VAASAN YLIOPISTO

TEKNIIKAN JA INNOVAATIOJOHTAMISEN YKSIKKÖ

OHJELMISTOTEKNIIKKA

Aleksi Lahikainen

MOBIILISOVELLUS HIOMATUOTTEIDEN TESTAUKSESTA SAATAVAN TIEDON KERUUSEEN

Diplomityö, joka on jätetty tarkastettavaksi diplomi-insinöörin tutkintoa varten Vaasassa 10.4.2018

Työn valvoja	Professori Jouni Lampinen
Työn ohjaajat	DI Mika Filander Ins. Atte Leppälä

SISÄLLYSLUETTELO

LYHENNELUETTELO

AOT	Ahead-of-time, ennenaikainen kääntäminen
ART	Android Runtime Environment, Androidin suoritusympäristö
CLR	Common Language Runtime, .NET-sovelluskehyksen suoritusympäristö
HAVS	Hand-arm Vibration Syndrome, tärinätauti
HTTP	Hypertext Transport Protocol, tiedonsiirtoprotokolla
ISO	International Organization for Standardization, kansainvälinen standardoimisjärjestö
JIT	Just-in-time, ajonaikainen kääntäminen
JSON	JavaScript Object Notation, standardoitu tiedon esitysmuoto
MSSQL	Microsoft SQL Server, tietokantaohjelmisto
MVC	Model-View-Controller, ohjelmistoarkkitehtuuri
MVVM	Model-View-ViewModel, ohjelmistoarkkitehtuuri
OMG	Object Management Group, oliotekniikoita kehittävä konsortio
REST	Representational State Transfer, HTTP-protokollaan perustuva arkkitehtuurimalli
SDK	Software Development Kit, ohjelmiston kehitystyökalukokoelma
SQL	Structured Query Language, standardoitu kyselykieli
UML	Unified Modeling Language, notaatiostandardi
WPF	Windows Presentation Foundation, vektoripohjainen käyttöliittymäkirjasto

VAASAN YLIOPISTO
Tekniikan ja innovaatiojohtamisen yksikkö
Tekijä: Aleksi Lahikainen
Diplomityön nimi: Mobiilisovellus hiomatuotteiden testauksesta saatavan tiedon keruuseen
Valvoja: Professori Jouni Lampinen
Ohjaajat: DI Mika Filander, Ins. Atte Leppälä
Tutkinto: Diplomi-insinööri
Koulutusohjelma: Tietotekniikan koulutusohjelma
Suunta: Ohjelmistotekniikka
Opintojen aloitusvuosi: 2012
Diplomityön valmistumisvuosi: 2018 **Sivumäärä:** 65

TIIVISTELMÄ

Hiomatuotteisiin kuuluvat hiomapaperit käsi- ja konehiontaan erilaisille pinnoille. Käytettävä hiomapaperi tulee valita hiottavan pinnan materiaalin ja muodon, sekä tavoitellun hiontatuloksen mukaan. Näistä seikoista johtuen hiomatuotteiden tuotekehityksessä tarvitaan tietoa käytännön sovelluksista, jotta tuotteiden kestävyyttä ja hiontaominaisuuksia voidaan kehittää erityyppisiä pintoja varten. Kerättävä tieto voi olla kokemuksiin tai mittauksiin perustuvaa. Jotta tietoa saataisiin mahdollisimman laajasti eri toimialoilta, on tuotteiden testauksessa järkevää tehdä yhteistyötä asiakkaiden kanssa. Tässä diplomityössä suunnitellaan ja toteutetaan mobiilisovellus, joka kerää tietoa hiomakoneelta Bluetoothin välityksellä ja ohjaa käyttäjää toimimaan oikein testausprosessin aikana. Testauksen jälkeen testaajat voivat antaa palautetta testatuista tuotteista. Sovellus integroidaan osaksi erästä hiomakoneiden aiheuttaman tärinän seurantaan tarkoitettua sovellusta.

Sovelluksen vaatimusmäärittely oli tehty osittain ennen tämän työn aloittamista, mutta vaatimukset tarkentuivat projektin edetessä lopulliseen muotoonsa. Työhön kuului asiakas- sekä palvelinsovellusten suunnittelu ja toteutus sekä järjestelmän ydintoimintojen kevyt testaus. Mobiilisovellus toteutettiin järjestelmäriippumattomien mobiilisovellusten toteutukseen tarkoitetussa *Xamarin*-kehitysympäristössä. Muita keskeisiä teknologioita projektissa olivat Bluetooth, Microsoft SQL Server sekä Node.js.

Työn tuloksena syntyi toimiva mobiilisovellus, jolla saadaan kerättyä numeerista dataa hiomakoneelta sekä käyttäjän kokemuksia ja mielipiteitä hiomatuotteista. Sovellus julkaistaan käyttäjien ladattavaksi sitten kun se on läpäissyt asiakastestauksen. Mobiilisovelluksen tueksi tullaan myöhemmin kehittämään erillinen järjestelmä kerätyn tiedon lukemista ja analysointia varten.

AVAINSANAT: mobiilisovellus, tiedonkeruu, hiomatuotteet, tuotetestaus

UNIVERSITY OF VAASA
The School of Technology and Innovations
Author: Aleksi Lahikainen
Topic of the Thesis: A mobile application for collecting data from testing of abrasive products
Supervisor: Professor Jouni Lampinen
Instructors: M. Sc. (Tech.) Mika Filander, B. Sc. (Tech.) Atte Leppälä
Degree: Master of Science in Technology
Degree Programme: Degree Programme in Information Technology
Major of Subject: Software Engineering
Year of Entering the University: 2012
Year of Completing the Thesis: 2018 **Pages:** 65

ABSTRACT

Abrasive products include sandpapers for hand or machine sanding for different surfaces. The sandpaper to be used must be selected according to the material and shape of the surface to be sanded, as well as the desired sanding result. Due to these factors, the product development of abrasive products requires information on practical applications in order to develop durability and abrasive properties of the products for different types of surfaces. The information to be collected can be based on experiences or measurements. In order to get information as widely as possible from different sectors, it is practical to cooperate with customers in product testing. In this thesis, a mobile application is designed and implemented, which collects data from a sander via Bluetooth and guides the user to operate correctly during the testing process. The application module will be integrated into an application originally developed for monitoring vibrations caused by sanders.

The requirements analysis of the application was made partly before starting this thesis, but the requirements were refined as the project progressed. The project involved designing and implementing client and server side applications as well as concise system testing of the core functionalities. The mobile application was implemented using a cross-platform development environment called *Xamarin*. Other key technologies in the project are Bluetooth, Microsoft SQL Server and Node.js.

As a result of this work, a functional mobile application was created to collect numerical data from sander as well as user experiences and opinions on abrasive products. The application will be released for public use after it passes the customer verification process. A separate system for reading and analyzing collected data will be developed later.

KEYWORDS: mobile application, data collection, abrasive products, product verification

1 JOHDANTO

Työntekijöiden turvallisuus ja työergonomia ovat asioita, joista työnantajan tulisi huolehtia. Hiomakoneita ja muita pyörivää liikettä hyödyntäviä koneita käytettäessä työntekijän ylävartaloon ja etenkin käsiin kohdistuu huomattava määrä tärinää. Pitkällä aikavälillä tärinä voi aiheuttaa erilaisia verenkiertoon, tuki- ja liikuntaelimistöön sekä hermostoon liittyviä sairauksia. Suomessa ammattitautiasetuksessa (1347/1988) on määritelty tärinästä johtuviksi ammattitaudeiksi valkosormisuus-oireyhtymä ja yläraajan monihermovaurio (Työterveyslaitos 2014a). Tärinästä johtuvista sairauksista puhutaan yleisesti tärinätautina (engl. *hand-arm vibration syndrome,* HAVS*).* Suomessa käsitärinälle altistuu arviolta 130 000 työntekijää vuodessa. (Työterveyslaitos 2014a, 2014b.)

Tärinästä johtuvien tautien ehkäisemiseksi työnantajan tulee seurata työkoneista työntekijöille aiheutuvaa tärinää. Tätä varten Euroopan Unioni on määrännyt tärinädirektiivin (44/2002), jossa työnantajaa velvoitetaan selvittämään työntekijään kohdistuvan tärinän määrä, arvioimaan siitä johtuvat riskit ja tarvittaessa tekemään toimenpiteitä tärinän vähentämiseksi. (Työterveyslaitos 2014c.) Tärinän määrä voidaan selvittää erilaisilla antureilla ja mittareilla tärinää aiheuttaviin työkoneisiin kiinnitettynä. Tärinää mitattaessa tulee selvittää päivittäinen tärinän vaikutusaika sekä käteen kohdistuvan tärinän kiihtyvyys ISO-standardien 5349-1 ja 5349-2 mukaisesti. (Työterveyslaitos 2014a.) Myös koneen valmistajan on annettava tietoa käsitärinästä, mikäli tärinän kiihtyvyys ylittää toimintaarvon 2,5m/s^2 (Suomen Työterveyslääkäriyhdistys 2016). Vaatimus tärinän valvonnasta tarjoaa myös mahdollisuuden hyödyntää tärinän mittaukseen käytettävistä antureista saatavaa tietoa tuotekehityksessä.

1.1 Tutkimuksen taustat

Tämä työ tehtiin Devatus Oy:lle osana Mirka Oy:n tilaamaa projektia. Devatus Oy on vaasalainen vuonna 2010 perustettu ohjelmistokehitystalo. Yrityksen ydinosaamiseen kuuluu mobiilisovellukset, verkkopalvelut ja tietojärjestelmät, konseptointi ja prototyyppisuunnittelu sekä integraatiopalvelut. Yrityksen suurimpia asiakkaita Mirkan lisäksi ovat

ABB, Danfoss ja Wärtsilä. (Devatus 2017.) Mirka Oy on perustettu vuonna 1943, ja nykyään se on yksi maailman suurimmista hiomatuotteiden valmistajista. Yrityksen tuotevalikoima koostuu hionta- ja kiillotustarvikkeista ja koneista kokonaisiin hiontajärjestelmiin. Mirkan pääkonttori sijaitsee Jepualla Uudessakaarlepyyssä. (Mirka 2017a.)

Tärinää voidaan mitata laitteeseen kiinnitettävällä anturilla, joka mittaa taajuuspainotettua kiihtyvyyttä x-, y-, ja z-akseleilta. (ISO 5349-1:2001.) Tutkimuksen tilaaja on asentanut hiomakoneisiinsa anturit tärinän mittausta varten. Antureilta saatava data välitetään mobiililaitteille Bluetooth-teknologian avulla, ja tärinäarvoja voidaan tarkastella *myMirka*-mobiilisovelluksen avulla. Sovellus näyttää hetkellisen tärinäaltistuksen reaaliajassa sekä tärinäkertymän vuorokauden ajalta. Tärinänseurannan lisäksi sovelluksella voidaan seurata hiomakoneen pyörintänopeutta. (Mirka 2016a.)

Tässä tutkimuksessa jatketaan kyseisen sovelluksen kehittämistä lisäämällä siihen ohjattu toiminto tuotteiden verifiointia ja validointia varten, sekä siitä saatavan tiedon keräykseen. Verifioinnilla pyritään varmistamaan, että tuote on valmistettu suunnitelman mukaisesti. Verifiointi vastaa kysymykseen *"valmistammeko tuotetta oikein?"*. Validoinnilla puolestaan varmistetaan tuotteen soveltuvuus suunniteltuihin käyttötarkoituksiin, ja se vastaa kysymykseen *"valmistammeko oikeaa tuotetta?".* (Ward, Clarkson, Bishop & Fox 2002: 2–3, Simpson 2015.) Jatkossa tässä työssä tuotteiden verifioinnista ja validoinnista puhutaan testauksena.

1.2 Tutkimusongelma ja tavoitteet

Työn tavoitteena on luoda sovellus, joka toteuttaa asiakkaan vaatimukset asetetulle ongelmalle. Työssä olennaisimpana osana on sovelluksen toiminnallisuus ja helppokäyttöisyys. Helppokäyttöisyyden saavuttamiseksi käyttöliittymä- ja käyttäjäkokemussuunnittelu ovat olennaisessa osassa työtä.

Tämä diplomityö toteutetaan konstruktiivisella tutkimusotteella, eli tutkimus pyrkii luomaan konkreettisen ratkaisun annettuun ongelmaan. Tutkimusongelma voidaan tiivistää seuraaviin kahteen tutkimuskysymykseen:

- Miten hiomatuotteiden testaus ja siitä syntyvän tiedon kerääminen voidaan toteuttaa tehokkaasti mobiilisovellusta käyttäen?
- Mitä teknologioita työn toteutuksessa tarvitaan?

1.3 Työn rakenne

Työn toisessa luvussa esitellään tutkimuksen viitekehys ja laajennettavan sovelluksen taustat. Kolmannessa luvussa esitellään työn toteutuksessa käytettävät laitteet ja teknologiat. Neljännessä luvussa kuvataan sovelluksen vaatimusmäärittely. Viidennessä luvussa esitellään suunnitelmat, joiden pohjalta sovellus toteutetaan. Kuudennessa luvussa kerrotaan sovelluksen toteutusvaiheesta, sen aikana kohdatuista ongelmista ja niiden ratkaisuista sekä sovelluksen toiminnasta. Seitsemännessä luvussa käsitellään sovelluksen testaus ja kahdeksannessa luvussa johtopäätökset.

2 MOBIILISOVELLUKSEN HYÖDYNTÄMINEN TUOTETESTAUKSESSA

Nykypäivänä esineiden internetiä, eli IoT:tä (*Internet of Things*) hyödyntäviä laitteita on käytössä yli 20 miljoonaa kappaletta, ja määrän on arvioitu kasvavan noin 75 miljoonaan vuoteen 2025 mennessä. (Statista 2018.) Esineiden internetiin verkottuneita laitteita voidaan ohjata ja diagnosoida etänä, ja niissä olevista antureista voidaan kerätä dataa eri tarkoituksia varten. Raakadataa analysoimalla esimerkiksi laitteiden käyttötottumukset voidaan ottaa paremmin huomioon tuotekehityksessä. (Oliana 2016.) Hiomapyöröjen tuotekehityksessä pyritään kehittämään esimerkiksi kestävyyttä, hiontatehokkuutta sekä vähentämään hionnasta aiheutuvan pölyn määrää. Parhaiten tietoa hiontatuotteiden soveltuvuudesta erilaisiin olosuhteisiin ja käyttötarkoituksiin saadaan teettämällä testaus eri sovellusalojen asiakkailla tuotteiden oikeissa käyttöympäristöissä. Tässä tutkimuksessa tarkasteltavassa testausprosessissa tuotteiden testaus teetetään asiakkailla ja niistä saatavat tulokset kerätään mobiilisovelluksen avulla.

Testauksessa pintaa hiotaan erilaisilla hiomapyöröillä, ja lopuksi testaaja arvioi niiden soveltuvuutta kyseiseen pintamateriaaliin. Testattavien tuotteiden mukana toimitetaan vertailutuotteet, joihin vertaamalla testaajat arvioivat tuotteiden ominaisuuksia. Käyttäjien antaman arvioinnin lisäksi testauksessa kerätään hiomakoneen anturien avulla esimerkiksi koneen kuluttamaa sähkövirtaa ja pyörintänopeutta. Numeerisen datan ja sanallisen palautteen avulla voidaan analysoida tuotteiden toimivuutta erilaisissa olosuhteissa ja soveltuvuutta erilaisiin käyttötarkoituksiin. Aiemmin testauksesta on saatu ainoastaan käyttäjien antama palaute, joten tässä työssä syntyvän sovelluksen avulla myös antureilta saatava raakadata saadaan talteen, ja sitä voidaan hyödyntää käyttäjäarvioiden kanssa yhä parempien tuotteiden kehittämiseen.

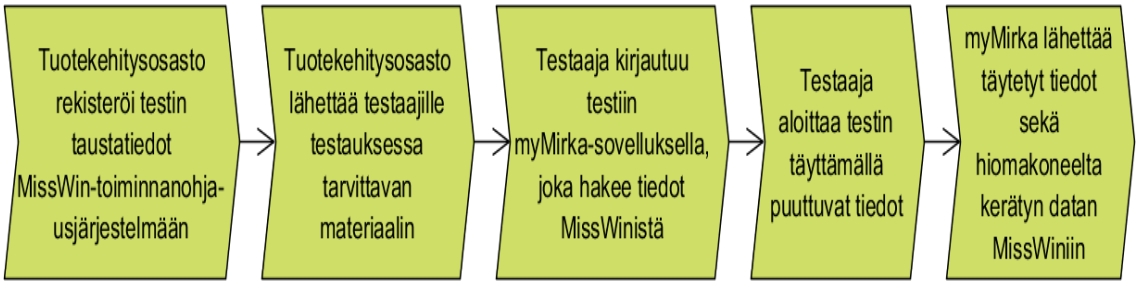
2.1 MyMirka-mobiilisovellus

Mirkan visiona on luoda pintakäsittelyteollisuuteen uusi digitaalinen tulevaisuus, jota varten se on lanseerannut myMirka-mobiilisovelluksen. Sovellus mahdollistaa hiomakoneista työntekijään kohdistuvan tärinän mittaamisen sekä hiomakoneen pyörintänopeuden seurannan. Sovellus osaa myös ohjeistaa kuinka tärinätasoa voidaan alentaa. Sovelluksesta on saatavilla myös maksullinen myMirka Pro -versio, joka sisältää ominaisuuden päivittäisen tärinäaltistuksen seuraamiseen. Sen avulla tärinäaltistusta voidaan tarkastella graafisessa muodossa viiden minuutin ja 30 päivän pituisina seurantajaksoina. (Mirka 2016a.) Tässä työssä syntyvä tuotetestausmoduuli tulee olemaan rajoitettu ominaisuus, joka on käytettävissä vain erikseen määritellyille testaajille.

