

sia voi esiintyä, täytyy käytössä arvioida, onko puheentunnistuksen tarkkuus ja nopeus riittävä haluttuun toimintoon.

Puheentunnistaminen on vaikeampaa kielillä, joissa on monimutkaisia sanojen taivutuksia. Suomen kieli kuuluu näihin. Suomen kielessä taivutetun sanan perusosa tai kanta voivat muuttua taivutuksessa vaikeuttaen oikean sanan tunnistamista. Tämän takia tarvitaankin laajempi sanasto, jossa on huomioitu taivutetut sanamuodot. Monimutkaisen kielen sanasto voi käytännössä olla jopa kymmenkertainen englannin sanastoon verrattuna. Laajempi sanasto tarkoittaa, että samankaltaisia sanoja on enemmän. Tällöin tunnistaminen on hitaampaa ja todennäköisyys sille, että sana tunnistetaan väärin, kasvaa. Laajemman sanaston myötä myös akustisen mallin opettamiseen tarvittavaa aineiston kokoa täytyy suurentaa. [13, s. 13–17.]

3 Suunnittelu

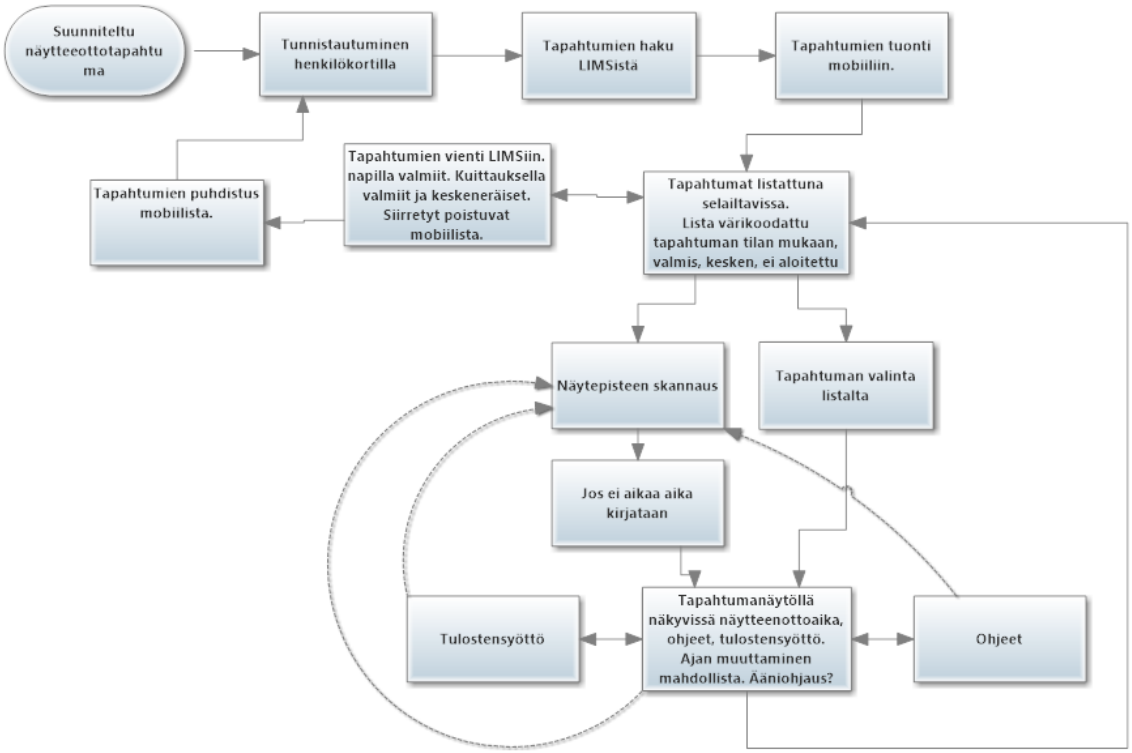
Tämän projektin lähtötilanne on se, että asiakkaalla on yrityksen Software Point Oy:n LIMS käytössä. Asiakkaan LIMS-järjestelmään tehdään kannettavalla kämmentietokoneella käytettävä mobiilisovellus. Mobiilisovellukseen tullaan alussa synkronoimaan tietoa verkon kautta asiakkaan LIMS-järjestelmästä. Tämän jälkeen sovellusta käytetään ilman verkkoyhteyttä olevissa tiloissa tietojen keräämiseen. Käytön lopuksi kerätyt tiedot lähetetään verkon kautta asiakkaan LIMS järjestelmään.

Projekti aloitettiin määrittelemällä asiakkaan vaatimukset. Tulokset kirjattiin kahteen erilliseen dokumenttiin nimiltään toiminnallinen määrittely ja tekninen määrittely. Seuraavissa osioissa on kuvattu sovelluksen haluttu toiminnallisuus, sen arkkitehtuuri sekä sovelluskehityksessä ja käytössä tarvittava teknologia.

3.1 Toiminnallisuus

Toiminnallisuuden yksi keskeinen asia on laitteen ja ohjelman käytettävyys. Ohjelman käyttäjällä tulee olemaan päällä suojavarusteita, joista erityisesti suojakäsineet vaikeuttavat laitteen kosketusnäytön käyttöä. Koska kosketusnäytön käyttö suojakäsineillä on hankalaa, tulee kaikkia ohjelman toimintoja voida ohjata laitteen kiinteän näppäimistön kautta. Kannettavaa mobiililaitetta käytetään enimmäkseen yhdellä kädellä, mikä tulee myös huomioida toteutuksessa.

Toinen toiminnallisuudelle asetettu vaatimus on sen käytettävyyden yksinkertaisuus. Kukin toiminto tulee toteuttaa siten, että käyttäjän tarvitsee koskea laitteeseen mahdollisimman vähän. Käyttäjäinteraktioiden minimointi toteutetaan muun muassa automaattitallennuksilla. Käyttöliittymä toteutetaan suomenkielisenä, ja se tulee sisältämään kuvassa 7 suunnitellun toiminnallisuuden. Toiminnallisuuden eri vaiheet on selitetty tarkemmin seuraavissa kappaleissa.



Kuva 7. Toiminnallisuuden kaavio [14].

Sovelluksen käyttö alkaa sisäänkirjautumisnäytöltä, jossa käyttäjä syöttää henkilökorttinsa numeron manuaalisesti tai lukee sen viivakoodin lukijalla. Käyttäjän kirjautuessa sisään hänen oikeutensa tarkistetaan LIMS-järjestelmässä. Kun käyttäjä on tunnistettu onnistuneesti, sovelluksessa siirrytään suunniteltujen näytteenottotapahtumien hakukriteereiden rajausnäytölle.

Hakukriteerinäytöllä on haettavien suunniteltujen näytteenottotapahtumien rajaamisehtoja. Näytöllä voi olla erilaisia rajausehtoja riippuen sisäänkirjautuneesta käyttäjästä. Ehdot määritellään LIMS-järjestelmässä. Ne ovat esimerkiksi paikka, työryhmä, aloitushetki ja lopetushetki. Käyttäjä rajaa hakua ja lähettää hakuehdot LIMS-järjestelmään. Sovelluksessa siirrytään suunniteltujen näytteenottotapahtumien valintanäytölle, johon on listattu hakukriteereiden mukaiset tapahtumat.