2.2 Tarkasteltavan testausprosessin nykytila ja tulevaisuuden näkymät

Tähän asti testaus on suoritettu siten, että Mirkan tuotekehitysosasto kirjaa toiminnanohjausjärjestelmään taustatiedot testauksesta: testaajat, ohjeistuksen testausta varten, sekä tiedot testattavista tuotteista. Tämän jälkeen testaajat saavat sähköpostitse pyynnön testauksesta, jonka jälkeen Mirka toimittaa testaajille kaikki testauksessa tarvittavat tuotteet ja välineet sisältävän testaussarjan. Testaajat täyttävät ensin verkossa olevaan testiraporttilomakkeeseen esitiedot testauksesta, jonka jälkeen tuotteet testataan ohjeistuksen mukaisesti. Testauksen jälkeen testiraporttilomake täytetään loppuun: lomakkeessa määritellään tarkasti mm. ympäristön olosuhteet, käytetyt hiontamenetelmät sekä hiottavan pinnan ominaisuudet. (Mirka 2016b, Mirka 2017b.)

Tämän tutkimuksen tavoitteena on suunnitella ja toteuttaa testausprosessia ohjaava mobiilisovellus, joka yksinkertaistaa, nopeuttaa ja antaa yksityiskohteisempaa tietoa testauksen kulusta ja tuotteiden käyttäytymisestä. Testausprosessin ja tiedonkeruun yksinkertaistamiseksi sovellukseen luodaan ohjattu toiminto, jossa käyttäjää ohjeistetaan testauksen jokaisessa vaiheessa. Prosessin nopeuttamiseksi suuri osa tällä hetkellä manuaalisesti syötettävistä tiedoista voidaan hakea valmiiksi esimerkiksi tietokannasta tai hiomakoneen antureista. Kuvassa 1 tulevaisuuden testausprosessi on esitetty kaavion muodossa.



Kuva 1. Testausprosessin kulku mobiilisovellusta hyödyntäen (mukaillen Mirka 2016b).

Koska testaaja voi olla käytännössä kuka tahansa Mirkan tuotteiden kanssa työskentelevä, pyritään sovellus tekemään niin yksiselitteiseksi, että testausprosessi ei vaadi testaajalta aiempaa kokemusta kyseisestä testausprosessista tai myMirka-sovelluksesta. Mobiilisovelluksen lisäksi työssä luodaan tietokanta, johon testauksen tiedot tallennetaan. Tietokannan rakenne tullaan myöhemmin integroimaan osaksi MissWin-järjestelmää.

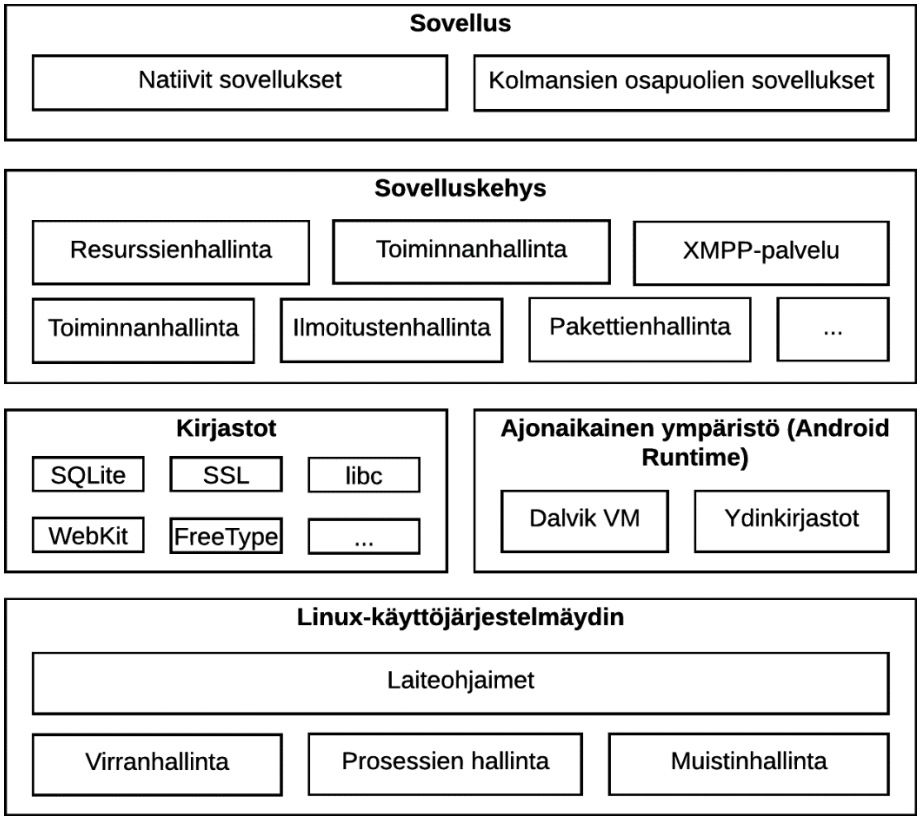
3 TYÖN TOTEUTUKSESSA KÄYTETTÄVÄT TEKNOLOGIAT JA LAITTEET

3.1 Soveltuvat mobiililaitteet

Nykyään mobiililaitevalmistajia on markkinoilla valtava määrä laitemalleista puhumattakaan. Tästä johtuen laitteiden ominaisuudet voivat poiketa hyvinkin paljon toisistaan. Tässä työssä toteutettavan sovelluksen käyttöön vaaditaan mobiililaite, joka täyttää vähintään seuraavat vaatimukset: Android-käyttöjärjestelmän versio 5.0, kosketusnäyttö, Bluetooth-vastaanotin ja pääsy Internetiin. Muilta osin laitteen tyypillä tai muilla ominaisuuksilla ei ole toiminnallisten vaatimusten kannalta merkitystä. Suurin osa markkinoilla olevista Android-mobiililaitteista täyttävät sovelluksen asettamat laitteistovaatimukset, joten käytettävän laitteen valinta riippuu pääasiassa käyttäjän mieltymyksistä. Suurinäyttöisen taulutietokoneen etuna on helppolukuisuus, mikäli sovelluksen mittareita seurataan laitteen näytöltä hionnan aikana. Toisaalta älypuhelimen pienempi koko helpottaa laitteen kuljetusta. Tässä työssä sovellusta kehittäessä käytettiin LG Leon älypuhelinta Android versiolla 6.0.

3.1.1 Android-käyttöjärjestelmä

Open Handset Alliancen ja Googlen kehittämä Android on tällä hetkellä maailman suosituin mobiilikäyttöjärjestelmä 85 % markkinaosuudellaan älypuhelimissa (IDC 2017). Se perustuu avoimen lähdekoodin Linux-käyttöjärjestelmäytimeen, ja sen arkkitehtuuripino koostuu viidestä eri kerroksesta. Nämä kerrokset on esitetty kuvassa 2. Alimmaisena Androidin arkkitehtuuripinossa on Linux-käyttöjärjestelmäydin, eli Linux-kerneli, joka sisältää kaikki käyttöjärjestelmän toiminnot ja toimii abstraktiokerroksena laitteiston ja sovelluskerroksen välissä. (Smyth 2016: 85–92.)



Kuva 2. Android arkkitehtuuripino (mukaillen Smyth 2016: 86).

Toiseksi alimpaan kerrokseen kuuluvat natiivi kirjasto sekä suoritusympäristö. Natiivi kirjasto koostuu sovellusten kehittämisessä tarvittavista kirjastoista, kuten esimerkiksi C/C++ -kirjastosta (*libc*), OpenGL ES -grafiikkakirjastosta ja SQLite-tietokantakirjastosta. Suoritusympäristö sisältää joko Dalvik-virtuaalikoneen tai ART-suoritusympäristön (*Android Runtime Environment*) sekä Java-ydinkirjastot. (Smyth 2016: 86–90.)

Dalvik-virtuaalikoneen ansiosta sovelluksia voidaan suorittaa samanaikaisesti ja tehokkaasti ns. ilmentyminä. Esimerkiksi sovelluksen kaatuminen ei näin ollen vaikuta muihin suoritettaviin sovelluksiin. Virtuaalikoneessa sovelluksen kääntämisvaiheessa luotu Dalvik-tavukoodi, tai sovelluksen tarvitsema osa siitä, käännetään ajon aikana (engl. *Just-In-Time*, JIT) suorittimen ymmärtämälle konekielelle. (Smyth 2016: 87–88.) Ajonaikainen kääntäminen nopeuttaa sovellusten suorittamista laitteissa, joissa laitteiston resurssit ovat rajalliset (Ehringer 2010).

Dalvik-virtuaalikone tuli käyttöön ensimmäisen kerran Android-versiossa 2.2 (*Froyo*), mutta versiossa 5.0 (*Lollipop*) se korvattiin ART-suoritusympäristöllä, joka noudattaa ennenaikaista kääntämistä (engl. *Ahead-Of-Time*, AOT) (Panigrahy 2015). Tällöin koko sovellus käännetään sovelluksen asennusvaiheessa kokonaisuudessaan. AOT-kääntämisen etuina on sovelluksien nopeampi käynnistysaika ja tehokkaampi virrankäyttö, koska sovellusta ei tarvitse kääntää uudelleen jokaisella suorituskerralla. (Thakur 2015.)

Toiseksi ylin kerros on sovelluskehys, joka muodostaa erilaisten palveluiden avulla ympäristön sovellusten suorittamiselle ja hallinnalle. Sovelluskehyksen palvelut ja kuvaukset niiden toiminnasta on listattu taulukossa 1.

Taulukko 1. Androidin sovelluskehyksen palvelut (Smyth 2016: 91–92).

Palvelu	**Kuvaus**
Toiminnanhallinta (Activity Manager)	Hallitsee sovellusten elinkaarta ohjaamalla käyttäjälle näkyviä toimintaikkunoita eli aktiviteetteja.
Palveluntarjoajat (Content Providers)	Mahdollistavat sovellusten keskinäisen tiedonjakamisen.
Resurssienhallinta (Resource Manager)	Tarjoaa pääsyn lähdekoodin ulkopuolisiin resursseihin kuten käyttöliittymä- ja grafiikkatiedostoihin.
Ilmoitusten hallinta (Notification Manager)	Sallii sovellusten näyttää käyttäjälle ilmoituksia ja varoituksia.
Näkymienhallinta (View System)	Tarjoaa erilaisia komponentteja graafisen käyttöliittymän luontiin.
Pakettienhallinta (Package Manager)	Hoitaa sovelluksien asentamisen ja päivittämisen sekä tarjoaa tietoa asennetuista sovelluksista.
Sijaintipalvelu (Location Manager)	Mahdollistaa sovellusten vastaanottaa tietoa sijainnista.

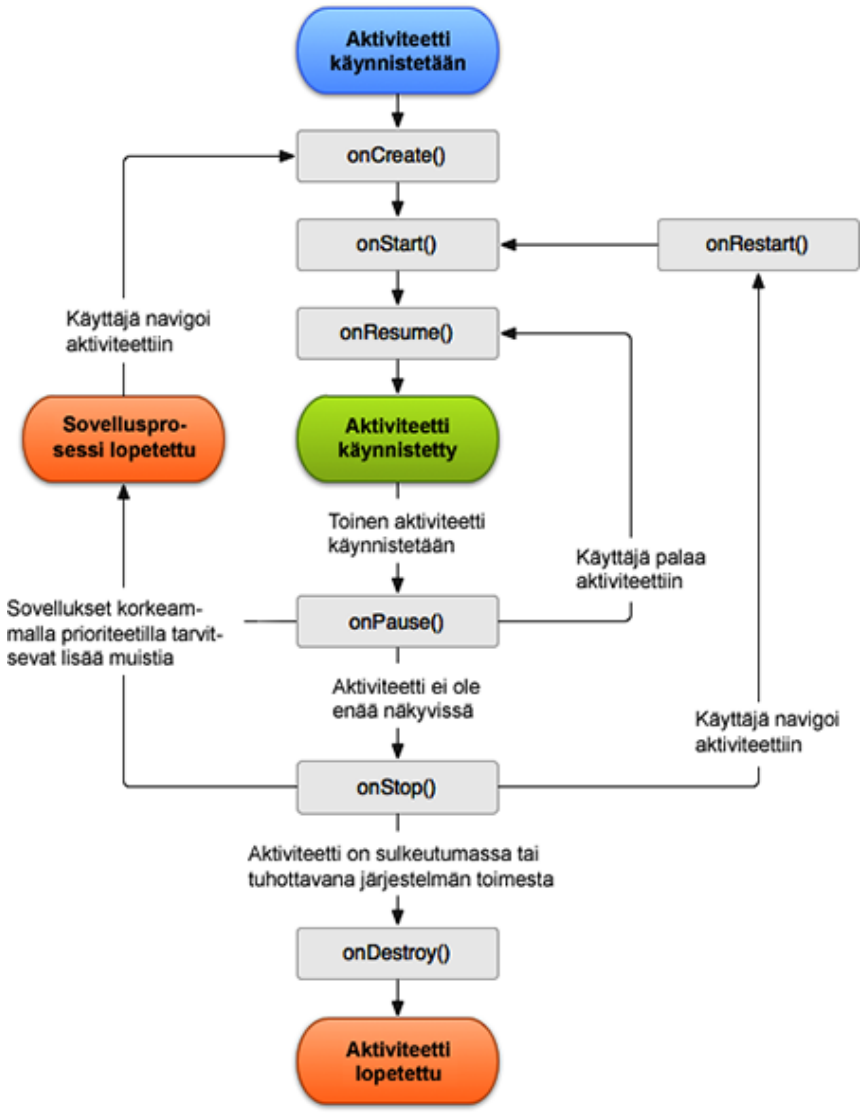
Puhelinpalveluiden hallinta (Telephony Manager)	Tarjoaa sovelluksille tietoa puhelinverkon tilasta ja mahdollistaa puhelinyhteyden käytön.

Arkkitehtuuripinon ylimpänä kerroksena on sovelluskerros, johon kuuluvat esiasennetut, eli käyttöjärjestelmän mukana tulevat sovellukset, sekä käyttäjän asentamat kolmansien osapuolien sovellukset. (Smyth 2016: 92.)

3.1.2 Androidin aktiviteetit ja niiden elinkaari

Yksi Androidin oleellisimmista sovelluskomponenteista on aktiviteetti (engl. *activity)*, jonka tehtävänä on luoda ikkuna näkymää varten. Aktiviteettien avulla luotuja ikkunoita käytetään useimmiten koko näytön kokoisina, mutta ne voivat olla myös ns. kelluvia ikkunoita. Aktiviteetteja hallitaan pinorakenteessa, jossa viimeksi luotu lisätään pinon ylimmäiseksi. Aktiviteetilla on neljä tilaa: käynnistetty (engl. *running*), aktiivinen (engl. *active*), pysäytetty (engl. *paused*) ja keskeytetty (engl. *stopped*).

Aktiviteetin suoritus tapahtuu *onCreate*- ja *onDestroy*-metodien suorituksen välillä, ja näkyvissä oleva toiminta *onStart*- ja *onStop*-metodien välillä. Käyttäjän vuorovaikutus puolestaan on mahdollista ainoastaan *onResume*-metodin kutsusta lähtien *onPause*-metodin kutsumiseen asti. (Google and Open Handset Alliance (N.d.) 2017a.) Aktiviteetin tilat ja siirtymät niiden välillä on esitetty kuvassa 3.



Kuva 3. Androidin aktiviteetin elinkaari. Muokattu lähdettä Google and Open Handset Alliance (N.d.) 2017a.

Aktiviteetin elinkaaren ensimmäinen vaihe on sen käynnistäminen, jolloin aktiviteetissa suoritetaan onCreate-, onStart- ja onResume -metodit. OnCreate-metodissa aktiviteetti luodaan ensimmäistä kertaa, ja onStart-metodi suoritetaan siinä vaiheessa, kun aktiviteetti tulee näkyviin. Aktiviteetin tila muuttuu käynnistetyksi ja käyttäjän vuorovaikutus sen kanssa voi alkaa, kun onResume-metodi on suoritettu. Käynnistetty-tilasta lopetettu-tilaan siirtyessä aktiviteetti voi suorittaa onPause-, onStop- ja onDestroy-metodit. OnPausea kutsutaan esimerkiksi silloin, kun käyttäjän vuorovaikutusta ei havaita riittävän pitkään aikaan ja järjestelmä himmentää näytön kirkkautta. OnStop-metodia kutsutaan silloin, kun aktiviteetti ei ole enää näkyvillä – esimerkiksi toisen aktiviteetin käynnistyessä.

Aktiviteetin elinkaari päättyy sen tuhoamiseen joko käyttäjän tai järjestelmän toimesta. Ennen tuhoutumista on mahdollista suorittaa komentoja onDestroy-metodin sisällä. (Google and Open Handset Alliance (N.d.) 2017a.)

Aktiviteettien sisään voidaan sulauttaa *fragmenteja*, jotka voidaan ajatella eräänlaisina aliaktiviteetteina. Fragmenteilla on niitä käyttävistä aktiviteeteista riippumaton elinkaari, jonka ansiosta niitä voidaan luoda ja tuhota aktiviteetin suorituksen aikana. Fragmenteja voidaan myös uudelleen käyttää eri aktiviteeteissa itsenäisinä komponentteina. (Google and Open Handset Alliance (N.d.) 2017b.)

3.2 Xamarin Platform -kehitysympäristö

Xamarin Platform on kehitysympäristö järjestelmäriippumattomaan sovelluskehitykseen. Se perustuu avoimen lähdekoodin Mono-kehitysympäristöön, joka sisältää C#-kääntäjän, CLR-suoritusympäristön (*Common Language Runtime*) sekä luokkakirjaston. Sen avulla voidaan kehittää mobiilisovelluksia Android-, iOS- ja Windows-mobiilialustoille C#-ohjelmointikielellä sekä hyödyntäen Microsoftin .NET-sovelluskehystä. Xamarinilla toteutetut sovellukset käännetään kullekin käyttöjärjestelmälle natiiveiksi sovelluksiksi. Käyttöliittymää ei kuitenkaan voida suoraan kääntää jokaiselle käyttöjärjestelmälle sopivaksi, vaan se on toteutettava jokaiselle erikseen. Tästä huolimatta lähdekoodin uudelleenkäyttöaste lisääntyy huomattavasti: Xamarin Inc:n mukaan keskimäärin jopa 75 % lähdekoodista voidaan jakaa eri laitealustojen kesken. Xamarin Platformia voidaan käyttää sekä Windows- että macOS-ympäristöissä Visual Studio -kehitysympäristön lisäosana. (Reynolds 2014: 21–24, Xamarin 2017.)

3.3 Laitteiden välinen langaton kommunikaatio

Hiomakoneen ja mobiililaitteen tulee kommunikoida keskenään, jotta antureista saatava tieto saadaan käsiteltäväksi sovellukseen. Tässä työssä laitteiden väliseen kommunikaatioon käytetään Bluetoothia, joka on standardi laitteiden väliseen langattomaan lyhyen

kantaman tiedonsiirtoon, ja likiverkkojen (engl. *Personal Area Network*, PAN) luomiseen. Ruotsalainen telekommunikaatioyritys Ericsson aloitti sen kehityksen vuonna 1994 tavoitteenaan luoda edullinen ja pienellä virralla toimiva yhteys matkapuhelinten ja niiden lisävarusteiden välille (Thompson, Kline & Kumar 2008: 4). Vuonna 1998 perustettiin Bluetooth Special Interest Group (Bluetooth SIG), jonka päämääränä on kehittää ja laajentaa Bluetooth-konseptia. Bluetooth SIG koostuu nykyään yli 30 000 jäsenyrityksestä (Bluetooth SIG 2016).

Mirkan DEROS- ja DEOS-sarjojen hiomakoneissa käytetään Bluetooth LE (*Low Energy*) -teknologiaa, jonka suurimpana erona perinteiseen Bluetooth-teknologiaan on pienempi virrankulutus. Bluetooth LE -lähetin-vastaanotin voidaan kytkeä lepotilaan kun yhteyttä ei tarvita tiedonsiirtoon. Perinteinen Bluetooth-teknologia päihittää Bluetooth LE:n tiedonsiirtonopeudessa, jonka ansiosta se soveltuu paremmin esimerkiksi median suoratoistoon. Matalan virrankulutuksen ansiosta Bluetooth LE soveltuu erinomaisesti akku- ja paristokäyttöisiin laitteisiin, joiden tarvitsee siirtää ainoastaan pienikokoisia datapaketteja kerrallaan. Suosittuja sovelluskohteita ovat älykellot ja paikannuslaitteet. (CSR 2010.)

3.4 Tiedon varastointi ja siirto

Kaikki testauksessa kerätty tieto tallennetaan Internet-yhteyden välityksellä tietokantaan myöhempää tarkastelua varten. Lähetettävä tieto muunnetaan JSON-muotoon ja lähetetään Node.js:llä toteutettuun rajapintaan. Rajapinta käsittelee sovelluksesta vastaanotetut tiedot ja lähettää ne edelleen erilliselle tietokantapalvelimelle. Kaikki kommunikaatio mobiilisovelluksen ja palvelinten välillä tapahtuu REST-arkkitehtuurimallin mukaisilla viesteillä.

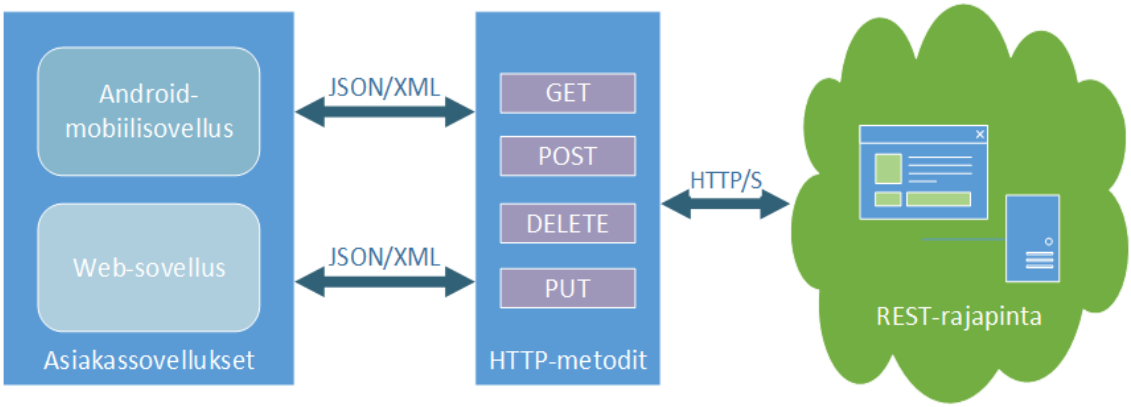
3.4.1 Tietokantaohjelmisto

Tietokantapalvelin käyttää Microsoftin kehittämää SQL Server 2012 -relaatiotietokanta-järjestelmää, joka perustuu Transact-SQL -kyselykieleen (T-SQL). Transact-SQL on laajennettu versio IBM:n kehittämästä ANSI-standardoidusta SQL-kyselykielestä (*Structured Query Language*). Suurimpina eroina T-SQL:n ja perinteisen SQL:n välillä on T-SQL:n tuomat mahdollisuudet proseduraaliseen ohjelmointiin, paikalliset muuttujat sekä laajempi joukko erilaisia funktioita datan käsittelyyn. Lisäksi T-SQL:ssä DELETE- ja UPDATE-komentojen toteutus poikkeaa ANSI-SQL:stä. (Bajaj 2011.)

3.4.2 REST-arkkitehtuurimalli

REST (*REpresentational State Transfer*) on arkkitehtuurimalli, jonka avulla voidaan toteuttaa yhteinen rajapinta hajautettujen hypermediajärjestelmien välille asettamatta rajoitteita yksittäisiin järjestelmiin. Nykyaikaiseen web-arkkitehtuuriin soveltuvan REST-mallin esitteli Roy Fielding Kalifornian yliopistossa tekemässä väitöskirjassaan vuonna 2000. REST-malli perustuu rajoitteisiin, joita arkkitehtuurin tulee noudattaa ollakseen REST-mallin mukainen.

Arkkitehtuurimallin kuusi rajoitetta ovat asiakas-palvelin -arkkitehtuurityyli, tilattomuus, välimuisti, yhdenmukainen rajapinta, kerroksittainen järjestelmä ja ladattava koodi. (Fielding 2000: 75–86.) Kuvassa 4 on esitetty REST-arkkitehtuurimallin idea pääpiirteittäin.



Kuva 4. REST-arkkitehtuurimalli (mukaillen Kiran 2014).

Asiakas-palvelin -arkkitehtuurityylin perusajatuksena on jakaa ohjelmiston osien vastuu asiakas- ja palvelinkomponenteille. Tämän ajatuksen mukaan esimerkiksi käyttöliittymän esittäminen on asiakaskomponentin vastuulla ja tiedon varastointi palvelinkomponentin vastuulla. Tällä tavoin asiakas- ja palvelinkomponentteja voidaan kehittää toisistaan riippumatta, kunhan niiden välinen rajapinta pysyy muuttumattomana. (Fielding 2000: 78.)

Tilattomuusrajoitteen mukaan jokaisen asiakkaan palvelimelle lähettämän pyynnön tulee sisältää kaikki pyynnön käsittelemiseksi tarvittavat tiedot. Rajoite lisää skaalautuvuutta, näkyvyyttä ja luotettavuutta. Skaalautuvuus paranee, sillä palvelimen ei tarvitse tallettaa tilaa eri pyyntöjen välillä. Näkyvyys lisääntyy, sillä palvelin saa selville pyynnön tarkoituksen suoraan vastaanotetusta pyynnöstä. Luotettavuus puolestaan paranee, koska rajoite helpottaa osittaisista virheistä toipumista. (Fielding 2000: 78–79.)

Verkon suorituskyvyn parantamiseksi RESTissä käytetään välimuisti-rajoitetta. Tämä tarkoittaa käytännössä sitä, että palvelimen lähettämästä vastauksesta on tultava ilmi, voidaanko tieto tallettaa välimuistiin. Asiakassovellus voi palvelimen salliessa hyödyntää välimuistiin tallennettua tietoa myöhemmin lataamatta sitä palvelimelta uudestaan. Tämä vähentää identtisten pyyntöjen lähettämistä joko osittain tai kokonaan, ja näin ollen parantaa asiakkaan ja palvelimen välisen verkon suorituskykyä sekä vähentää asiakasohjelmassa näkyvää viivettä. Välimuistia käytettäessä haittapuolena on luotettavuuden heikentyminen, sillä välimuistiin tallennettu tieto voi vanhentua. (Fielding 2000: 79–80.)

Keskeinen osa REST-mallia, joka erottaa sen muista verkkopohjaisista arkkitehtuurityyleistä, on yhdenmukainen rajapinta komponenttien välillä. Sen ansiosta järjestelmän arkkitehtuuri yksinkertaistuu ja komponenttien välisen vuorovaikutuksen näkyvyys lisääntyy. Yhdenmukainen rajapinta -rajoite mahdollistaa komponenttien itsenäisen kehittämisen, mutta heikentää järjestelmän suorituskykyä. Suorituskyvyn heikkeneminen johtuu standardoidusta tavasta siirtää tietoa komponenttien välillä, jolloin kyseinen muoto ei välttämättä ole paras mahdollinen yksittäisen komponentin näkökulmasta. (Fielding 2000: 81–82.)

Kerroksittainen järjestelmä -rajoite määrittelee arkkitehtuurin koostuvan hierarkkisista kerroksista, joista jokainen voi nähdä vain tasoa ylemmän ja alemman kerroksen, joiden kanssa kyseinen kerros on välittömässä vuorovaikutuksessa. Järjestelmän osien näkyvyyden rajoittaminen yksittäisiin kerroksiin mahdollistaa kerrosten käyttämisen järjestelmän kapselointiin. Kapseloinnilla voidaan esimerkiksi estää vanhentuneiden palvelinkomponenttien ja asiakassovellusten käyttö (Fielding 2000: 82–83). Käytännössä kerroksittainen järjestelmä mahdollistaa esimerkiksi välityspalvelimen käytön asiakkaan ja palvelimen välillä ilman rajapinnan muuttamista (Systä 2009). Huonona puolena kerroksittaisessa järjestelmässä on tiedon käsittelyyn kuluvan ajan lisääntyminen, josta johtuva heikentynyt suorituskyky näkyy käyttäjälle viiveenä (Fielding 2000: 83).

Kuudes ja viimeinen rajoite on ladattava koodi (engl. *Code-On-Demand*), jonka tarkoituksena on mahdollistaa asiakassovelluksen toiminnallisuuksien lisääminen sallimalla pienimuotoisten komentosarjojen ja sovelmien, kuten Javascript- tai Flash-sovellusten, lataaminen palvelimelta. Tällöin asiakassovelluksen laajentaminen on joustavampaa, kun toiminnallisuutta voidaan tarpeen vaatiessa lisätä toteuttamatta sitä asiakassovelluksen sisällä. Tämän rajoitteen käyttö kuitenkin huonontaa järjestelmän näkyvyyttä, ja näin ollen johtaa ristiriitaan näkyvyyttä parantavien rajoitteiden kanssa. Tästä syystä Fielding määrittelee rajoitteen valinnaiseksi, sillä sen tuomat hyödyt tulevat esiin vain järjestelmissä, joissa ladattavaa koodia tarvitaan. (Fielding 2000: 84–85.)

3.4.3 JSON-tiedostomuoto

JSON eli JavaScript Object Notation on tiedonsiirtoon tarkoitettu tiedostomuoto, joka perustuu JavaScript-ohjelmointikieleen ja ECMA-262-standardiin. JSON standardoitiin vuonna 2013 ECMA-404 -standardilla. Formaatti on täysin riippumaton ohjelmointikielestä, vaikka sen notaatiossa onkin paljon yhteneväisyyksiä C-ohjelmointikielen sukuisiin kieliin. (ECMA-404.)

JSON perustuu kahteen yleiseen rakenteeseen: avain-arvo -pareista koostuviin järjestämättömiin kokoelmiin sekä arvoja sisältäviin järjestettyihin listoihin. Järjestämätön kokoelma vastaa käytännössä olioperustaisten ohjelmointikielien oliota, sanakirjaa (engl.

dictionary), hajautustaulua (engl. *hash table*) tai muuta vastaavaa tietorakennetta. Järjestetty lista vastaa ohjelmointikielien taulukkoa, vektoria tai listaa. (ECMA-404.)

JSON-olio muodostetaan aaltosulkuja käyttämällä: aaltosulkujen sisällä esitetään avain, ja sen jälkeen arvo. Avain ja arvo erotetaan toisistaan kaksoispisteellä, ja avain-arvo -parit pilkuilla.

Esimerkki JSON-muotoisesta oliosta:

```
{
  ”avain1”: ”arvo1”,
  ”avain2”: ”arvo2”,
  ”avain3”: 3,
  ”avain4”: 4
}
```

JSON-taulukko muodostetaan hakasuluilla, joiden sisällä esitetään arvot pilkuilla erotettuina:

```
[ ”arvo1”, 2, ”arvo3”, ”arvo4”, 5, 6 ]
```

Näitä kahta rakennetta voidaan yhdistellä halutun rakenteen saavuttamiseksi, ja niiden yleismaailmallisuudesta johtuen JSON-muotoinen data pystytään helposti muuttamaan käytetyn ohjelmointikielen syntaksin mukaiseen esitysmuotoon sekä päinvastoin. (ECMA-404.) Tässä työssä kaikki data asiakas- ja palvelinsovelluksen välillä siirretään JSON-muodossa.

3.4.4 Node.js-sovellusalusta

Node.js on Googlen V8 -JavaScript moottoriin perustuva palvelinpuolen suoritusympäristö, joka on suunnattu pitkäaikaisten palvelinprosessien ajamiseen. Javascript tunnetaan perinteisesti selaimessa suoritettavana ohjelmointikielenä, mutta Node.js mahdollistaa palvelinsovellusten luomisen Javascriptillä tarjoten samalla alustan niiden suorittamiseen ilman erillistä palvelinohjelmistoa. (Tilkov & Vinoski 2010: 80–81.)

Toisin kuin suurin osa muista moderneista sovellusalustoista, Node.js:ssä samanaikaisten käskyjen suorittaminen pohjautuu asynkroniseen tapahtumaperustaiseen tiedonsiirtomalliin (engl. *asynchronous I/O event model)* monisäikeisyyden (engl. *multithreading)* sijaan. Tämä tekee siitä kevyen ja helposti skaalautuvan alustan esimerkiksi aiemmin esitellyn REST-mallin mukaisiin rajapintatoteutuksiin. (Tilkov & Vinoski 2010: 80.) Tässä työssä Node.js:ää käytetään asiakassovelluksen ja tietokantapalvelimen välisen rajapinnan toteutuksessa. Asiakassovellus lähettää rajapintaan GET- ja POST-kutsuja tarpeen mukaan, jolloin rajapinta toteuttaa kunkin käskyn mukaisen toiminnon tietokantapalvelimelle, ja lopuksi lähettää tietokannasta saadun vastauksen takaisin asiakassovellukselle.

4 SOVELLUKSEN VAATIMUSMÄÄRITTELY

Tässä luvussa käsitellään sovelluksen vaatimusmäärittely. Määrittelyllä tarkoitetaan prosessia, jossa selvitetään projektin tarpeellisuus ja toteuttamiskelpoisuus. Yleensä määrittely jaetaan asiakasvaatimusten ja ohjelmistovaatimusten selvittämiseen. Ohjelmistovaatimusten pohjalta voidaan luoda toiminnallinen määrittely, jossa kuvataan järjestelmä ja sille asetetut vaatimukset. Määrittelyn ja suunnittelun apuna voidaan käyttää erilaisia kaavioita ja kuvia tekstin tukena. Tässä työssä suunnitelmia havainnollistetaan OMG:n (*Object Management Group*) vuonna 1997 standardoimaa UML-piirtotekniikkaa (*Unified Modeling Language*) käyttäen (Haikala & Märijärvi 2000: 89).

Sovellukselle ei työn alkaessa ollut olemassa selvää vaatimusmäärittelyä, vaan määrittelyä täydennettiin sovelluksen prototyyppien perusteella. Sovelluksen vaatimukset tarkentuivat ja muuttuivat työn edetessä monta kertaa ennen lopullista muotoaan.

4.1 Toiminnallinen ja ei-toiminnallinen määrittely

Toiminnallinen määrittely kuvaa ohjelmiston toiminnot ja toteutuksen vaatimukset sekä rajoitukset. Toiminnallisia vaatimuksia ovat esimerkiksi käyttöliittymä, ohjelmiston ominaisuudet ja kommunikaatio muiden järjestelmien kanssa. Toiminnallisten vaatimusten lisäksi määritellään ei-toiminnalliset vaatimukset, joita ovat esimerkiksi ohjelmiston suorituskykyyn ja käytettävyyteen liittyvät seikat. (Haikala & Märijärvi 2000: 26–27.) Alla on listattu määrittelyprosessin tuloksena mobiilisovellukselle asetetut vaatimukset.

Sovellukselle asetetut toiminnalliset vaatimukset:

- Sovelluksessa tulee olla mahdollisuus aloittaa testi ennalta määritellyn tunnisteen perusteella.
- Sovelluksen täytyy osata kommunikoida hiomakoneen kanssa.
- Sovelluksen tulee tallentaa testin aikana hiomakoneen antureilta kerätty tieto.
- Sovelluksessa tulee olla mahdollisuus arvioida testattavat tuotteet.

- Sovelluksen käyttöliittymän tulee olla selkeä ja helppokäyttöinen.