Näytteenottotapahtumien valintanäyttö sisältää listan kaikista haetuista näytteenottotapahtumista. Ne on värikoodattu tilan mukaan. Tilat ovat ei aloitettu, kesken ja valmis. Käyttäjä voi valita halutun näytteenottotapahtuman listalta selaamalla sitä laitteen näppäimistön kautta tai vaihtoehtoisesti lukemalla näytteenottopaikan viivakoodin. Viiva-

koodin lukemisella tapahtumalistalle suodattuu vain kyseisen paikan näytteenottotapahtumat.

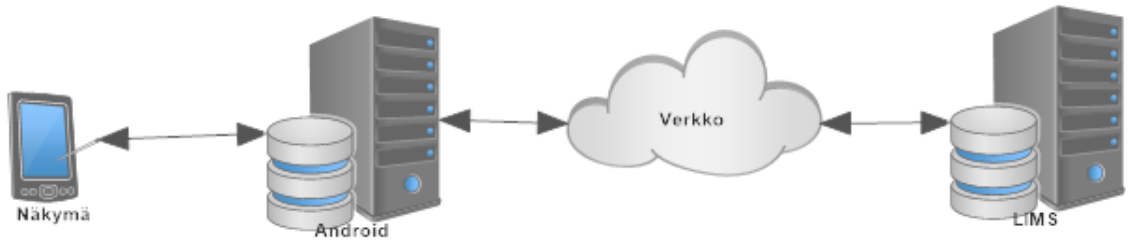
Kun käyttäjä valitsee listalta haluamansa näytteenottotapahtuman, ohjelmassa siirrytään näytteenottotapahtuman käsittelynäytölle. Tapahtuman käsittelynäytöllä on selkeästi näytteenottotapahtuman yksilöivät tiedot sekä näytteenottoaika. Näytöllä on näiden lisäksi kaksi välilehteä: ohjeet ja tulosten syöttö. Ohje-välilehdellä on näytteen ottamiseen liittyvä ohjeistus. Tulos välilehdellä on lueteltuna näytteeseen kuuluvat kirjattavat tulokset sekä kohta, johon käyttäjä voi kirjoittaa tarvittavat lisätiedot. Tulostensyöttö tulee tapahtua niin, että käyttäjän tarvitsee koskea mobiililaitteeseen mahdollisimman vähän. Näytteenottoajan päivittämiseen selvitetään ääniohjauksen soveltuvuutta.

Kun käyttäjä on kirjannut tarvittavat tulokset, voi hän lähettää näytteenottotapahtumat takaisin LIMSiin näytteenottotapahtumien listanäkymällä olevalla lähetä-painikkeella. Jos listalla on keskeneräisiä tapahtumia, niistä kysytään käyttäjältä, haluaako hän lähettää myös keskeneräiset tapahtumatiedot.

Käyttäjä voi lopettaa ohjelman käytön kaikissa ohjelman vaiheissa. Jos ohjelmassa on lähettämättömiä näytteenottotapahtumien tuloksia, varmistetaan käyttäjältä, haluaako hän varmasti kirjautua ulos, jolloin tapahtumille rekisteröidyt näytteenottoajat ja mahdolliset tulokset häviävät. Käyttäjällä on myös mahdollisuus palata takaisin edelliselle näytölle. Uloskirjautuessa mobiililaitteesta tyhjennetään käyttäjän ja tapahtumien tiedot.

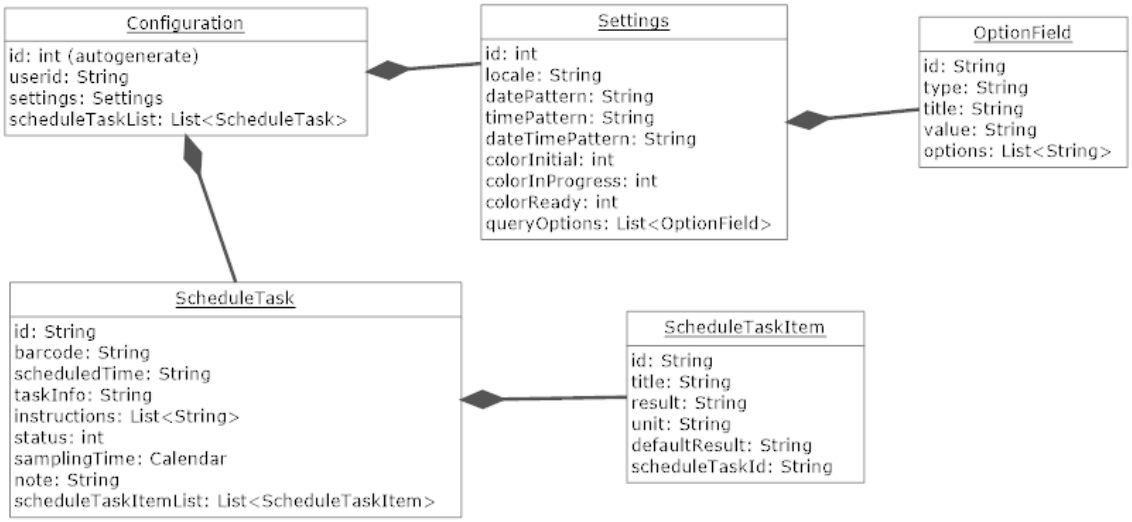
3.2 Arkkitehtuuri ja rajapinnat

Järjestelmän arkkitehtuuri on hyvin suoraviivainen, sillä sovelluksella on ainoastaan rajapinta LIMS-järjestelmän kanssa. Arkkitehtuuri on kuvattu alla olevassa kuvassa 8. Kannettavassa Android-laitteessa toimiva mobiilisovellus vastaanottaa ja lähettää tietoa LIMSiin Web Service rajapinnan kautta. Sovellus päivittää näkymän tiedot ja tallentaa näkymästä vastaanotetut tiedot SQLite-tietokantaansa. Sovelluksen on toimittava tietojen haun ja lähetyksen välillä kokonaan ilman verkkoyhteyttä.



Kuva 8. Järjestelmäkuvaus.

Tietokantana käytetään Androidin omaa laitteessa olevaa SQLite-tietokantaa. Tietokantaan tallennetaan konfiguraatio-olio, jonka rakenne on esitetty kuvassa 9. Konfiguraatio-olio pitää sisällään muun muassa käyttäjän tunnisteen, asetukset ja näytteenottotapahtumalistan. Tietoa tietokannasta saadaan konfiguraatio-olion kautta, johon tietoa tallennetaan.



Kuva 9. Konfiguraatio-olion rakenne.

Tietokantaan tallentamista helpottamaan käytetään sovelluksessa Googlen Room-kirjastoa. Room-kirjasto toimii abstraktina tasona SQLite-relaatiotietokannan ja sovelluksen välissä. Kirjasto mahdollistaa tietokannassa olevan tiedon käsittelyn yhden konfiguraatiotaulun välityksellä.

3.3 Teknologia

Mobiilisovellus suunnitellaan kuvassa 10 näkyvään Honeywell CN75 -kämmentietokoneeseen yhteensopivaksi. Laitteessa on Android 6.0 -käyttöjärjestelmä vastaten Android APIn tasoa 23, jota sovelluksen on vähintään tuettava. Laitteessa on viivakoodinlukija ja mahdollisuus verkkoyhteyteen joko itse laitteesta tai telakan avulla. Näyttö on pienehkö kosketusnäyttö, kooltaan 3,5 tuumaa. Sovellusta voidaan ohjata kosketusnäytöltä, mutta kaikkien toimintojen on toimittava laitteen kiinteän näppäimistön kautta. Näppäimistö on kuvan 10 kaltainen suppea näppäimistö. Laitteessa on lisäksi kummallakin sivulla kaksi painiketta, jotka eivät näy kuvassa.