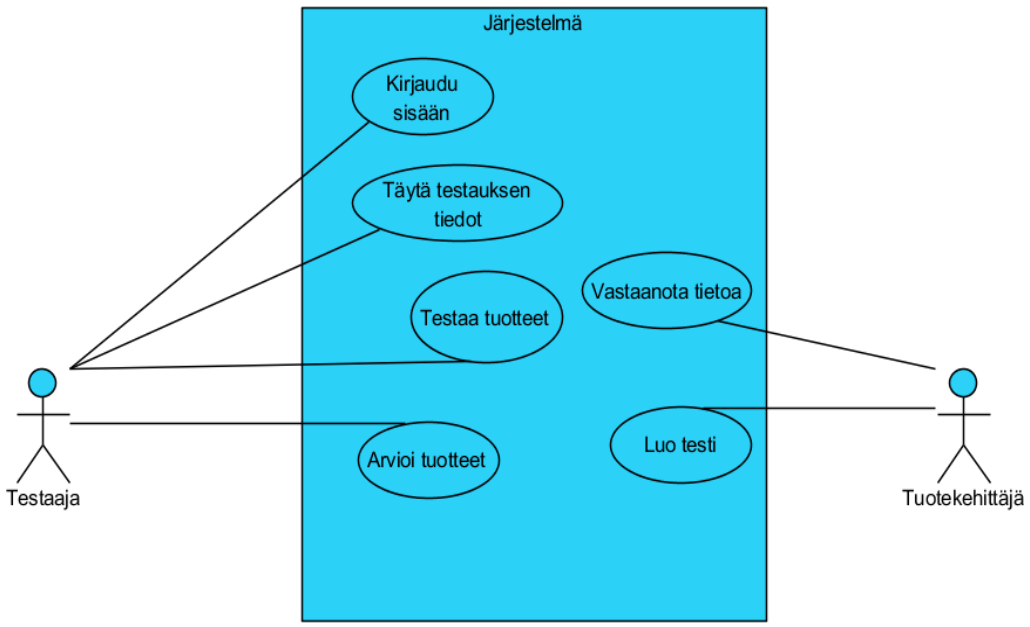
Sovellukselle asetetut ei-toiminnalliset vaatimukset:

- Sovelluksessa ei saa olla suuria vasteaikoja pyyntöjen välillä.
- Sovelluksen tiedonsiirrossa on otettava huomioon tietoturvallisuustekijät.

4.2 Käyttötapaukset

Käyttötapauksilla kuvataan järjestelmän toiminnallisuus ja käyttäjäroolit. Käyttäjärooli voi olla henkilö tai toinen järjestelmä. Käyttötapauksilla pyritään kuvaamaan asiakasvaatimukset, ja vältetään ottamasta kantaa varsinaiseen toteutukseen. Käyttötapaukset voidaan kuvata käyttötapauskaaviolla (engl. *use case diagram).* (Haikala & Märijärvi 2000: 141–145.)

Kuvassa 5 on UML-notaation mukaisesti kuvattu sovelluksen käyttötapauskaavio. Sovelluksen käyttäjärooleja ovat testaaja ja tuotekehittäjä, ja mahdollisia käyttötapauksia on kuusi.



Kuva 5. Sovelluksen käyttötapauskaavio.

Käyttötapaus 1: *Kirjaudu sisään*

Testaaja kirjautuu ennalta määrätyllä testitunnuksella, jolloin tietokannasta haetaan valmiiksi määritellyt tiedot tunnuksen perusteella.

Käyttötapaus 2: *Täytä testauksen tiedot*

Testaaja täyttää testaukseen liittyvät taustatiedot, kuten tiedot hiottavasta pinnasta, laitteistosta ja hiontatavasta. Tiedot on täytettävä ennen varsinaisen testauksen aloittamista.

Käyttötapaus 3: *Testaa tuotteet*

Testaajan pitää pystyä käyttämään sovelluksessa testaustilaa, jolloin sovellus tallentaa hiomakoneelta saatavan datan talteen ja lähettää ne tietokantaan.

Käyttötapaus 4: *Arvioi tuotteet*

Testaajan tulee pystyä arvioimaan testatut tuotteet valmiin lomakkeen avulla.

Käyttötapaus 5: *Vastaanota tietoa*

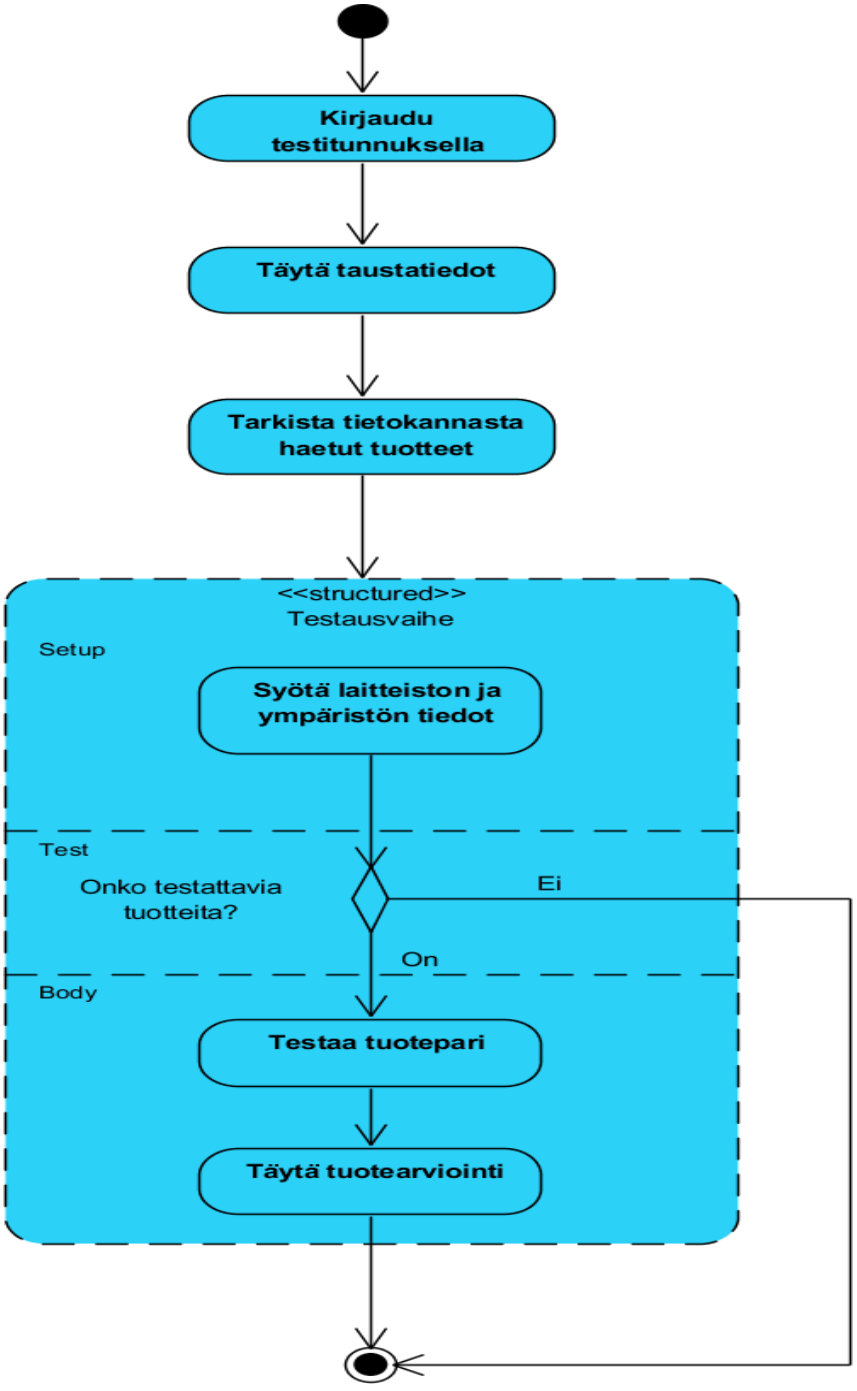
Tuotekehittäjän tulee pystyä tarkastelemaan mobiilisovelluksen lähettämiä tietoja jälkikäteen.

Käyttötapaus 6: *Luo testi*

Tuotekehittäjän tulee pystyä luomaan järjestelmään testitapaus, johon testaaja kirjautuu mobiilisovelluksella.

Yllä esitetyistä käyttötapauksista voidaan johtaa aktiviteettikaavio, joka kuvaa testaajaroolin käyttötapaukset aktiviteetteina ja näyttää niiden väliset tilasiirtymät. Kuva 6 esitetään testausprosessin kulku aktiviteettikaaviona. Kaaviossa toinen käyttötapaus on jaettu

kahteen osaan: täytä taustatiedot sekä syötä laitteiston ja ympäristön tiedot -aktiviteetteihin. Näistä jälkimmäinen, sekä käyttötapaukset kolme ja neljä muodostavat silmukan, joka kuvaa vaihetta, jossa suoritetaan varsinainen tuotetestaus. Silmukan sisään merkityt aktiviteetit suoritetaan jokaiselle testitunnukselle kirjatulle tuoteparille erikseen.



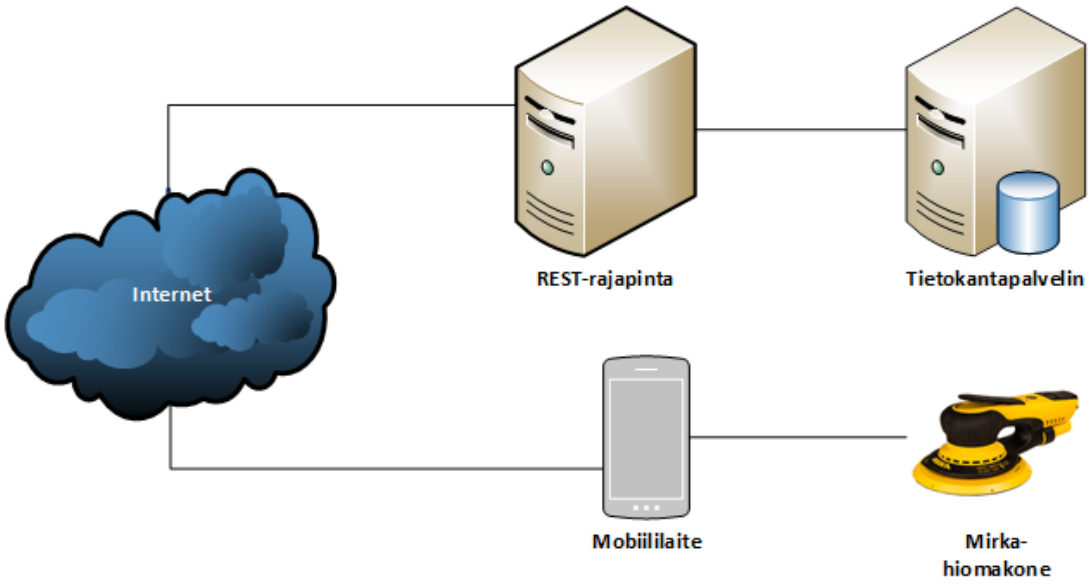
Kuva 6. Sovelluksen aktiviteettikaavio.

5 SOVELLUKSEN SUUNNITTELU

Tässä luvussa esitellään sovelluksen suunnitteluun liittyvät vaiheet ja ratkaisut. Suunnittelulla tarkoitetaan prosessia, jossa määrittelyvaiheen tulokset muunnetaan teknisen toteutuksen kuvaukseksi. Suunnittelu voidaan jakaa pienempiin kokonaisuuksiin, joista laajin kuvaus on arkkitehtuurisuunnittelu ja suppein moduulisuunnittelu (Haikala & Märijärvi 2000: 66). Tässä työssä suurin osa suunnittelusta oli tietomalli- ja käyttöliittymäsuunnittelua, sillä työn tuloksena syntynyt sovelluksen osa liittyy suurempaan kokonaisuuteen, ja täten sovelluksessa pyrittiin kunnioittamaan aiemmin tehtyjä arkkitehtuurisia päätöksiä.

5.1 Järjestelmän arkkitehtuuri

Kuvassa 7 on kuvattu tässä työssä käsiteltävän järjestelmän arkkitehtuuri. Se koostuu mobiilisovelluksesta, hiomakoneesta, REST-rajapinnasta sekä tietokannasta. Hiomakone kommunikoi mobiilisovelluksen kanssa Bluetooth-yhteyden avulla lähettäen tietoa koneen käytöstä. Mobiilisovellus lähettää tiedot REST-rajapinnalle, joka hoitaa lopulta tiedon lähettämisen tietokantaan.

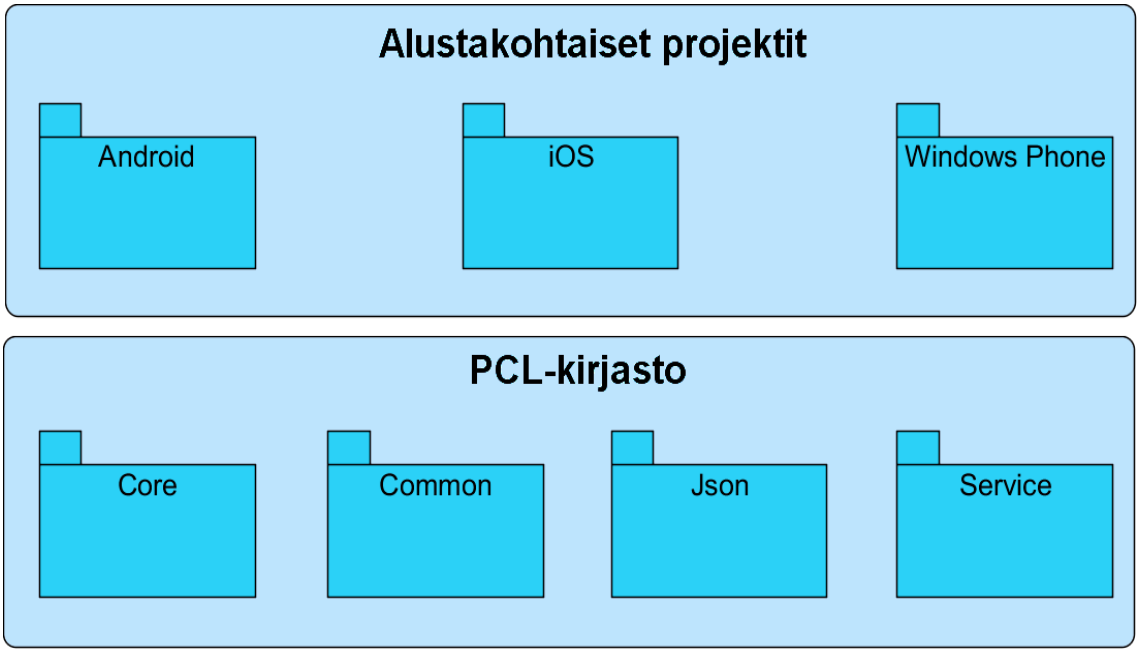


Kuva 7. Järjestelmän arkkitehtuuri.

5.2 Mobiilisovelluksen arkkitehtuuri

Arkkitehtuurisuunnittelussa kuvataan järjestelmän moduulit ja niiden rajapinnat, yleisrakenne ja toteutusfilosofia. Sen perimmäisenä tarkoituksena onkin määritellä järjestelmälle rakenne ja käytännöt, jotka luovat pohjan ymmärrettävän, ylläpidettävän, laajennettavan ja skaalautuvan ohjelmiston toteutukselle (Haikala & Märijärvi 2000: 69–71).

Sovelluksen arkkitehtuuri koostuu alustakohtaisista projekteista sekä PCL-kirjastosta (*Portable Class Library*). Kuvassa 8 on esitetty sovelluksen arkkitehtuuri kaaviona, josta nähdään PCL-kirjaston koostuvan *Core*-, *Common*, *Service*- ja *Json*-projekteista.



Kuva 8. Sovelluksen arkkitehtuuri.

Core-projekti sisältää rajapinnat Common-, Service- ja Json-projekteille, jotta niiden sisältämiä komponentteja voidaan käyttää alustakohtaisissa projekteissa. Common-projekti koostuu pääasiassa näkymämalleista, tietomalleista ja komponenteista Bluetooth- ja verkkoyhteyksien luomiseen. Service-projekti sisältää niin ikään sovelluksen ja palvelimen välisen verkkoyhteyden muodostamisessa tarvittavia palveluluokkia. Json-projektissa on ainoastaan JSON-muotoisen datan muuntamiseen eli serialisointiin tarvittavat komponentit.

5.3 Järjestelmän tietomalli

Tietomalli kuvaa järjestelmässä käsiteltävien tietojen esitystavan. Tietomallissa kuvataan asiat eli entiteetit, jotka voivat olla esimerkiksi ohjelmiston luokkia, tietokannan tauluja tai molempia. Entiteeteillä voi olla ominaisuuksia, eli attribuutteja, sekä suhteita muihin entiteetteihin. (Maciaszek & Lion 2005: 42–45.) Hiomatuotteiden testauksessa kerättävää tietoa on paljon, joten tietomallin rakenteella on tärkeä rooli selkeän ja tehokkaan tiedonkäsittelyn takaamiseksi. Järjestelmän tietomalli koostuu kahdeksasta entiteetistä, ja se on esitetty ER-kaaviona (*Entity-Relationship Model*) kuvassa 9. Tässä luvussa esitetään tietomallin entiteetit ja niiden attribuutit sekä suhteet muihin entiteetteihin.

Entiteetti *Fieldtest_data:*

Entiteetin attribuutit sisältävät perustiedot testistä: testaajan yhteystiedot, yrityksen tiedot sekä yksilöllisen tunnuksen (Id), joka toimii myös kirjautumistunnuksena testiin.

Entiteetti *Fieldtest_product:*

Attribuutit sisältävät testattavien tuotteiden perustiedot ja ominaisuudet, kuten nimen ja mallin, hiontakarkeuden sekä hiomapyörön koon. Tähän entiteettiin ei talleteta tietoa asiakassovelluksella, vaan sen sisältämä tieto on staattista ja kyseiseen entiteettiin lisätään tietoja käytännössä vain tuotevalikoiman muuttuessa.

Entiteetti *Fieldtest_productset:*

Sisältää Id-tunnukset Fieldtest_product -entiteettiin sekä testituotteelle että referenssituotteelle. Entiteetillä on suhteet Fieldtest_evaluation- ja Fieldtest_data-entiteetteihin. Käyttäjä arvioi jokaisen tuoteparin erikseen, joten jokaisella arviolla on yksi-yhteen -suhde tuoteparii. Jotta testitunnuksen perusteella voidaan tunnistaa testauksessa käytetyt tuotteet ja tuoteparit, Fieldtest_data -entiteetin yksilöivä tunnus on sidottu Fieldtest_productset -entiteettiin.