Kuva 10. Honeywell CN75 -kämmentietokone.

Android-sovelluksen kehitysympäristönä käytetään Android Studio -ohjelmaa, jossa sovellus kirjoitetaan Java- ja XML-ohjelmointikielillä. Lisäksi tietokantakyselyjä varten tarvitaan SQL (Standard Query Language) -kieltä. SQL on standardoidu kyselykieli, jolla voidaan tehdä relaatiotietokantaan hakuja, muutoksia ja lisäyksiä. Mobiilisovelluksen ja LIMS-järjestelmän rajapinta toteutetaan käyttäen Web Service -tekniikkaa. Web Service mahdollistaa sovelluksen kommunikoinnin tietoverkon yli.

4 Selvitys ääniohjauksen toteuttamisesta

4.1 Ääniohjauksen perusvaatimukset

Toteutuksen alussa perehdyttiin ääniohjauksen toteuttamiseen osana mobiilisovellusta. Koska ääniohjauksen toteuttaminen olisi vaatinut liikaa aikaa, jätettiin se lopulta pois. Seuraavassa kuvataan ääniohjauksen mahdollisen toteuttamisen kannalta oleellisia vaatimuksia ja tuloksia.

Ääniohjausta haluttiin käyttää lähinnä apuna näytteenottoajan kirjaamisessa. Sen tulisi toimia suomeksi ja mahdollisimman täsmällisesti. Esimerkiksi käyttäjän sanoessa ”aika” kirjautuisi käsittelyssä olevalle näytteelle näytteenottoaika. Etuna olisi kirjatun näytteenottoajan täsmällisyys ja kirjaamisen helppous, sillä käyttäjän ei tarvitsisi koskea laitteeseen lainkaan. Virheelliset puheen tunnistukset aiheuttaisivat puolestaan vääriä näytteenottoaikoja näytteille. Käyttäjä ei näitä välttämättä huomaisi, sillä ääniohjauksen haluttaisiin toimivan automaattisesti. Projektissa oli myös tavoitteena napin painalluksen korvaaminen äänikomennolla, joten napin painamista vaativia sovelluksia ei voitu hyödyntää. Samalla vaivalla, kun käyttäjä painaisi nappia antaakseen äänikomennon, voitaisiin jo itse toiminto toteuttaa.

4.2 Valmiit sovellukset ja niiden rajoitukset

Selvittelyn alussa tutustuttiin saatavilla oleviin puheentunnistuksen sovelluksiin, joita on lukuisia. Näistä suurin osa tarvitsee verkkoyhteyden toimiakseen. Koska mobiilisovelluksen määrittelyn mukaan näytteenottotapahtuman tietojen kirjaamishetkellä ei ole verkkoyhteyttä käytettävissä, ei esimerkiksi Androidin omaa äänentunnistusluokkaa SpeechRecognizer voitu hyödyntää [15]. Ilman verkkoyhteyttä olevia puheentunnistus sovelluksia on muutamia, mutta yksikään näistä ei sovellu tämän projektin hyödynnettäväksi asiakkaan asettamien vaatimuksien takia. Ongelmaksi muodostuu, että joko sovelluksessa ei ole suomen kieltä valittavana, aktivoituakseen täytyisi painaa jotain painiketta tai pitäisi sanoa jokin tietty herätyssana kuten Googlen sovelluksessa “Ok Google”.

Esimerkki puheentunnistussovelluksesta, joka toimii ilman verkkoyhteyttä, on Androidille saatava sovellus nimeltään Saiy. Saiyn käytössä aktivointinapin lisäksi ongelmaksi

koituu se, että sovellus opetetaan tunnistamaan ja analysoimaan yhden käyttäjän puhetta. Tämä ei sovellu tilanteeseen, jossa samaa laitetta käyttää useampi eri henkilö.

Yksi avoimen lähdekoodin puheentunnistusohjelma, jonka voisi muokata sopivaksi projektin käyttötarkoitukseen, on CMUSphinxin sovellus PocketSphinx [16]. Suomen kieltä ei tässäkään ole valmiina, mutta se on siihen mahdollista lisätä.

4.3 PocketSphinxin sovelluksen muokkaaminen projektille sopivaksi

PocketSphinxin valmiiseen ääniohjauksen toiminnan runkoon voidaan itse määrittää sanakirja, kielimalli ja akustinen malli. Tämä mahdollistaa ääniohjauksen muokkaamisen soveltuvaksi tämän projektin tarpeisiin.

Tarvittavista malleista sanakirja on yksinkertaisin toteuttaa itse. Sanakirjaan tulee lueteltuna tunnistettavat sanat ja niiden ääntämisen mallit. Sanat voidaan joko itse kirjoittaa tai käyttää valmista suomen kielen sanakirjaa. Näitä on verkon kautta yleisesti saatavilla. Sanakirja voidaan kirjoittaa myös itse hyödyntäen valmiita sovelluksia kuten Phonetisaurus [17] ja Sequitur [18]. Kaikki ääniohjauksen tunnistettavat sanat on oltava lueteltuina sanakirjassa, koska muuten ohjelma ei niitä tunnista. [19.]

Sanakirjan lisäksi sanojen tunnistamiseen tarvitaan kielimalli. Kielimalli sisältää tunnistettavia sanoja ja niiden esiintymisen todennäköisyyksiä. Kielimallin tekemiseen tarvitaan tekstiaineistoja, joista siivotaan muun muassa lyhenteet pois ja muutetaan numerot sanoiksi. Tekstiaineistosta voidaan työkalujen avulla laskea kielessä esiintyviä sanoja sekä niiden esiintymistodennäköisyyksiä. [20.]

Sanakirjan ja kielimallin lisäksi puheen analysointiin tarvitaan vielä akustinen malli, joka on näistä työläin tehdä itse. Akustisen malliin tarvitaan lista lauseista, sanakirja lauseissa esiintyvistä sanoista ja äänite, jossa lauseet puhutaan. Näistä saadaan työkalujen avulla muodostettua akustinen malli. Akustinen malli ei kuitenkaan ole vielä tällaisenaan valmis, vaan sitä täytyy opettaa. Akustisen mallin opettaminen on noin kuukauden mittainen prosessi, johon tarvitaan ainakin 50 tuntia nauhoitettua puhetta eri puhujilta. Tässä vaiheessa selvitystä tultiin siihen tulokseen, ettei puheentunnistuksen akustista mallia ruveta toteuttamaan. [20.]

4.4 PocketSphinxin toiminnallisuus

Selvittelytyön aikana kokeiltiin PocketSphinxin valmista ääniohjauksen toiminnallisuutta. Sitä testattiin valmiina olevalla englannin kielen sanakirjalla, kielimallilla ja akustisella mallilla. PocketSphinxin toiminnallisuus perustuu kahteen eri tyyliseen sanojen tunnistamiseen. Ensimmäinen on aktivointisana. Aktivointisanalla tarkoitetaan sanaa, jonka ohjelma poimii kaiken muun puheen joukosta samalla, kun tunnistus on aktiivisena koko ajan ohjelman taustalla analysoiden puhetta. PocketSphinxin oletusaktivointisana on ”Oh mighty computer”, mutta asiakkaan sovelluksessa se olisi ollut suomenkielinen sana kuten ”aika”. Kyseistä aktivointisanaa ei kuitenkaan pysty muokkaamaan jo olemassa olevilla työkaluilla, vaan sen vaihto vaatisi merkittävää lisätyötä. Lisäksi sanan tulee olla vähintään 3–4 tavua pitkä, eikä se saa olla puheessa yleisesti ilmenevä sana.