Entiteetti *Fieldtest_equipment:*

Sisältää tiedot testauksessa käytetystä laitteistosta ja hiottavasta pinnasta. Testauksessa voidaan vaihtaa laitteistoa eri tuoteparien välillä, joten entiteetin Id-tunnus on sidottu Fieldtest_productset -entiteettiin.

Entiteetti *Live_data:*

Sisältää testin aikana hiomakoneelta kerätyt tiedot, kuten kiihtyvyydet x-, y- ja z-akseleilla, hetkellisen sähkövirran ja jännitteen sekä pyörimisnopeuden. Entiteetin yksilöivä Id-kenttä on sidottu testitunnukseen ja testattavan tuotteen tunnukseen.

Entiteetti *Fieldtest_property:*

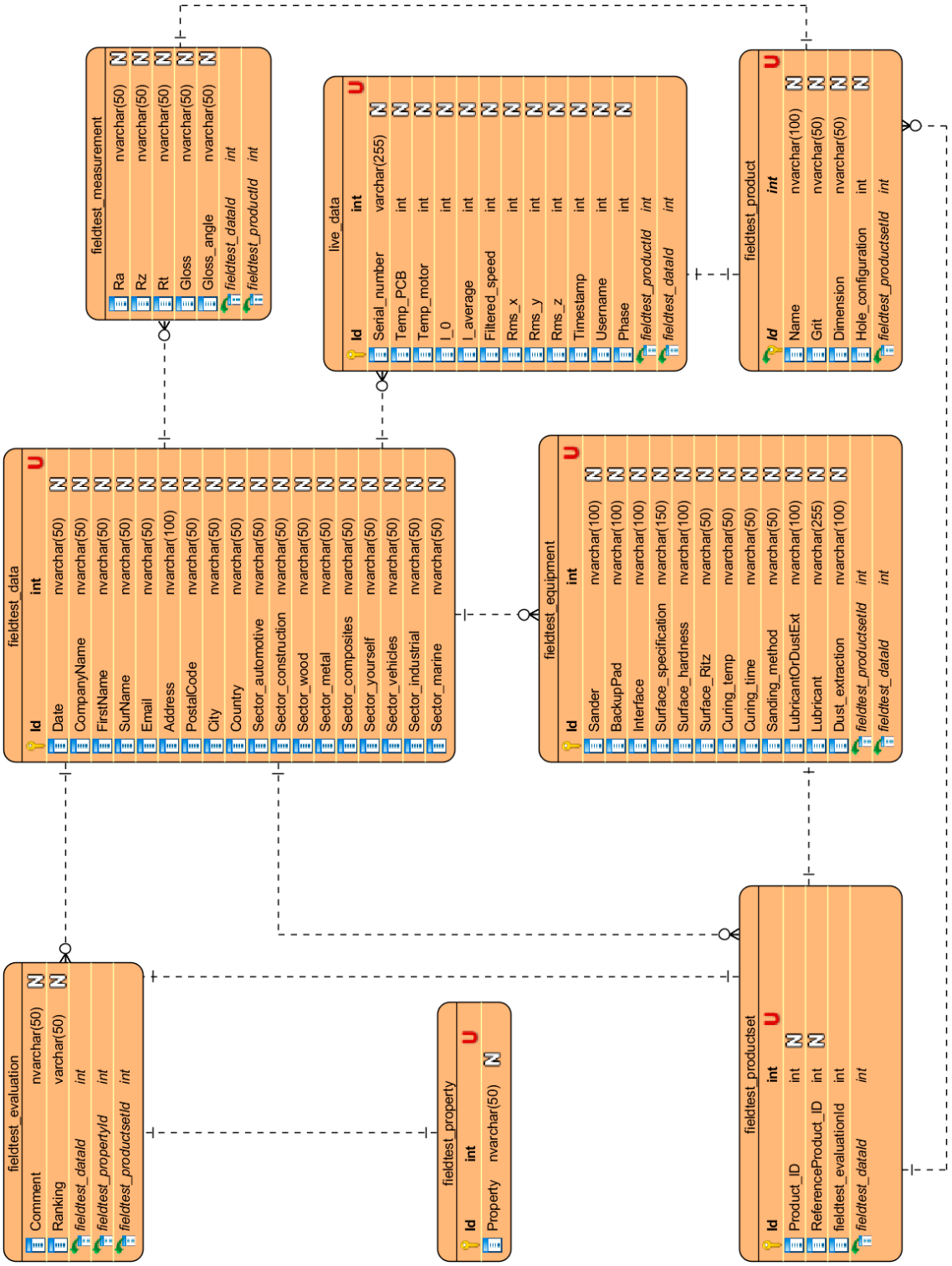
Sisältää arvioitavat ominaisuudet, jotka näytetään mobiilisovelluksen arviointinäkymässä. Tähän entiteettiin ei tallenneta mitään tietoa asiakassovelluksesta.

Entiteetti *Fieldtest_evaluation:*

Sisältää käyttäjän antamat arviot ja kommentit kullekin arvioitavalle ominaisuudelle. Entiteetti on sidottu testitunnukseen sekä arvioitavan ominaisuuden ja tuoteparin yksilöiviin tunnuksiin.

Entiteetti *Fieldtest_measurement:*

Entiteetin attribuutteja ovat hiotusta pinnasta mitattavat arvot, kuten pinnan karheus (Ra, Rz, Rt), kiilto ja mittauskulma. Entiteetillä on yksi-yhteen -suhde Fieldtest_product -entiteettiin sekä monen suhde yhteen Fieldtest_data -entiteetin kanssa.



Kuva 9. Tietomallisuunnitelma ER-kaaviona esitettynä.

5.4 Käyttöliittymäsuunnittelu

Työssä toteutettava sovellus integroidaan osaksi jo olemassa olevaa sovellusta, joten teeman osalta hyödynnetiin sovelluksen nykyistä teemaa. Käyttöliittymän suunnittelussa suurimpana haasteena oli sen luominen mahdollisimman yksinkertaiseksi, jotta näkymien määrä pysyisi kohtuullisena hyvän käyttäjäkokemuksen kannalta. Taustatietojen- sekä palautteenkeruussa erilaisia syötekenttiä tarvittiin paljon, mutta niitä ei saanut olla yksittäisessä näkymässä liikaa, jotta käyttäjä ei turhaannu kenttiä täyttäessä.

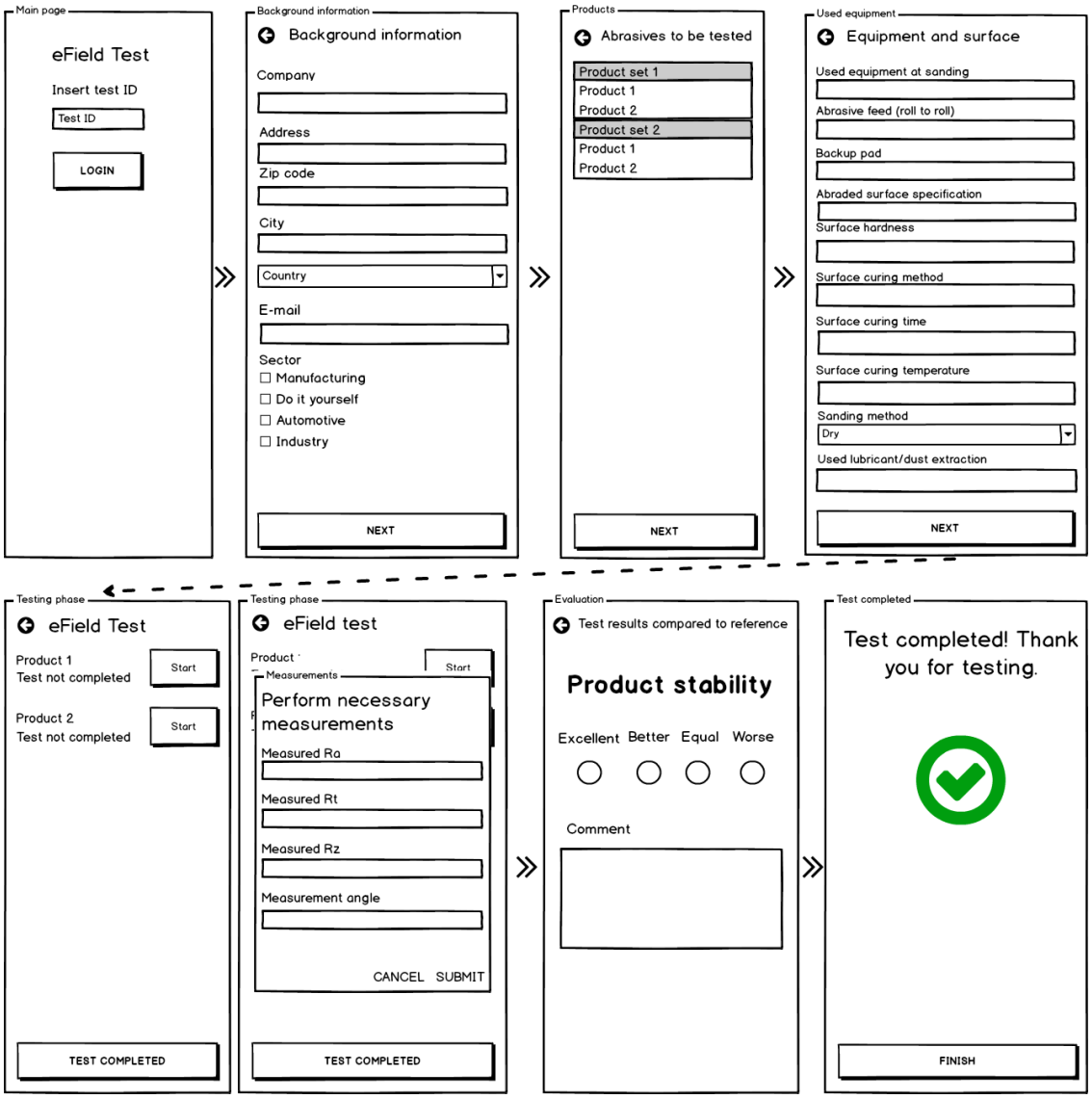
Sovelluksen eri näkymistä tehtiin luonnokset asiakkaan nykyisen testiraporttipohjan perusteella. Siihen verrattuna sovellukseen tuli lopulta asiakkaan toivomuksesta enemmän kerättävää tietoa, joten niiden järjestys piti saada loogiseksi. Näkymien käyttöliittymistä tehtiin aluksi yksinkertaiset luonnokset, jonka jälkeen asiakas antoi niistä palautetta. Tätä menetelmää toistettiin, kunnes käyttöliittymä vaikutti selkeältä ja vastasi asiakkaan toivomuksia. Luonnosten rinnalla käytettiin prototyyppejä sovelluksesta sisältäen vain käyttöliittymän ilman tietojen tallennusta tai muuta logiikkaa. Käyttöliittymä muokkautui nykyiseen muotoonsa pikkuhiljaa projektin edetessä asiakaspalautteen ja erilaisten ratkaisujen kokeilun tuloksena. Sovelluksen toiminnallisuuksien ollessa lähes valmiita todettiin joidenkin ratkaisujen olevan käyttäjäkokemuksen kannalta huonoja, jonka seurauksena käyttöliittymä koki vielä joitakin muutoksia.

Ensimmäisessä liitteessä on kuva ensimmäisestä käyttöliittymäsuunnitelmasta sisältäen sovelluksen näkymät. Kyseisessä versiossa eri näkymiä on yhdeksän. Tässä vaiheessa vaatimuksena oli myös uusien testien luominen sovelluksessa, mutta toiminnallisuus koettiin lopulta tarpeettomaksi ja se poistettiin lopullisesta versiosta. Suunnitelmasta puuttui myös näkymä testattavan tuotteen vaihtamiselle. Toisessa liitteessä on ensimmäisen suunnitelman pohjalta tehty paranneltu versio, jossa on myös luonnos tuotteiden vaihtonäkymästä, ja päivitetty versio tuotteiden lisäysnäkymästä.

Ensimmäisessä suunnitelmassa käyttäjän oli mahdollisuus lisätä testattaviin tuotteisiin hiomapyöröjä sekä alustalloja. Alustallan osalta päädyttiin lisäämään yksi tekstikenttä

testauslaitteisto- ja ympäristö -lomakkeeseen, koska samaa alustallaa käytetään molemmille testituotteille. Tämän lisäksi suunnitelmaan lisättiin enemmän ohjeistusta eri vaiheiden välille. Näiden muutosten takia näkymien määrä lisääntyi kahdella. Tässä versiossa ei kuitenkaan otettu huomioon sitä, että testissä tulee olla mahdollista testata useita eri tuotepareja, sillä kyseinen vaatimus oli tässä vaiheessa jäänyt epäselväksi.

Kolmas suunnitelma, jonka pohjalta käyttöliittymät lopulta toteutettiin, on esitetty kuvassa 10. Suunnitelma poikkesi melko paljon aiemmista suunnitelmista, ja muutosten ansiosta testin kulkua saatiin selkeytettyä huomattavasti.



Kuva 10. Käyttöliittymäsuunnitelma, jonka pohjalta käyttöliittymä lopulta toteutettiin.

Aloitusnäkymästä poistettiin mahdollisuus testin luomiseen, ja näin ollen voitiin myös poistaa näkymät, joissa testattavien tuotteiden lisäys tehtiin. Niiden sijaan näytetään vain tietokannasta haettu lista testattavista tuotteista, jotta käyttäjä voi tarkistaa niiden vastaavan toimitettuja tuotteita. Aiemmassa suunnitelmassa ei myöskään ollut erillistä näkymää varsinaiselle testausvaiheelle, vaan siinä hyödynnettiin aloitusnäkymää lisäämällä siihen painikkeet testin aloittamiselle ja pysäyttämiselle, sekä hiomapyörön valinnalle.

Suunnitelmassa uutena näkymänä tuli testausnäkymä, jossa näytetään jokaisen iteraation tuoteparit, jotka käyttäjän tulee testata ainakin kerran ennen seuraavaan vaiheeseen siirtymistä. Tämän koettiin selkeyttävän testin kulkua ja ohjaavan käyttäjää suorittamaan kaikki vaiheet loppuun. Tämän johdosta myös eri vaiheiden välissä näytettävät ohjeistusnäkymät poistettiin ja korvattiin ohjedialogeilla, jotka näytetään tarvittaessa, mikäli käyttäjä yrittää esimerkiksi siirtyä seuraavaan vaiheeseen ennen edellisen suorittamista loppuun.

Myös tuotteiden arviointinäkymä koki suuren muutoksen: aiemmasta listanäkymästä luovuttiin ja jokainen arviointiperuste päätettiin laittaa omaan näkymäänsä, jolloin käyttäjän on helpompi keskittyä niihin yksi kerrallaan. Arviointinäkymiin saatiin näin ollen myös kommenttikenttä mahtumaan samaan näkymään. Aiemmassa suunnitelmassa arviointinäkymä täyttyi valintanapeista, joten kommenttikentät olisivat vieneet tarpeettoman paljon tilaa näkymästä. Muutoksen ansiosta tilanpuute ei ollut enää ongelma ja kommenttikentälle löytyi sopiva paikka näkymästä. Arviointinäkymä päätettiin toteuttaa siten, että arviointiperusteiden välillä voi siirtyä näyttöä pyyhkäisemällä.

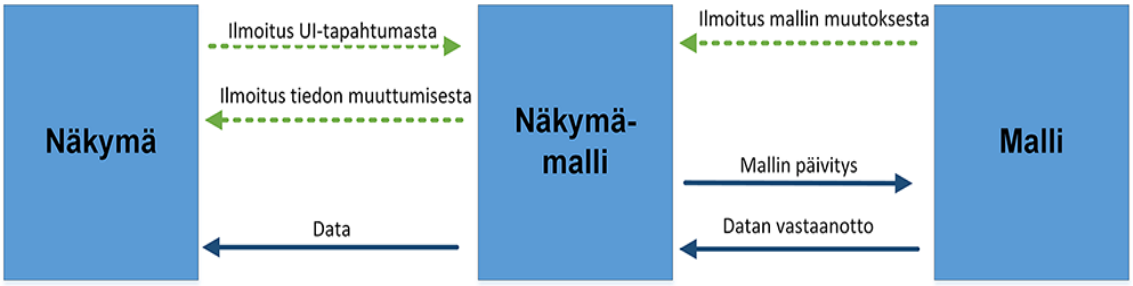
6 SOVELLUKSEN TOTEUTUS

Toteutusvaihe koostuu eri sovelluskomponenttien ohjelmoinnista ja integroinnista muihin komponentteihin. Tässä luvussa kerrotaan sovelluksen toteutuksessa käytetyistä menetelmistä ja työkaluista sekä mobiili- ja palvelinsovellusten toiminnasta.

6.1 Suunnittelumalli ja kehitysmenetelmä

Sovelluksen toteutus perustuu MVVM-suunnittelumalliin, joka on lyhenne englannin kielisistä sanoista Model-View-ViewModel. Microsoft kehitti suunnittelumallin MVC-mallin (*Model-View-Controller*) pohjalta WPF-sovelluksia (*Windows Presentation Foundation*) varten. Suunnittelumalli soveltuu hyvin myös alustariippumattomien sovellusten kehittämiseen. (Peppers, Taskos, Bilgin 2016.)

Perinteisesti käyttöliittymiä on luotu lisäämällä logiikka käyttöliittymäkomponenttien yhteyteen. Tällaisessa lähestymistavassa näkymäluokan koko kasvaa sekä käyttöliittymän, logiikan ja tietomallien välille syntyy vahvat riippuvuussuhteet. Tästä johtuen yksittäisten näkymien ylläpito muuttuu hankalaksi etenkin, jos sovellusta kehittää useampi henkilö samanaikaisesti. Nimensä mukaisesti MVVM-suunnittelumallin periaatteena on jakaa sovellus kolmeen osaan: malliin (engl. *model*), näkymään (engl. *view*) sekä näkymämalliin (engl. *viewmodel*). Sovelluksen osittamisen tarkoituksena on helpottaa sovelluksen ylläpitoa ja testausta. (Shukkur 2013.) Kuvassa 11 on esitetty MVVM-mallin osat ja niiden tehtävät.