Toinen PocketSphinxin tukema puheentunnistamisen tyyli on tunnistuksen aktivointi esimerkiksi napin painalluksella. Aktivoinnin jälkeen ohjelma kuuntelee rajatulle listalle määriteltyjä sanoja. Esimerkki vastaavasta toiminnallisuudesta on puhelinsovellus, jossa ohjelma kuuntelee aktivoinnin jälkeen käyttäjän käskyä kuten “soita matti meikäläinen”. Puheentunnistus analysoi puheen antaen tuloksena yhteystiedoissa olevan nimen, joka on lähimpänä kuunneltua nimeä. Tämän jälkeen sovellus soittaa ko. henkilölle. PocketSphinxin huomattiin toimivan vastaavassa tilanteessa siten, että ohjelma hakee aina parhaan vaihtoehdon listalta vaikkei nimeä listalla olisikaan. Tämä kasvattaisi helposti väärien tunnistuksien määrää. Johtuen sekä edellä mainitusta syystä että tunnistuksen aktivoimisesta napin painalluksella ei tämä puheentunnistamisen tyyli soveltuisi käytettäväksi tässä projektissa.

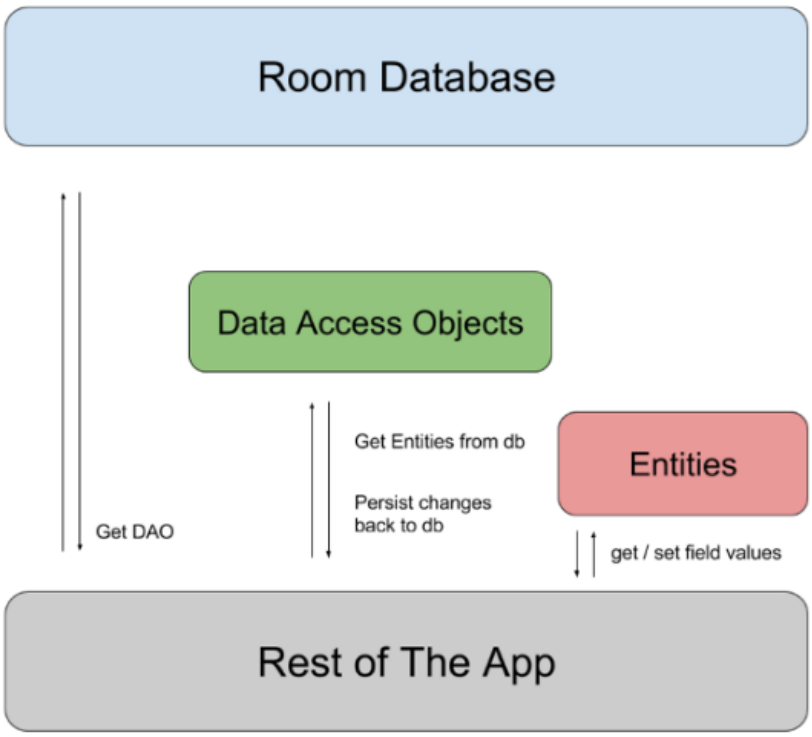
Koska tällä hetkellä ei ollut saatavilla valmista, pienellä muokkauksella asiakkaan tarpeisiin soveltuvaa ääniohjausta, päätettiin se jättää kokonaan pois. Ääniohjauksen toteuttaminen veisi huomattavasti lisää aikaa, kasvattaisi sovelluksen hintaa merkittävästi eikä olisi tämänkään jälkeen täydellinen. Puheentunnistus perustuu aina todennäköisyyksiin, joten väärin tunnistettujen sanojen mahdollisuus on huomioitava toteutuksessa. Tässä projektissa olikin järkevämpää käyttää nappia tallentamaan näytteenoton aika.

5 Toteutus

5.1 Tietokanta

Mobiilisovelluksen käytönaikainen tieto tallennetaan laitteessa olevaan SQLite-tietokantaan viiteen tauluun. Tietokantahakujen toteuttamiseksi mietittiin kahta eri vaihtoehtoa: suoria SQL-hakulauseita ja Room-kirjastoa. Näistä päädyttiin Room-kirjastoon, koska se yksinkertaistaa olioiden tallentamisen tietokantaan nopeuttaen projektin toteuttamista sekä on yhteensopiva Android-sovellusten kanssa.

Kuvassa 11 on kuvattu Room-tietokannan arkkitehtuurin rakenne. Tietokantaan tulevat taulut muodostetaan Entity-luokista. DAO (Data Access Objects) on luokka, jossa määritellään tietokannan hakulauseet. Kaikki tietokantakyselyt suoritetaan kutakin Entity-luokkaa vastaavan DAO-luokan kautta. Tietokantaan yhteydessä olevat DAO-luokat ja Entity-luokat määritellään yhdessä luokassa, joka sisältää annotaation Database.



Kuva 11. Room-tietokanta-arkkitehtuuri [21].

Room-kirjaston kautta tietokantaan tallennettu olio muutetaan merkkijonoksi, ja tietokannasta haettu olio palautuu oliona. Tietokantaan tallennettavan olion muuttujien täytyy olla primitiivimuotoa, sillä muuten väliin tarvitaan muunnin. Room-kirjaston muunnin merkitään TypeConverter-annotaatiolla. Esimerkiksi merkkijono listan muuntaminen merkkijonoksi ja takaisin tehdään, kuten esimerkkikoodissa 5 näkyy.

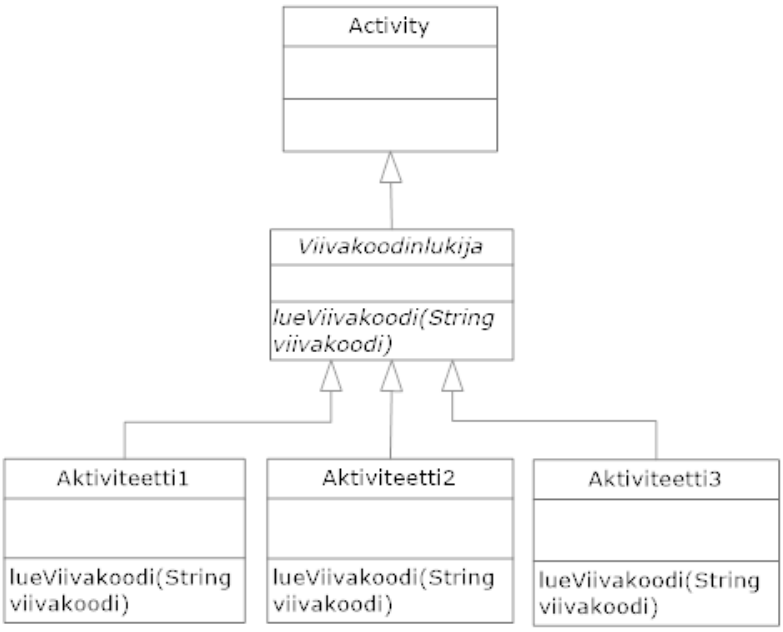
```
@TypeConverter
public String fromList(List<String> list){
    if(list == null){
        return null;
    }
    Gson gson = new Gson();
    return gson.toJson(list);
}

@TypeConverter
public List<String> toList(String data){
    if(data == null){
        return null;
    }
    Gson gson = new Gson();
    Type type = new TypeToken<List<String>>(){}
    .getType;
    return gson.fromJson(data, type);
}
```

Esimerkkikoodi 5. TypeConverter-muunnin muuttaa merkkijonolistan merkkijonoksi, ja takaisin listaksi.

Ensimmäisessä muuntimessa muutetaan merkkijonolista merkkijonoksi. Lista muutetaan JSON (JavaScript Object Notation) -muodossa olevaksi merkkijonoksi Gson-kirjaston avulla. Gson on Java-kirjasto, jota käytetään olioiden sarjallistamiseen ja muuttamiseen takaisin olioiksi. Jälkimmäisessä muuntimessa muutetaan merkkijono takaisin koodissa määritetyksi olioksi. Koska sovellus sisältää olioita, joiden muuttujina on olioita, täytyi jokainen olio muuttaa erillisen muuntimen kautta merkkijonoksi.

5.2 Viivakoodin luku

Mobiilisovelluksessa viivakoodinlukua tarvitaan käyttäjän ja näytteenottopaikan tunnistamiseen. Honeywell CN75 -kämmentietokoneessa on vakiona viivakoodinlukija. Sen käsittelyä varten Honeywellilta on saatavilla oma API. Koska viivakoodinlukemista tarvitaan sovelluksen lähes kaikissa aktiviteeteissa, päätettiin siitä tehdä abstrakti luokka. Luokkarakenne on esitetty kuvassa 12.



Kuva 12. Luokkarakenne esittää abstraktin viivakoodilukija luokan sijainnin luokkarakenteessa.

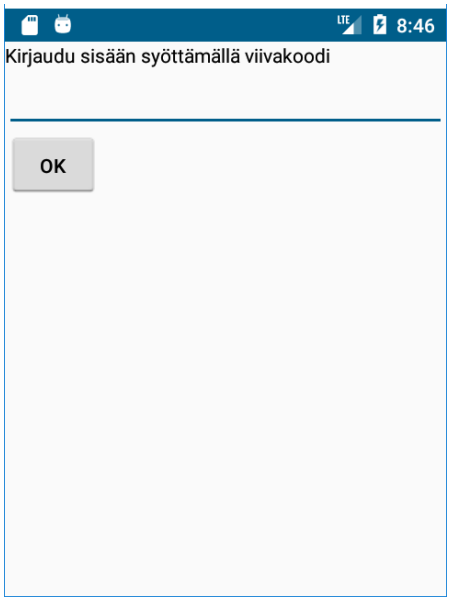
Sovelluksen muut aktiviteetit perivät viivakoodinlukija-aktiviteetin, jonka kautta ne perivät myös luokan Activity. Abstraktissa luokassa käsitellään yhteys viivakoodinlukijaan, sen avaaminen ja sulkeminen sekä luetun tiedon vastaanottaminen. Luokassa on yksi abstrakti metodi, jonka toteutus määritellään sen perivissä luokissa. Näin saadaan jokaiselle aktiviteetti luokalle oma käsittely viivakoodinlukijalta saatua tietoa varten.

5.3 Prototyyppi

Tässä osiossa esitellään sovelluksen tämänhetkinen prototyyppi. Sovelluksen toiminnallisuus on esitelty kappaleessa 3.1, ja tässä osiossa keskitytäänkin esittämään pääsääntöisesti käyttöliittymän ratkaisuja. Mobiilisovelluksen toiminnallisuus ja käyttöliittymä ovat valmiita, ja seuraavaksi työn alla on Web Service rajapinnan toteuttaminen.

Itse toiminnallisuuteen ja käyttöliittymään tulee vielä pieniä asiakkaan toiveiden mukaisia muokkauksia sitä mukaan, kun asiakas pääsee sovellusta kokeilemaan.

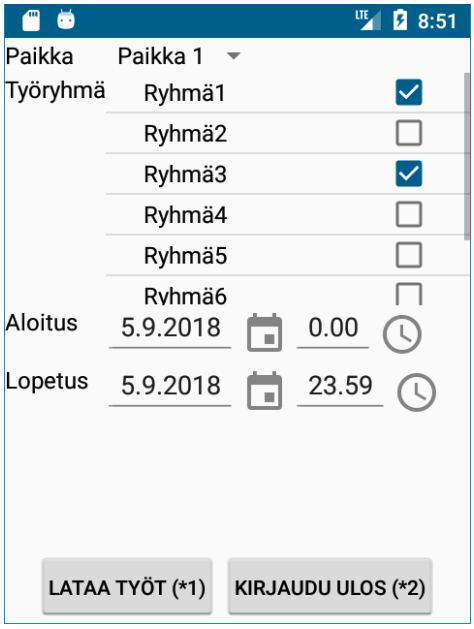
Android-sovelluksia käytetään pääsääntöisesti kosketusnäytön kautta toisin kuin tämän projektin mobiilisovellusta, jonka on oltava kokonaisuudessaan käytettävissä näppäimistön kautta. Sovelluksesta poistettiin kosketusnäytön ominaisuuksia, kuten kosketusnäytölle ilmaantuva näppäimistö, asettamalla tekstinsyöttökentän setShowSoftInputOnFocus arvoksi false. Tekstinsyöttökentistä poistettiin myös oikeinkirjoituksen tarkistus. Esimerkiksi kuvassa 13 näkyvässä sisäänkirjautumissivun tekstinsyöttökentässä käyttäjä voi joko lukea viivakoodin tai syöttää oman tunnisteensa kiinteän näppäimistön avulla.



Kuva 13. Sovelluksen sisäänkirjautumisen näkymä.

Sovellusta käytetään halkaisijaltaan 3,5 tuuman näytön omaavalla laitteella. Toteutuksessa haluttiin maksimoida näytön käyttöalue, joten sovellukselle määriteltiin toimintopalkiton (no action bar) teema. Toimintopalkki olisi näytön yläreunassa oleva osio, jossa olisi esillä muun muassa sovelluksen nimi. Toimintopalkkia voisi hyödyntää esimerkiksi haku toiminnallisuuksiin, mutta tässä sovelluksessa sille ei ole tarvetta. Näytön yläreunaan jätettiinkin ainoastaan palkki, jossa näkyy muun muassa laitteen akun varaus, verkkoyhteyden tila ja kellonaika.

Ohjelmiston suunnittelussa pyrittiin ottamaan huomioon ohjelmiston uudelleenkäytettävyys ja jatkokehityksen helppous. Ohjelmisto toteutettiinkin siten, että ohjelmisto on helposti muokattavissa LIMSin kautta. Näkymien luonti tehtiin mahdollisimman pitkälle ajonaikaisesti, ja osa värimäärityksistä tulee LIMSin kautta. Kuvassa 14 näkyvät suunniteltujen näytteenottotapahtumien hakukriteereiden määrittelyt on kaikki luotu ajonaikaisesti.



Kuva 14. Suunniteltujen näytteenottotapahtumien hakukriteereiden määritysnäkymä.