Kuva 11. MVVM-mallin tehtävät (mukaillen Ivanov 2015).

Näkymässä näytetään käytännössä vain käyttöliittymäkomponentit ja näkymämallilta saatava data. Käyttöliittymäkomponentit voivat lähettää näkymämallille tapahtumia (engl. *event*) joiden perusteella tietoja käsitellään. Malliluokka puolestaan voi lähettää tapahtumia näkymämallille esimerkiksi mallin sisältämän tiedon muuttuessa, jolloin näkymämallissa voidaan pyytää näkymää päivittämään komponentti, jossa tietoa käytetään. Näkymämalli toimii siis näkymän ja mallin välissä eräänlaisena rajapintana. Näkymämallissa voidaan hyödyntää datan sidontaa (engl. *binding*), jonka avulla voidaan luoda riippuvuuksia ominaisuuksien ja käyttöliittymäkomponenttien välille siten, että määrätty toimenpide suoritetaan aina, kun sidottu tieto muuttuu. Sidonta voi olla yksi- tai kaksisuuntainen tarpeesta riippuen. (Shukkur 2013.)

Kehitysmenetelmä mukaili ketteriä menetelmiä, joista lähimpänä voidaan pitää *Extreme Programming* -menetelmää. Extreme Programming -menetelmässä painotetaan mukautuvuutta ennustettavuuden sijaan. (Lehtonen, Tuomivaara, Rantala, Känsälä, Mäkilä, Jokela, Könnölä, Kaisti, Suomi, Isomäki & Ylitolva 2014: 6-7.) Tässä työssä kyseisen menetelmän arvoista toteutui pääsääntöisesti jatkuva integrointi, asiakastestit ja pienet julkaisut. Sovelluksen yksityiskohtaiset vaatimukset tarkentuivat projektin edetessä ja sovellus muokkautui nykyiseen muotoonsa asiakastestauksen ja jatkuvan palautteen perusteella.

6.2 Mobiilisovelluksen rakenne

Tässä luvussa esitetään mobiilisovelluksen tekninen toteutus pääpiirteittäin. Sovelluksen kehityksessä käytettiin Microsoft Visual Studio 2015 -kehitysympäristöä Xamarin 7.0 liitännäisellä varustettuna. Natiivista Android-ohjelmoinnista poiketen ohjelmointikielenä käytettiin Javan sijaan C#-ohjelmointikieltä. Sovellus on kehitetty toimimaan Android 5.0:lla (*Lollipop*) ja sitä uudemmilla versioilla. Kehityksessä käytettiin Androidin kehitystyökalukokoelman (engl. *Software Development Kit,* SDK) versiota 21.

6.2.1 Kolmansien osapuolten liitännäiset

Sovelluksessa käytetään MVVM-suunnittelumallin toteuttamiseksi avoimen lähdekoodin TinyIoC-toteutusta sekä Galasoftin MvvmLight -kirjastoa. TinyIoC:n avulla luodaan tietosäiliö (engl. *container*), jossa rekisteröidään esimerkiksi näkymämallit ja erilaisten palveluiden rajapinnat niitä vastaaviin toteutuksiin. Rekisteröityjen rajapintojen avulla luokkien ilmentymiä voidaan kierrättää sovelluksessa luomatta jokaisessa luokassa uutta. Tätä tekniikkaa kutsutaan riippuvuusinjektioksi (engl. *Dependency Injection*). Riippuvuusinjektio voidaan toteuttaa usealla eri tavalla, joista yleisin on injektoida tarvittavat riippuvuudet luokkien konstruktoreissa. Riippuvuusinjektion ansiosta luokkien ei tarvitse itse hallita riippuvuuksia, vaan niitä hallitaan yhdessä erillisessä tietosäiliössä. (Britch 2017.)

Luokkien rekisteröinti voidaan toteuttaa TinyIoC:n avulla seuraavasti:

```
private TinyIoCContainer _container;

_container = TinyIoCContainer.Current;

_container.Register<EquipmentViewModel>().AsSingleton();

_container.Register<INetworkHandler, NetworkHandler>();
```

Jolloin *NetworkHandler*-luokan instanssi voidaan injektoida *INetworkHandler*-rajapinnan avulla *EquipmentViewModel*-luokkaan sen konstruktorissa seuraavasti:

```
private readonly INetworkHandler _networkHandler;

public EquipmentViewModel(INetworkHandler networkHandler)
{
    if (null == networkHandler)
        throw new ArgumentNullException(
            nameof(networkHandler));

    _networkHandler = networkHandler;
}
```

EquipmentViewModel-luokan instanssi voidaan puolestaan hakea esimerkiksi näkymäluokassa seuraavasti:

```
private EquipmentViewModel _equipmentVM;

EquipmentViewModel EquipmentVM =>
_equipmentVM ?? (_equipmentVM = ViewModelLocator.Locator
   .EquipmentViewModel);
```

MvvmLight-kirjaston avulla MVVM-arkkitehtuuria saadaan täydennettyä ominaisuuksien sidonnalla. Sidonnan avulla tietoa ei tarvitse manuaalisesti päivittää joka kerta kun tieto muuttuu, vaan sidosten välinen tieto voidaan päivittää automaattisesti. Sidoksia voidaan luoda seuraavan esimerkin mukaisesti:

```
private Binding _sanderBinding;

_sanderBinding = this.SetBinding(() =>
    EquipmentViewModel.Sander,
    () => _editSander.Text,
    BindingMode.TwoWay);
```

Ylläolevassa esimerkissä luodaan yksityinen muuttuja *_sanderBinding* ja asetetaan sille sidos *SetBinding*-metodin avulla antamalla sidottavat ominaisuudet parametreina. Tässä tapauksessa luodaan kaksisuuntainen sidos EquipmentViewModel-näkymämalliluokan *Sander*-ominaisuuden ja näkymäluokan *_editSander*-tekstikentän välille. Näin ollen jommankumman ominaisuuden sisältämän tiedon muuttuessa tieto päivitetään vastaamaan uusinta muutosta.

Edellä mainittujen kolmansien osapuolten liitännäisten lisäksi sovelluksessa hyödynnetään NewtonSoftin JSON.Net -sovelluskehystä, jota käytetään tietojen serialisointiin. Serialisoinnilla tarkoitetaan tietomallin sisältämän tiedon muuttamista merkkijonoksi. Serialisoinnin käänteisoperaatio on deserialisointi, jossa puolestaan merkkijono muutetaan halutun tietomallin mukaiseksi. JSON.Net -sovelluskehyksen avulla oliot voidaan serialisoida JSON-formaattiin. (Reynolds 2014: 59.)

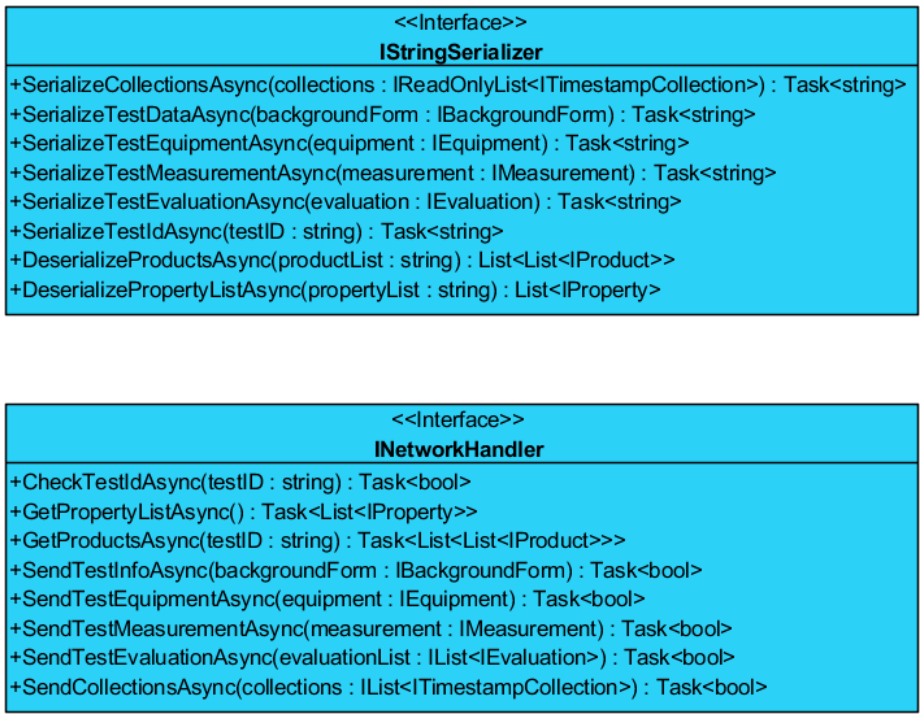
6.2.2 Luokkarakenne

Liitteessä 3 on esitetty mobiilisovelluksen luokkarakenne sisältäen eField Test -moduulin näkymien ja näkymämallien suhteet. Kaaviosta nähdään, että näkymäluokat ovat pääasiassa aktiviteetteja. Niitä käytetään kaikissa muissa näkymissä paitsi *FieldTestFragment*-

näkymässä, jonka kautta testaus aloitetaan. Kyseisessä näkymässä käytettiin aktiviteetin sijaan fragmentia, jotta se voitiin integroida jo olemassa olevaan navigointiin, joka on toteutettu Androidin *ViewPager*-komponenttia hyödyntäen.

Sovelluksen aktiviteetit toteuttavat Androidin AppCompat v7 -tukikirjastoon kuuluvan AppCompatActivity-kantaluokan OnCreate-metodin, joka ylikirjoitetaan *override*-avainsanalla. Tässä sovelluksessa OnCreate-metodissa kutsutaan kantaluokan OnCreate-metodin lisäksi useimmiten yksityisiä *SetupElements*-, *SetupToolbar*- ja *SetupBindings* -metodeja. SetupElements- ja SetupToolbar -metodeissa asetetaan käyttöliittymäkomponenteille esimerkiksi tapahtumakäsittelijät (engl. *event handler*). SetupBindings -metodeissa luodaan sidokset näkymämallin ominaisuuksien ja näkymän käyttöliittymäkomponenttien välille.

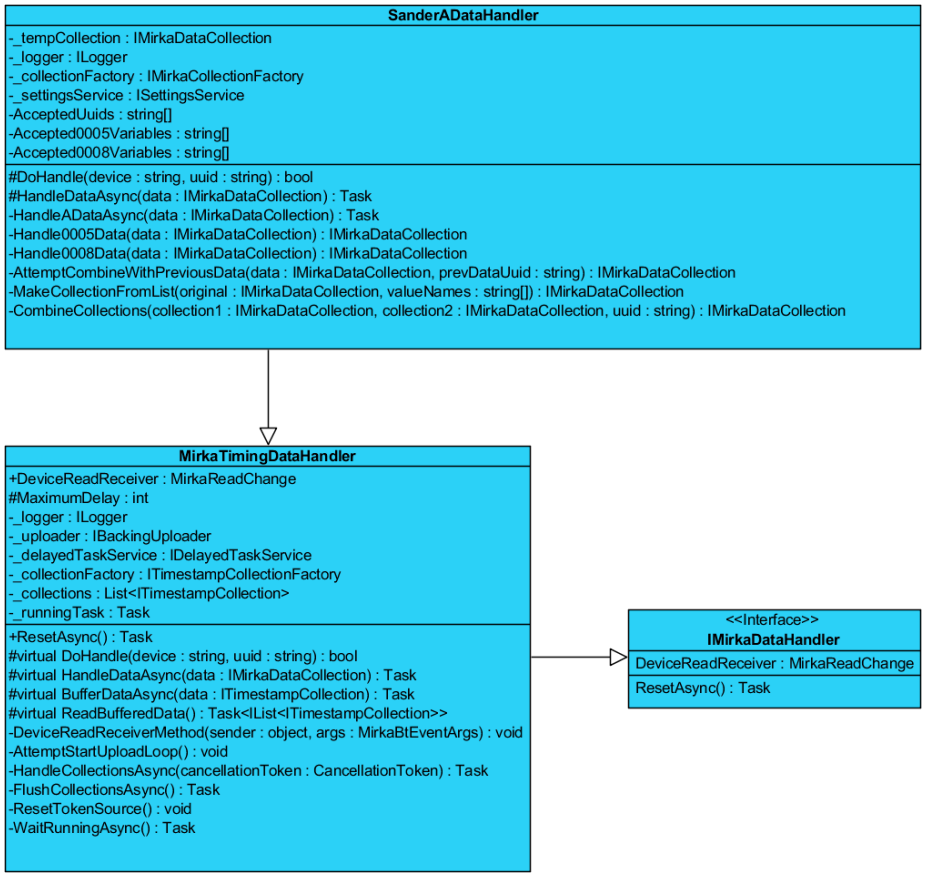
Liitteen 3 luokkakaaviossa esitettyjen luokkien lisäksi sovellus hyödyntää myös joitakin aiemmin luotuja luokkia, joita on täydennetty sovelluksen tarpeiden mukaan. Tällaisia luokkia ovat esimerkiksi NetworkHandler-, *StringSerializer*- ja *SanderADataHandler* -luokat, joista kahden ensimmäiseksi mainitun rajapintaluokat on esitetty kuvassa 12.



Kuva 12. NetworkHandler- ja StringSerializer-luokkien rajapinnat.

NetworkHandler-luokassa hoidetaan tietojen lähetys ja vastaanotto sovelluksen ja REST-rajapinnan välillä, ja StringSerializer-luokassa siirrettävä tieto serialisoidaan palvelimen vaatimaan muotoon. NetworkHandler-luokan metodit ovat asynkronisia, jolloin ne suoritetaan omissa säikeissään ja näin ollen sovelluksen käyttöliittymä pysyy responsiivisena myös tietojen lähetyksen aikana. (Nilanchala Panigrahy 2015: 98–99.) Näitä metodeja kutsutaan *await*-avainsanan kanssa niissä näkymämalleissa, joissa dataa halutaan siirtää palvelimen ja sovelluksen välillä.

Kuvassa 13 on esitetty hiomakoneen kanssa kommunikoinnista vastaava SanderADataHandler-luokka ja sen riippuvuudet luokkakaaviona. SanderADataHandler-luokka perii *MirkaTimingDataHandler*-luokan, joka toteuttaa *IMirkaDataHandler*-rajapinnan. SanderADataHandler käyttää kantaluokan metodeja tietojen vastaanottamiseen hiomakoneelta. Kerätystä datasta muodostetaan *MirkaDataCollection*-olioita, jotka serialisoidaan myöhemmässä vaiheessa JSON-muotoon ennen palvelimelle lähetystä.

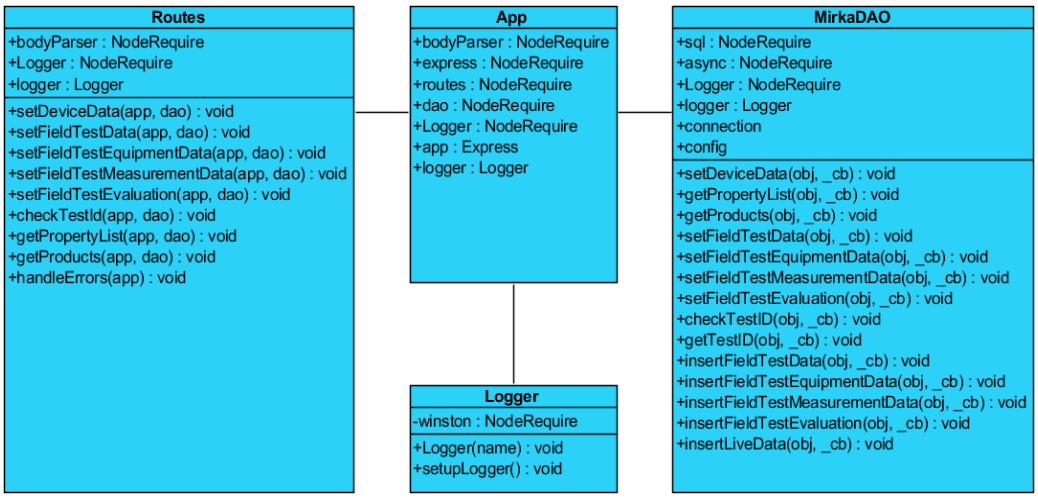


Kuva 13. Hiomakoneelta haettavan datan käsittelystä vastaavat luokat.

6.3 Palvelinsovelluksen rakenne

Palvelinsovelluksena (engl. *backend*) käytetään jo olemassa olevaa sovellusta lisäämällä siihen eField Test -moduulin tarvitsemat rajapintametodit. Palvelinsovellus on REST-arkkitehtuurimallin mukainen ja se on toteutettu Node.js -sovellusalustalla hyödyntäen Express.js -sovelluskehystä. Palvelinsovellus kommunikoi asiakassovelluksen ja tietokannan välillä, ja sen kautta voidaan siirtää tietoa molempiin suuntiin HTTP-protokollan (*Hypertext Transfer Protocol*) mukaisia POST- ja GET-metodeja käyttämällä. Mikäli lähetetyn viestin sisältö on kutsuttavan rajapintafunktion vaatimassa muodossa, palautetaan viestin lähettäjälle vastaus, joka sisältää tässä tapauksessa aina tietokannasta haettua tietoa. Mikäli viestin sisältö ei ole rajapintafunktion vaatimassa muodossa, palautetaan vastaavasti virhekoodi, jonka mukaan asiakassovelluksessa ilmoitetaan käyttäjälle toiminnon epäonnistumisesta virheviestein.