Hakukriteereiden esitystapa voidaan määritellä asiakkaan LIMS-ohjelmassa. Vaihtoehtoina ovat alasvetovalikko, selattava lista ja ajankohdan määritys, kuten kuvasta 15 näkyy. Prototyyppiin valittiin näytteenottopaikan valintaa varten Androidin Spinner-oliolla tehty alasvetovalikko. Työryhmälle valittiin Androidin ListView-oliolla luotu selattava lista. Ajankohdan määrittämiseen on puolestaan käytetty tekstinsyöttökenttiä, joihin käyttäjä voi itse kirjoittaa halutun päivämäärän ja kellonajan. Näiden lisäksi ajankohta voidaan valita kalenteri ja kello kuvista aukeavilla dialogeilla. Kalenteri ja kello -dialogin huono ominaisuus on se, että ne toimivat vain kosketuksen kautta eivätkä ole käytettävissä kiinteän näppäimistön kautta. Dialogit päädyttiin kuitenkin lisäämään sovellukseen, sillä käyttäjällä on tietyissä tilanteissa mahdollisuus käyttää sovellusta kosketusnäytöltä ilman suojavarusteita.

Kuva 15. Alasvetovalikko (vasmemmalla ylhäällä), selattava lista (vasemmalla alhaalla) ja kello dialogi (oikealla).

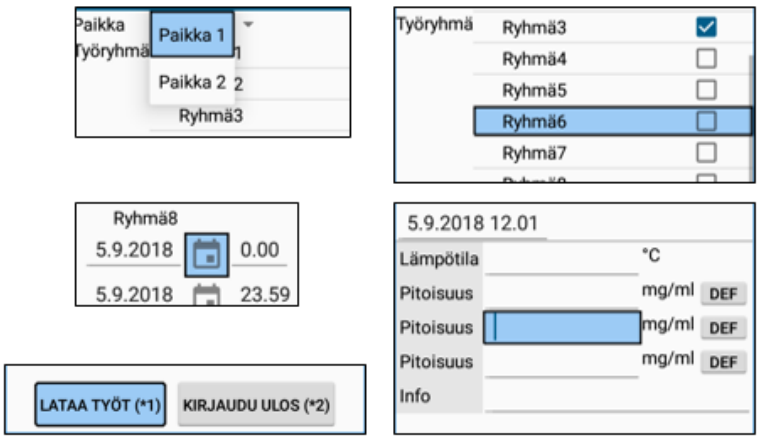
Alasvetovalikon ja selattavan listan rivit luotiin ArrayAdapter-olion avulla koodiesimerkin 6 mukaisesti. Riveille määriteltiin sekä tapahtumakäsittelijät että muita tyylipiirteitä. Alasvetovalikon muotoiluun käytettiin ArrayAdapterin yhtä valmista muotoilua nimeltä android.R.layout.simple_spinner_item. Muotoilu määrittelee sen, miltä alasvetovalikko näyttää silloin, kun valikko ei ole avattuna. Koska alasvetovalikon aktiiviselle riville haluttiin toteuttaa muokattu muotoilu, määriteltiin se ArrayAdapterin setDropDownViewResource-metodilla.

```
List<String> values; //määritelty aiemmin
Spinner spinner = new Spinner(this);
ArrayAdapter adapter = new ArrayAdapter<String>(
this, android.R.layout.simple_spinner_item, values);
adapter.setDropDownViewResource(R.layout.
spinner_dropdown_item_custom);
spinner.setAdapter(adapter);
```

Esimerkkikoodi 6. Alasvetovalikon (Spinner) luominen.

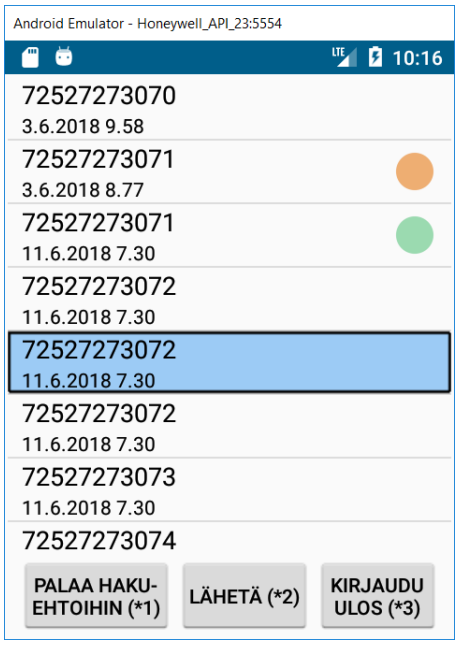
Asiakkaan vaatimus oli saada koko sovellukselle yhtenäinen ja selkeä aktiivisen kentän osoittava muotoilu. Aktiivisen kentän korostamiseen haluttiin sekä selkeä väri että musta reunus, kuten kuvassa 16 on esitetty. Tätä toteutusta hankaloittaa se, ettei aktiivisen kentän muotoilua voida määrittää yleisesti yhdessä paikassa ohjelmiston koodia Android-sovelluksissa.

Projektin koodin tyylimäärityksissä voidaan määrittää *colorControlHighlight*-väri, mutta tämä antaa vain haalean korostuksen kentille. Siksi asiakkaan vaatimusten täyttämiseksi täytyi painikkeille (Button) määrittää aktiivinen tausta kuvana. Kuvat piti puolestaan määrittää 9-patch image -tiedostomuotoisena, mikä mahdollistaa taustakuvan koon muokkaamisen kohteen koon mukaisesti. Tekstinsyöttökenttien, listan rivien ja kuvien aktiivinen tausta määritettiin XML-tiedoston shape-tunnisteessa.



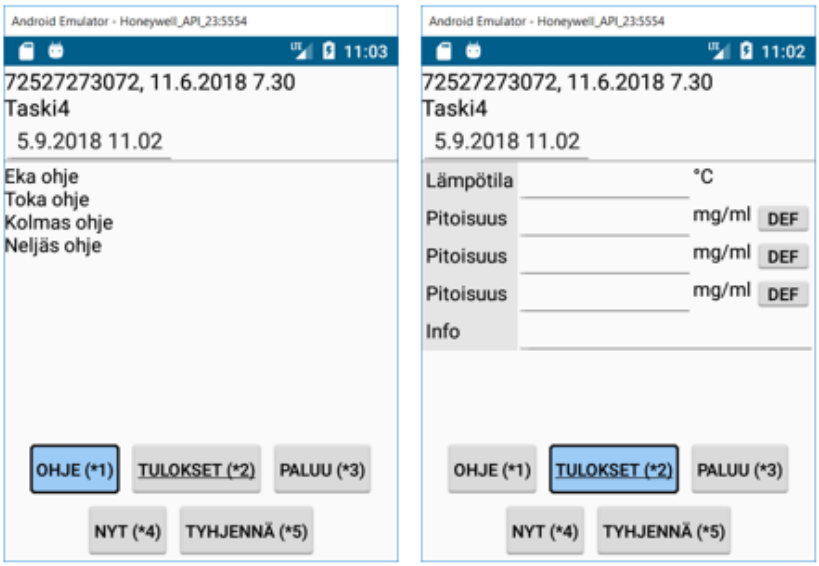
Kuva 16. Aktiivinen kenttä eri näytön komponenteissa.