Kuvassa 14 on esitetty palvelinsovelluksen luokkarakenne tässä työssä merkityksellisten funktioiden ja attribuuttien osalta. *App*-luokassa sisällytetään sovelluksen tarvitsemat moduulit Noden *require*-funktion avulla, ja tämän jälkeen käynnistetään sovellus. Sovelluksen käynnistys tapahtuu komentoriviltä komennolla `node App.js`. *Logger*-luokassa luodaan instanssi lokimoduulista, jonka avulla voidaan tallentaa tietoa sovelluksen suorituksesta, ja esimerkiksi mahdollisista virhetilanteista myöhempää tarkastelua varten. Lokien luomisessa käytetään winston-kirjastoa.



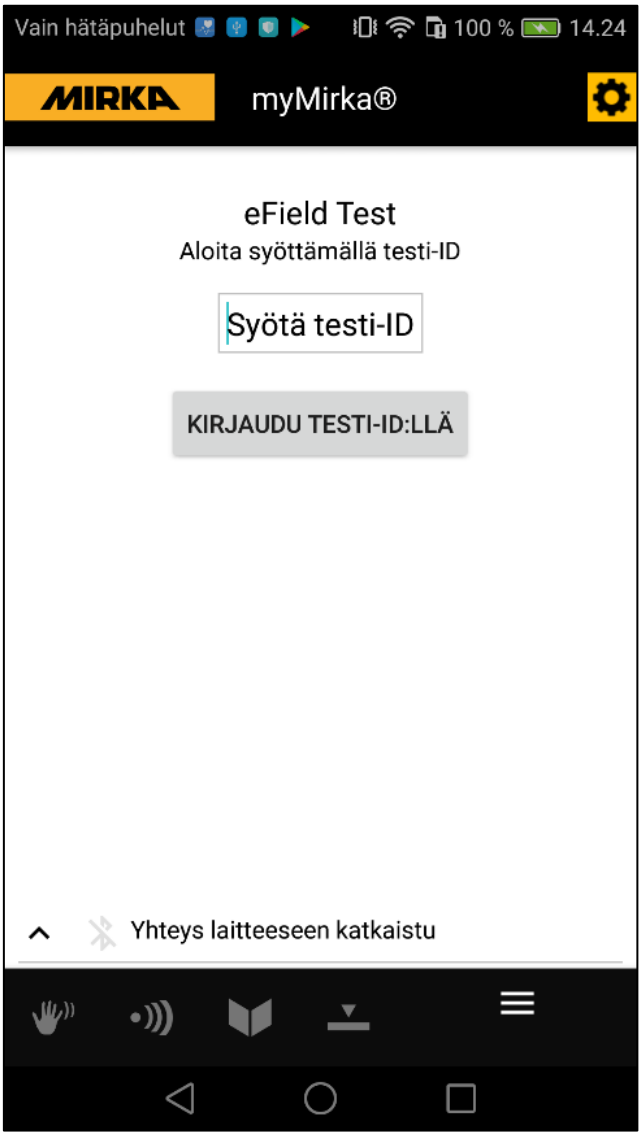
Kuva 14. Palvelinsovelluksen luokkarakenne.

Routes-luokassa määritetään rajapinnan URL-osoitteet, ja reititetään ne *MirkaDAO*-luokan funktioihin, joissa tietokantakutsut ja datan käsittely hoidetaan. Routes-luokan funktioissa myös tarkistetaan vastaanotettujen HTTP-pyyntöjen oikeellisuus ja jäsennetään ne Noden body-parser -kirjastoa hyödyntäen helpommin käsiteltävään muotoon. Tässä vaiheessa palautetaan tarvittaessa virheviesti, jolloin vältytään virheellisiltä tietokantakutsuilta.

MirkaDAO-luokassa *set*-alkuisissa funktioissa parsitaan taulukkomuotoinen sisältö yksittäisiksi riveiksi ja kutsutaan vastaavaa *insert*-alkuista funktiota. Kyseisessä funktiossa luodaan tietokantayhteys ja kutsutaan tietokantapalvelimen tallennettua proseduuria, jossa suoritetaan varsinainen SQL-kysely. SQL-kyselyn voisi vaihtoehtoisesti lähettää suoraan palvelinsovelluksesta, mutta tallennettujen proseduurien etuna on mahdollisuus parametrisoituihin kyselyihin, jotka estävät oikein käytettynä tietokannan altistumisen SQL-injektiohyökkäyksille (OWASP 2017). *Get*-alkuisissa funktioissa kutsutaan myös tallennettua proseduuria, mutta set-alkuisista funktioista poiketen niissä palautetaan takaisinkutsuna (engl. *callback*) SQL-kyselystä saatu sisältö.

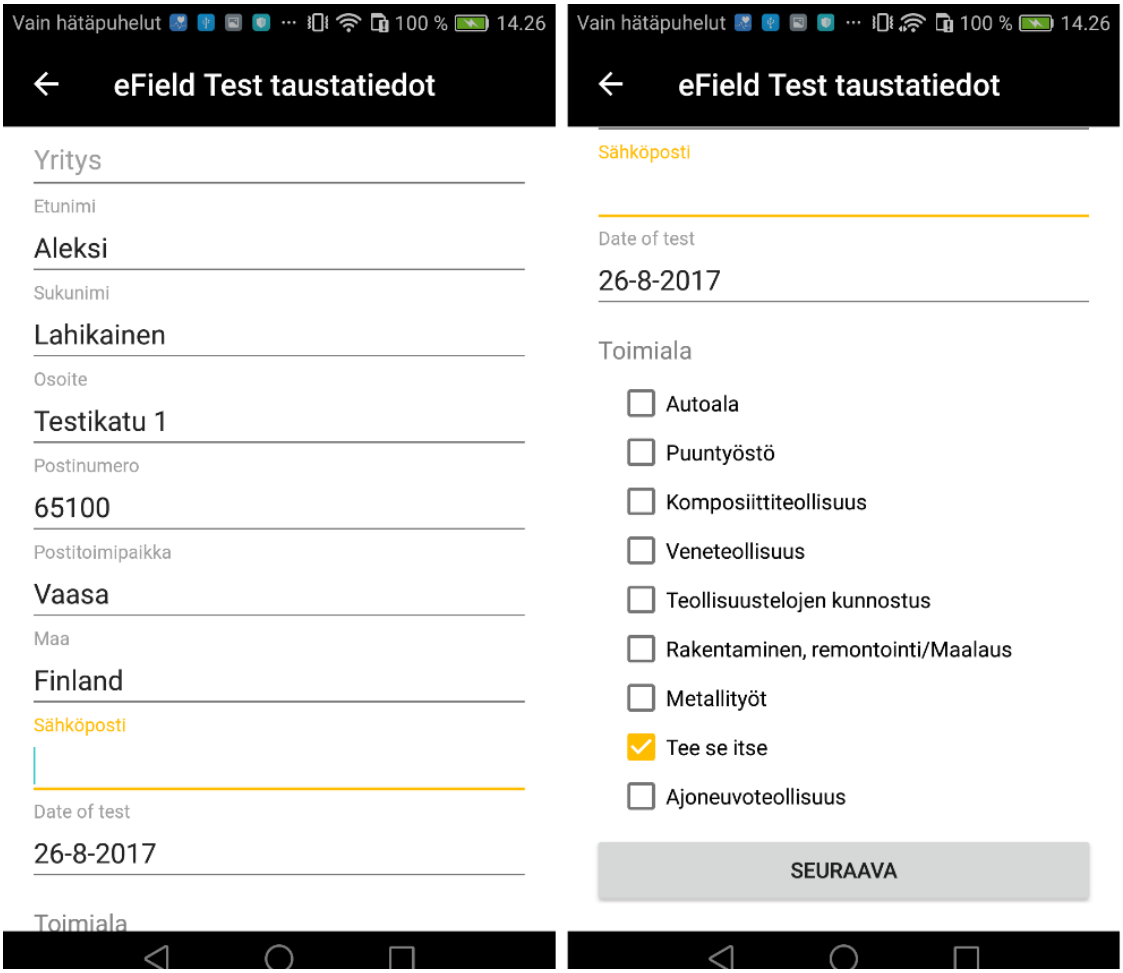
6.4 Mobiilisovelluksen toiminta ja käyttö

Tuotteiden testaus aloitetaan syöttämällä yksilöllinen testitunnus tekstikenttään. Kun käyttäjä painaa kirjautumispainiketta, sovellus lähettää palvelimelle kutsun tehdä tietokantahaku syötetyllä testitunnuksella. Käyttäjä ohjataan seuraavaan näkymään, mikäli syötetty testitunnus löytyy tietokannasta. Kuvassa 15 on kuvakaappaus eField Test -aloitusnäkymästä. Tämän näkymän logiikka hoidetaan FieldTestFragment-luokassa.



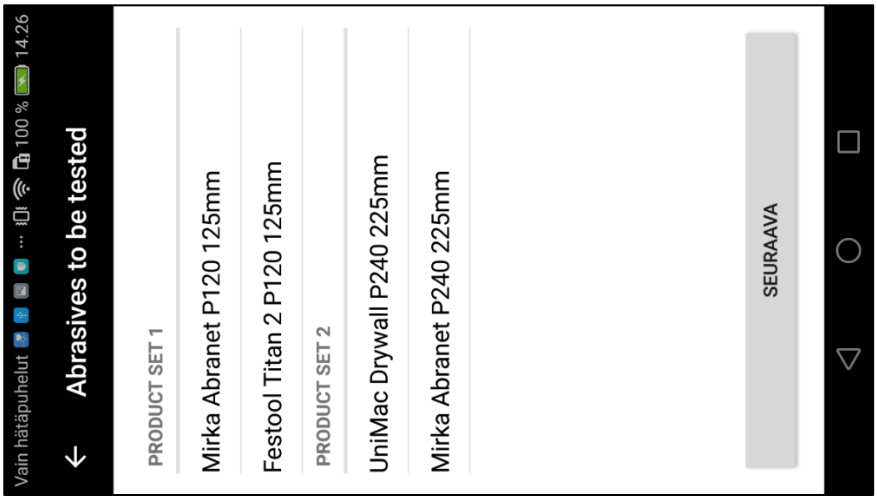
Kuva 15. eField Test -aloitusnäkymä.

Seuraavassa näkymässä käyttäjältä kysytään taustatiedot kuvan 16 mukaisella lomakkeella. Näkymän ohjaus hoidetaan *TestInfoActivity*-luokassa ja syötetyt tiedot on sidottu *BackgroundInfoViewModel*-näkymämalliluokkaan, jotta tiedot voidaan palauttaa kenttiin, mikäli käyttäjä palaa takaisin tähän näkymään.



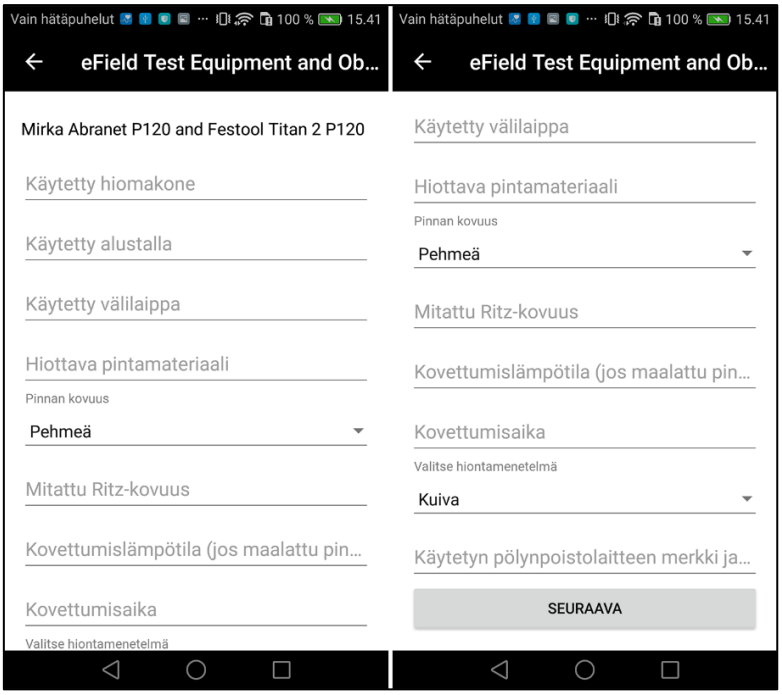
Kuva 16. Taustatietojen keruunäkymä.

Taustatietojen keruun jälkeen käyttäjälle näytetään testattavat tuotteet tuotepareina. Tuoteparien molemmat tuotteet testataan samalla laitteistolla ja samoissa olosuhteissa. Laitteisto ja olosuhteet voivat kuitenkin vaihdella tuoteparien välillä. Sovelluksen ensimmäisissä versioissa oli mahdollista lisätä testattavat tuoteparit itse, mutta tämän toiminnon osalta vaatimukset muuttuivat ja nykyään tuotteet haetaan tietokannasta testitunnuksen perusteella. Tuotteet näytetään käyttäjälle kuvan 17 mukaisessa näkymässä. Tätä näkymää ohjataan *TestProductsActivity*-luokassa, ja tuoteparien tilaa ja tietoja pidetään yllä *ProductViewModel*-näkymämalliluokassa.



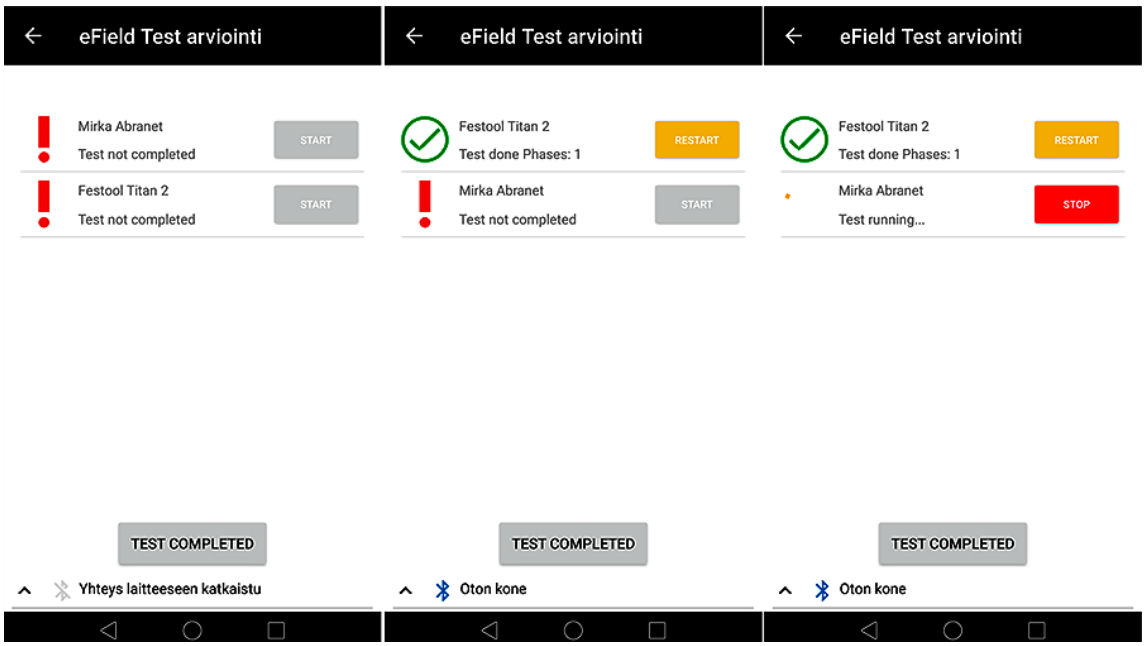
Kuva 17. Testattavat tuotteet -näkymä.

Seuraavassa vaiheessa testauksessa käytettävän laitteiston ja testausympäristön tiedot täytetään kuvan 18 mukaisessa näkymässä. Tämän näkymän tiedot täytetään jokaiselle tuoteparille ennen testauksen aloittamista, jolloin eri tuotepareja voidaan testata erilaisissa olosuhteissa ja erilaisella laitteistolla testattavien tuotteiden ominaisuuksista riippuen. Näkymä luodaan *TestEquipmentActivity*-luokassa ja tuotetietojen nouto hoidetaan EquipmentViewModel-näkymämalliluokassa.



Kuva 18. Käytettävän laitteiston ja testausympäristön tiedot -näkymä.

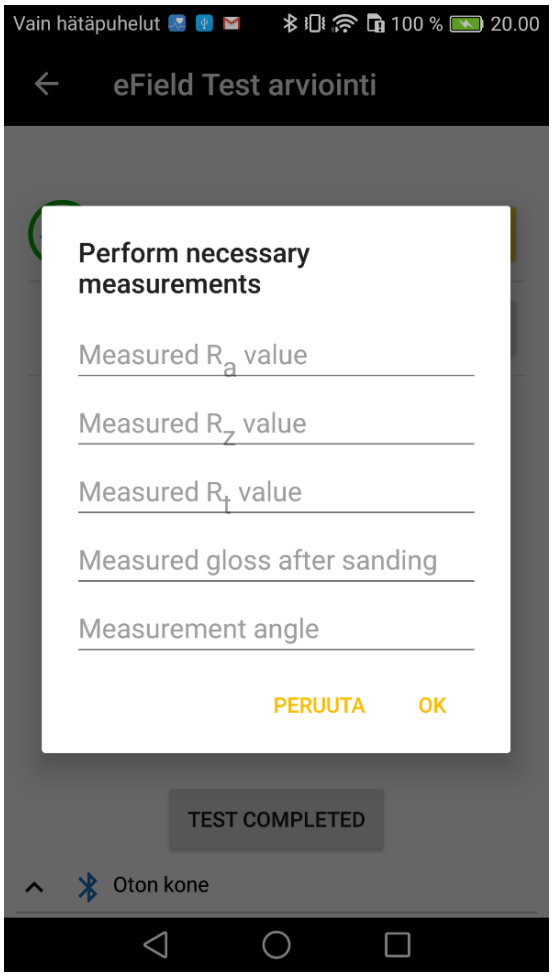
Tietojen täyttämisen jälkeen siirrytään näkymään, jossa varsinainen testaus tapahtuu. Näkymässä näytetään testattava tuotepari kuvan 19 mukaisesti. Testaus aloitetaan painamalla testattavan tuotteen kohdalla *Start*-painiketta, jolloin sovellus alkaa kerätä tietoa hiomakoneelta. Mikäli hiomakonetta ei ole paritettu mobiililaitteen kanssa, sovellus ilmoittaa siitä käyttäjälle ja paritus voidaan tehdä näkymän alareunassa olevan valitsimen kautta. Näkymää ohjataan *TestStartedActivity*-luokassa ja kerätty data lähetetään palvelimelle testauksen aikana *BackingUploader*- ja NetworkHandler-luokkien avulla.