Näytölle listataan suunnitellut näytteenottotapahtumat käyttäjän määrittelemien hakukriteereiden mukaisesti. Näkymässä haluttiin eritellä eri tilassa olevat näytteenottotapahtumat, joten keskeneräisten rivien loppuun lisättiin oranssi pallo ja valmiiden rivien loppuun vihreä pallo kuvan 17 mukaisesti. Näytteenottotapahtuma voidaan valita listalta manuaalisesti tai lukemalla viivakoodi, jolloin näytteenottotapahtuman valinta tapahtuu automaattisesti. Jottei käyttäjän tarvitse selata koko listaa loppuun asti, toteutettiin alhaalla oleville toimintopainikkeille pikanäppäimet. Laitteen suppean fyysisen näppäimistön takia pikanäppäimet toteutettiin siten, että painikkeiden rivi aktivoituu tähti (*) -näppäimellä.



Kuva 17. Näkymä suunniteltujen näytteenottotapahtumien listauksesta.

Näytteenottotapahtuman käsittely toteutettiin yhden aktiviteetin alle. Näkymän sisältö vaihtuu sen mukaisesti, onko käyttäjä valinnut alavalikosta kohdan *ohje* vai *tulokset*. Kummatkin näkymät ovat esitelty kuvassa 18. Näkymän sisällön vaihtuminen on toteutettu aktiviteettiin liitetyn fragmentin avulla. Fragmentit toteutetaan omana luokkana, jotka periytetään Fragment-luokasta.



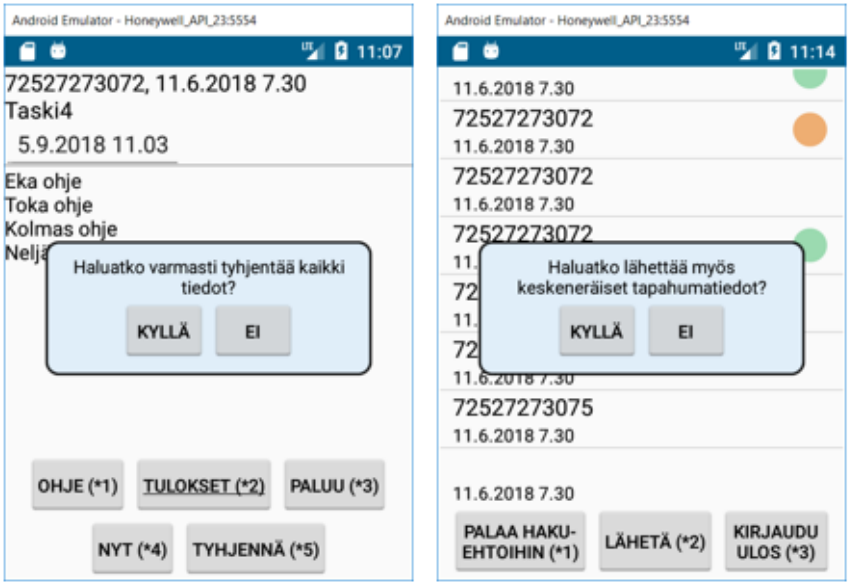
Kuva 18. Näytteenottotapahtuman käsittelynäkymä, jossa sisältö vaihtuu aktiivisen fragmentin mukaisesti.

Fragmentin näyttämiseen aktiviteetissa tarvitaan koodiesimerkissä 7 olevat vaiheet. Aktiviteetin ja fragmentin välillä tapahtuvat tiedonsiirto hoidettiin Bundle-olion avulla. Sovelluksen oliota ei pysty sellaisenaan siirtämään, vaan se täytyy ensin sarjallistaa tarkoittaen olion muuttamista tavuiksi siirtoa varten. Bundle-olion putSerializable-metodi toteuttaa olion sarjallistamisen. Metodille annetaan parametrina sekä merkkijonoavain että siirrettävä olio. Fragmetti luokkaan siirretty sarjallistettu olio voidaan muuttaa avaimen avulla takaisin olioksi.

```
Fragment fragment = new OmaFragment();
Bundle bundle = new Bundle();
bundle.putSerializable("SCHEDULE_TASK", scheduleTask);
fragment.setArguments(bundle);
FragmentManager fm = getSupportFragmentManager();
FragmentTransaction ft = fm.beginTransaction();
ft.replace(R.id.showFragment, fragment);
ft.setTransition(FragmentTransaction.
TRANSIT_FRAGMENT_OPEN);
ft.commit();
```

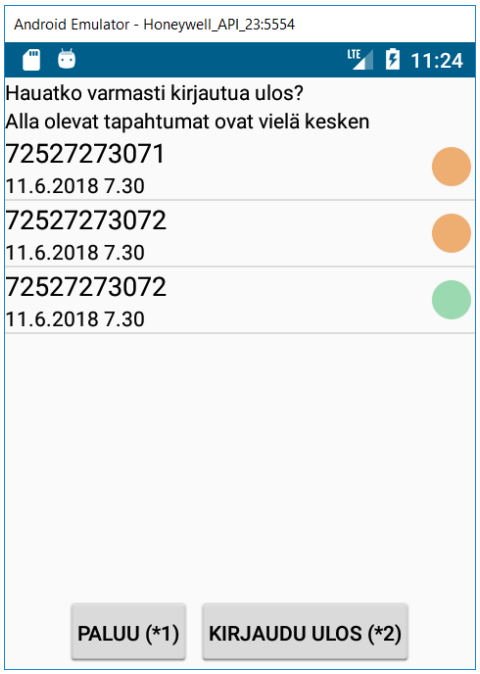
Esimerkkikoodi 7. Fragmentin luominen ja asettaminen näkyviin.

Sovellukseen lisättiin kuvassa 19 näkyvät dialogi-ikkunat, joihin käyttäjä joutuu vastaamaan jatkaakseen toimintaansa. Dialogeihin käytettiin Andoidin PopupWindow-luokkaa. Kaikki dialogit pystyttiin tekemään samalla PopupWindow'n layout-tiedostolla, johon liitettiin tapauskohtaiset tekstit ja tapahtumakäsittelyt. Ensimmäinen dialogi "Haluatko varmasti tyhjentää tiedot?" lisättiin varmistamaan se, että käyttäjä todella haluaa tyhjentää näytteenottotapahtuman tiedot. Tyhjentäminen tapahtuu yhtä painiketta painamalla, joten dialogilla halutaan estää tahattomat virheet. Toinen dialogi kysyy käyttäjältä, haluaako hän lähettää myös keskeneräiset tapahtumatiedot.



Kuva 19. Dialogit.

Sovelluksesta uloskirjautuminen varmistetaan käyttäjältä erillisen näkymän kautta. Varmistus on tarpeen, sillä uloskirjautumisen yhteydessä laitteen tietokantaan tallennetut tiedot tyhjennetään, eikä tapahtumaa voida perua. Kun käyttäjä on lähettänyt kaikki käsiteltyjen näytteenottotapahtumien tiedot LIMS-järjestelmään, varmistetaan käyttäjältä pelkästään uloskirjautuminen. Jos käyttäjällä on laitteella käsiteltyjä näytteenottotapahtumia, joita ei ole lähetetty LIMS-järjestelmään, tiedotetaan käyttäjää uloskirjautuessa kyseisistä tapahtumista. Kuvassa 20 esitetään uloskirjautumisen varmistus tilanteessa, jossa käyttäjällä on keskeneräisiä tapahtumia. Uloskirjautuminen voidaan kuitenkin halutessa suorittaa mutta tällöin keskeneräiset tapahtumatiedot tyhjennetään. Vaihtoehtoisesti käyttäjä voi palata sovellukseen viimeistelemään tai lähettämään näytteenottotapahtumat.



Kuva 20. Uloskirjautumisen varmistus.

Päivämäärä ja kellonaika kentille tehdään ajonaikaista validointia. Käyttäjällä ei ole mahdollista valita määrittelemätöntä päivää tai kellonaikaa kalenteri ja kello dialogeissa. Koska sovelluksessa on mahdollistettu ajankohdan syöttäminen manuaalisesti, täytyy käyttäjän syöttämä tieto tarkistaa. Tilanteissa, joissa käyttäjä on antanut virheellisen päivän tai kellonajan, käyttöliittymä huomauttaa virheestä heti, eikä mahdollista sovelluksessa etenemistä ennen kuin virhe on korjattu.

6 Yhteenveto

Insinöörityön tavoitteena oli kehittää mobiilisovellus osaksi asiakkaan LIMS-järjestelmää ja tuottaa yritykselle tietoa sovelluksen kehittämisestä. Mobiilisovellus suunniteltiin asiakkaan vaatimusten mukaisesti heidän valitsemaan laitteeseen sekä ympäristöön, jossa on rajoituksia verkkoyhteyden saatavuuden kanssa. Työssä toteutettiin onnistuneesti vaatimusten mukainen mobiilisovellus, paneuduttiin LIMSin, Android ohjelmoinnin ja ääniohjauksen teoriaan sekä kasvatettiin Software Point Oy:n tietotaitoa mobiilisovelluksien ohjelmoinnista.

Ääniohjauksen käyttämisen mahdollisuutta osana sovellusta tutkittiin. Valmiita, tämän sovelluksen vaatimukset täyttäviä olevia sovelluksia ei kuitenkaan löytynyt. Erityisesti sovelluksen käyttö suomen kielellä, ja ilman verkkoyhteyttä rajasi lähes kaikki käytettävissä olevat sovellukset pois. Ääniohjauksen toteuttaminen olisi vaatinut sanakirjan, kielimallin ja akustisen mallin tekoa itse, joista akustinen malli olisi ollut erittäin työläs. Muokatun ääniohjauksen toteuttamisen todettiin kuitenkin olevan mahdollista, mutta käytössä olevan ajan puitteissa se päätettiin jättää tästä projektista pois. Tulevaisuudessa, jos siihen on tarvetta, voi mobiilisovellusta jatkokehittää tukemaan ääniohjausta.

Mobiilisovelluksen kehitystyön aikana yritykselle kertyi kokemusta Android-ohjelmoinnista. Android-ohjelmoinnissa on oma, aktiviteettien elinkaareen perustuva toimintalogiikka, joka hallitsee sovelluksen kulkua. Projektin aikana osaamista kertyi myös sovelluksen näkymien eri komponenttien käytöstä rajoituksineen. Mobiilisovellus kehitettiin siten, että sen uudelleenkäytettävyys ei ole työlästä, ja sitä on mahdollista muokata LIMS-järjestelmän kautta.

Mobiilisovelluksen koekäyttöön saattaminen vaatii enää Web Service -rajapinnan toteuttamisen, mutta muilta osin sovellus on täysin valmis ja toimiva. Sovellukseen tullaan vielä toteuttamaan asiakkaan koekäytön perusteella havaitsemia muutostarpeita.

Sovellusta voidaan edelleen jatkokehittää käsittämään ääniohjaus, mikäli asiakas sitoutuu sen vaatimaan työmäärään eli kustannuksiin. Toisena jatkokehitysajatuksena on lokalisaatio. Projektissa kehitetty mobiilisovellus on suunniteltu toimimaan suomenkielellä, mutta sovellukseen voisi hyvinkin liittää mahdollisuuden valita käytettävä kieli ja muotoilut.

Lähteet

1 Nakagawa, Allen S. 1994. LIMS: Implementation and Management. Pennsylvania, USA, The Royal Society of Chemistry. Sivut 2–4.

2 Harju, Jukka. 2013. Android Ohjelmoinnin Perusteet. Helsinki: Books on Demand GmbH.

3 Platform Architecture. Verkkoainesto. Google Developers. <https://developer.android.com/guide/platform/>. Päivitetty 3.9.2018. Luettu 23.9.2018.

4 Android API. Verkkoaineisto. Google Developers. <https://developer.android.com/reference/classes>. Päivitetty 6.6.2018. Luettu 3.8.2018.

5 Android Studio. Verkkoaineisto. Google Developers. <https://developer.android.com/studio/>. Luettu 3.8.2018.

6 Android API Activity. Verkkoaineisto. Google Developers. <https://developer.android.com/reference/android/app/Activity>. Päivitetty 6.6.2018. Luettu 3.8.2018.

7 Understand Tasks and Back Stack. Verkkoaineisto. Google Developers. <https://developer.android.com/guide/components/activities/tasks-and-back-stack>. Päivitetty 28.08.2018. Luettu 3.9.2018.

8 Android API Log. Verkkoaineisto. Google Developers. <https://developer.android.com/reference/android/util/Log>. Päivitetty 6.6.2018. Luettu 5.9.2018.

9 Basic Concepts of Speech Recognition. 2018. Verkkoaineisto. CMUSphinx. <https://cmusphinx.github.io/wiki/tutorialconcepts/>. Luettu 2.4.2018.

10 Schroeder Manfred R. 1999. The Speech Signal. Teoksessa Computer Speech. Springer Series in Information Sciences, vol 35. Springer, Berlin, Heidelberg. Sivut 105–106.

11 Zhao Yunxin, Xue Jian, Chen Xin. 2015. Ensemble Learning Approaches in Speech Recognition. Teoksessa Ogunfunmi T., Togneri R., Narasimha M. (ed.). Speech and Audio Processing for Coding, Enhancement and Recognition. Springer, New York, NY. Sivut 122–128.

12 Building a phonetic dictionary. Verkkoaineisto. CMUSpinx. <https://cmusphinx.github.io/wiki/tutorialdict/>. Luettu 3.4.2018.

13 Gregor Donaj and Zdravko Kačič. 2017. Speech Recognition in Inflective Languages. Teoksessa Language Modeling for Automatic Speech Recognition of Inflective Languages. SpringerBriefs in Electrical and Computer Engineering. Springer, Cham. Sivut 1–18.

14 Kauhanen Antti, Team Leader, Software Point Oy, Espoo, 30.1.2018.

15 Android API SpeechRecognizer. Verkkoaineisto. Google Developers. <https://developer.android.com/reference/android/speech/SpeechRecognizer.html>. Päivitetty 6.6.2018. Luettu 02.04.2018.

16 PocketSphinx on Android. Verkkoaineisto. CMUSphinx. <https://cmusphinx.github.io/wiki/tutorialandroid/>. Luettu 3.4.2018.

17 Phonetisaurus. Verkkoaineisto. <https://code.google.com/archive/p/phonetisaurus/>. Luettu 5.4.2018.

18 Sequitur G2P. Verkkoaineisto. <http://www-i6.informatik.rwth-aachen.de/web/Software/g2p.html>. Luettu 5.4.2018.

19 Building a phonetic dictionary. Verkkoaineisto. CMUSphinx. <https://cmusphinx.github.io/wiki/tutorialdict/>. Luettu 3.4.2018.

20 Building a language model. Verkkoaineisto. CMUSpinx. <https://cmusphinx.github.io/wiki/tutoriallm/>. Luettu 3.4.2018.

21 Save data in a local database using Room. Verkkoaineisto. Google Developers. <https://developer.android.com/training/data-storage/room/>. Luettu 3.9.2018.