Kuva 19. Tuotteiden testausnäkymä.

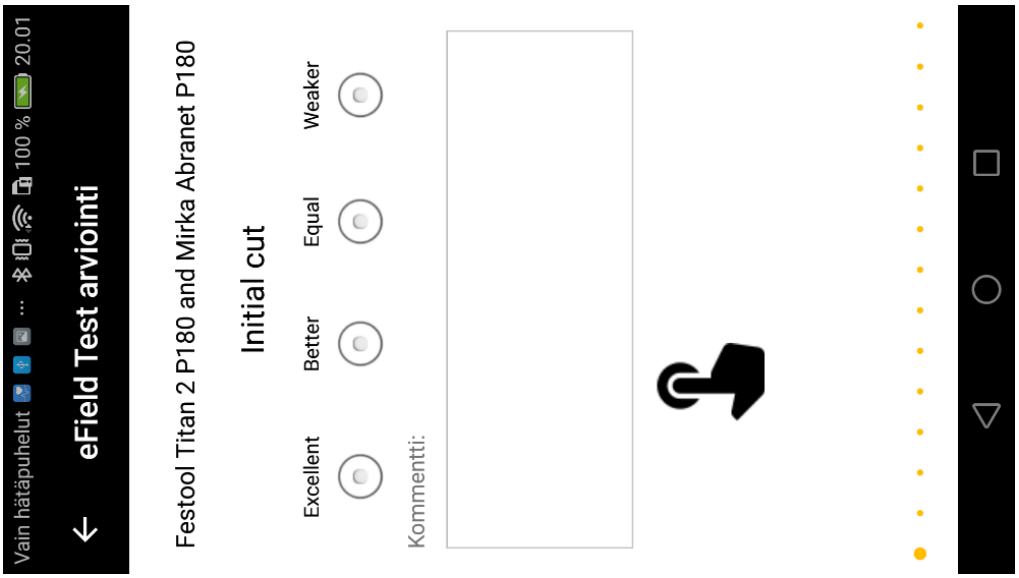
Testauksen voi keskeyttää milloin tahansa painamalla *Stop*-painiketta testattavan tuotteen kohdalla. Testaus voidaan suorittaa tuoteparin molemmille tuotteille niin monta kertaa kuin testaaja kokee tarpeelliseksi. Yksittäisen tuotteen voi testata uudelleen painamalla *Restart*-painiketta. Tietokantaan tallennettavassa datassa on mukana testauksen vaihe, joka myös näytetään testausnäkymässä jokaiselle tuotteelle. Tuoteparin testaus lopetetaan painamalla *Test completed* -painiketta, kun molempia tuotteita on testattu vähintään kerran.

Testausvaiheen jälkeen avautuu kuvan 20 mukainen ikkuna, jossa käyttäjältä kysytään hiotusta pinnasta mitattavia arvoja. Näiden tietojen syöttäminen on vapaaehtoista, ja ikkunan voi halutessaan sulkea ja jatkaa seuraavaan vaiheeseen.



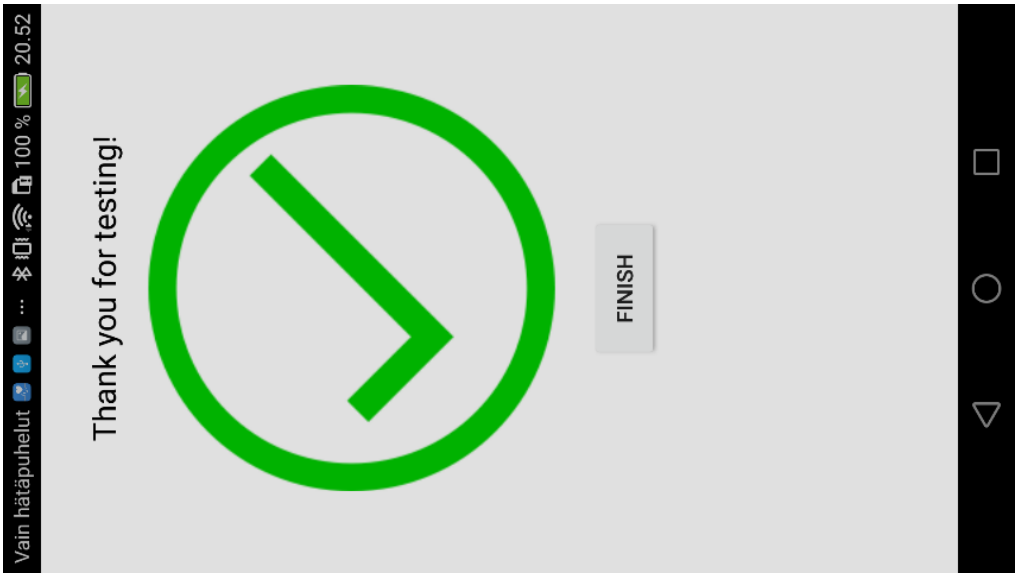
Kuva 20. Mittausten kyselynäkymä.

Seuraavassa vaiheessa käyttäjä arvioi testatun tuotteen vertaamalla sitä referenssituotteeseen. Tuotteen eri ominaisuudet arvioidaan neliportaisella Likert-asteikolla, ja jokaisesta ominaisuudesta voi halutessaan antaa sanallisen kommentin. Kysymysten välillä siirtyminen tapahtuu pyyhkäisemällä ruutua vasemmalle tai oikealle. Tämä ilmaistaan käyttäjälle ensimmäisessä kysymysnäkymässä näyttämällä pyyhkäisyanimaatio kommenttikentän alapuolella kuten kuvassa 21. Viimeisellä kysymyssivulla näytetään painike, jota painamalla tiedot lähetetään palvelimelle.



Kuva 21. Tuoteparin arviointinäkymä.

Mikäli testiin on rekisteröity useampi tuotepari, käyttäjä siirretään tässä vaiheessa uudelleen laitteiston ja testausympäristön kyselynäkymään, jonka jälkeen testaus ja arviointi suoritetaan uudelle tuoteparille. Kun kaikki tuoteparit on testattu ja arvioitu, siirretään käyttäjä kuvan 22 mukaiseen näkymään, jossa kiitetään testauksesta ja ilmoitetaan sen päättyneen. *Finish*-painiketta painamalla siirrytään takaisin eField Test -aloitusnäkymään.



Kuva 22. Testauksen päättymisestä ilmoittava näkymä.

7 SOVELLUKSEN TESTAUS

Sovelluksen testauksella tarkoitetaan suunnitelmallista virheiden etsimistä, jolla pyritään varmistamaan, että sovellus täyttää sille asetetut odotukset. Testaus voidaan jakaa eri tasoihin testattavan osan laajuuden perusteella. Testaustasoja ovat moduuli- eli yksikkötestaus, integraatiotestaus ja järjestelmätestaus. Yksikkötestaus kattaa yksittäiset moduulit, integraatiotestaus useista yksittäisistä moduuleista koostuvat osajärjestelmät, ja järjestelmätestaus koko järjestelmän. (Haikala & Märijärvi 2000: 270–273.) Usein ketterän ohjelmistokehityksen tapoihin kuuluu yksikkötestaus osana kehitysprosessia. Yksikkötestauksella voidaan varmistaa sovelluksen osien oikea toiminta esimerkiksi muutosten yhteydessä.

Tässä luvussa esitetään kehitetyn sovelluksen testaus. Testaus oli tasoltaan järjestelmätestausta, ja sitä tehtiin myös asiakkaan kanssa oikeassa käyttöympäristössä. Asiakas havaitsi sovelluksessa joitakin puutteita, jotka on sittemmin korjattu. Testitapaukset on jaettu aktiviteettien mukaan viiteen eri luokkaan, jotka on esitetty omissa taulukoissaan. Taulukot sisältävät solut testitapauksen yksilöivälle id-tunnukselle, testitapauksen kuvaukselle sekä odotetulle tulokselle. Taulukossa 2 on testin aloittamiseen ja kirjautumiseen liittyvät testitapaukset.

Taulukko 2. eField Testin aloitusnäkymän ja kirjautumisen testitapaukset.

Id	**Tapaus**	**Odotettu tulos**
A1	Painetaan testausikonia sovelluksen alavalikossa.	eField Test -aloitusnäkymä aukeaa.
A2	Syötetään testitunnus sille osoitettuun tekstikenttään ja painetaan kirjautumispainiketta.	Taustatietojen keruunäkymä aukeaa.
A3	Syötetään virheellinen testitunnus.	Virheilmoitus näytetään ruudun alareunassa.

Taulukossa 3 on esitetty taustatietojen keruuseen liittyvät testitapaukset. Nämä testitapaukset kattavat tietojenkeruunäkymien välillä navigoinnin, tietojen tallentamisen sovelluksen sisällä sekä virheilmoitusten näyttämisen.

Taulukko 3. Taustatietojen keruunäkymän testitapaukset.

Id	**Tapaus**	**Odotettu tulos**
B1.1	Jätetään jokin pakolliseksi merkitty tekstikenttä tyhjäksi ja painetaan Next-painiketta.	Virheilmoitus näytetään ruudun alareunassa ja puutteellisten tekstikenttien väri vaihtuu punaiseksi.
B1.2	Täytetään kaikki pakolliseksi merkityt tekstikentät ja painetaan Next-painiketta.	Testattavat tuotteet -näkymä aukeaa.
B2	Painetaan yläreunan työkalupalkissa olevaa nuolta.	Testauksen taustatiedot -näkymä aukeaa. Aiemmin täytetyt tiedot näkyvät lomakkeessa.
B3.1	Toistetaan tapaus B1.1, jonka jälkeen painetaan uudelleen Next-painiketta.	Testauksessa käytettävä laitteisto -näkymä aukeaa.
B3.2	Painetaan yläreunan työkalupalkissa olevaa nuolta.	Testattavat tuotteet -näkymä aukeaa.
B4.1	Toistetaan tapaus B1.1	Virheilmoitus näytetään ruudun alareunassa ja puutteellisten tekstikenttien väri vaihtuu punaiseksi.
B4.2	Toistetaan tapaus B1.2	Testausnäkymä aukeaa.

Taulukossa 4 on esitetty tuotteiden testausvaiheeseen liittyvät testitapaukset. Tapauksilla varmistetaan, ettei tuotetestausta voi aloittaa, mikäli yhteys hiomakoneeseen puuttuu, ja että testauksen tilat vaihtuvat painiketta painamalla oikein.

Taulukko 4. Testausnäkymän testitapaukset.

Id	**Tapaus**	**Odotettu tulos**
C1	Yhdistetään hiomakoneeseen sovelluksen alareunasta avautuvan valikon kautta, ja painetaan Start-painiketta halutun tuotteen kohdalla.	Tuotteen nimen vasemmalla puolella oleva punainen huutomerkki muuttuu pyöriväksi ympyräksi.
C2	Painetaan Stop-painiketta.	Tuotteen nimen vasemmalla puolella oleva pyörivä ympyrä muuttuu vihreäksi oikein-merkiksi.
C3	Painetaan Restart-painiketta.	Tuotteen nimen vasemmalla puolella oleva oikein-merkki muuttuu pyöriväksi ympyräksi.
C4	Painetaan Stop-painiketta.	Tuotteen nimen vasemmalla puolella oleva pyörivä ympyrä muuttuu vihreäksi oikein-merkiksi, ja Phase-tekstin jälkeen tuleva luku kasvaa yhdellä.
C5	Toistetaan toiselle tuotteelle tapaukset C1-C4.	
C6	Painetaan käyttöjärjestelmän Kumoa-painiketta.	eField Test -aloitusnäkymä aukeaa.
C7	Painetaan Continue test -painiketta.	Testausnäkymä aukeaa.
C8	Painetaan Test completed -painiketta.	Mittaustulokset -ikkuna aukeaa.

Taulukossa 5 on hiottavan pinnan mittaustulosten keruuseen liittyvät testitapaukset. Näissä testitapauksissa odotettu tulos on sama kentät täytettyinä ja tyhjiksi jätettyinä, sillä pinnan mittaukset tulee voida ohittaa esimerkiksi puuttuvan mittauslaitteiston tapauksessa.

Taulukko 5. Mittaustulosten keruunäkymän testitapaukset.

Id	**Tapaus**	**Odotettu tulos**
D1	Jätetään kaikki kentät tyhjiksi ja painetaan Ok-painiketta.	Ikkuna sulkeutuu.
D2	Täytetään kaikki kentät ja painetaan Ok-painiketta.	Ikkuna sulkeutuu.
D3	Painetaan Peruuta-painiketta.	Ikkuna sulkeutuu.
D4	Täytetään vain osa tiedoista ja painetaan Ok-painiketta.	Ikkuna sulkeutuu.

Taulukossa 6 käyttäjäarviointien keruuseen liittyvät testitapaukset, joilla varmistetaan, että navigointi arvioitavien ominaisuuksien välillä toimii. Täytettyjen tietojen tulee näkyä kentissä, mikäli käyttäjä navigoi takaisin jo arvioidun ominaisuuden näkymään.

Taulukko 6. Käyttäjäarvioinnin testitapaukset.

Id	**Tapaus**	**Odotettu tulos**
E1	Jätetään kaikki kentät tyhjäksi ja pyyhkäistään vasemmalle.	Arvioitava ominaisuus vaihtuu.
E2	Täytetään kaikki kentät ja pyyhkäistään vasemmalle.	Arvioitava ominaisuus vaihtuu.
E3	Pyyhkäistään oikealle.	Kenttiin aiemmin täytetyt tiedot ovat näkyvillä.
E4	Siirrytään viimeiselle sivulle.	Lähetä-painike näkyy näkymän alaosassa.
E5	Painetaan Lähetä-painiketta viimeisellä sivulla.	Testaus suoritettu -näkymä aukeaa, mikäli testattavia tuotepareja ei ole enempää. Mikäli tuotepareja on vielä testaamatta, siirrytään Testauksessa käytettävä laitteisto -näkymään.

Testauksen jälkeen tarkistettiin, että kaikki syötetyt tiedot olivat tallentuneet tietokantaan oikeassa muodossa. Järjestelmätestauksessa ei havaittu suuria puutteita ja kaikki mobiilisovellukselle tehdyt testit johtivat odotettuun tulokseen. Testauksessa havaittiin joitakin

käyttäjäkokemusta huonontavia asioita, kuten latausindikaattorin puuttuminen joissakin tilanteissa sekä käyttäjän ohjeistus virhetilanteissa. Näitä asioita tullaan jatkossa parantamaan ja testaamaan tarkemmin.

8 JOHTOPÄÄTÖKSET

Työn tavoitteena oli suunnitella ja toteuttaa mobiilisovellus hiomatuotteiden testaukseen ja siitä saatavan tiedon keräämiseen. Sovelluksen tarkoituksena oli lisätä hiomatuotteiden tuotekehityksessä tarvittavan tiedon määrää, jonka kerääminen erilaisissa ympäristöissä ja sovellusaloilla olisi muutoin vaikeaa.

Ennen sovelluksen suunnittelua perehdyttiin teknologioihin, joilla sovellus voidaan toteuttaa tehokkaasti. Suunnitteluvaiheessa suurimpana haasteena oli sovellukselle asetettujen vaatimusten muuttuminen projektin edetessä. Lopulta kuitenkin vaatimukset tarkentuivat siten, että sovellus voitiin toteuttaa nykyiseen muotoonsa. Projektin aikana täydentynyt vaatimusmäärittely käsiteltiin lyhyesti luvussa 4. Vaikka vaatimusten muuttuminen kehitystyön eri vaiheissa hidastikin projektin edistymistä, niillä oli positiivinen vaikutus lopputulokseen, ja lopulta sovelluksen käyttöliittymästä saatiin selkeä ja helposti käytettävä. Toisaalta myös vaatimusten täydentyminen projektin edetessä mahdollisti suunnittelun ja toteuttamisen aloittamisen melko pian ensimmäisten palaverien jälkeen, mikä onkin yksi ketterien kehitysmenetelmien eduista.

Sovellus toteutettiin Xamarin-kehitysympäristössä, joka mahdollistaa ohjelmakoodin jakamisen Android-, iOS- ja Windows Phone -käyttöjärjestelmille tarkoitettujen projektien kesken. Toteutuksessa suurimmat haasteet liittyivät sovelluksessa kerättävien tietojen tehokkaaseen käsittelyyn sovelluksen eri osien kesken, sekä niiden siirtoon sovelluksen ja palvelimen välillä. Loppujen lopuksi työn tuloksena syntyi käyttökelpoinen laajennus jo olemassa olevaan mobiilisovellukseen. Xamarin-ympäristö tuli nopeasti tutuksi C#-ohjelmointikielen ja natiivin Android-ohjelmistokehityksen ollessa ennalta jonkin verran tuttuja. Huonona puolena mainittakoon kehitysympäristön raskaus ja jokseenkin hankala käyttöönotto. Suosittelen Xamarin-kehitysympäristöä alustariippumattomien mobiilisovellusten kehitystyöhön, sillä parhaimmillaan sen avulla voidaan säästää runsaasti aikaa jakamalla ohjelmakoodia eri alustojen kesken. Pieniin tai yksinkertaisiin sovelluksiin en sitä kuitenkaan suosittele, sillä osa jaetun ohjelmakoodin avulla saavutettavasta ajan säästöstä hukataan aikaa vievässä käyttöönotossa. Monialustaisuus voi myös tehdä sovelluk-

sen arkkitehtuurista turhan kompleksisen sovelluksen vaatimuksiin nähden, jolloin kehitystyössä voi olla järkevämpää käyttää kullekin alustalle tarkoitettua natiivia kehitysympäristöä.

Sovellusta ja siihen integroitua laajennusta testattiin asiakkaan työtiloissa oikeassa käyttöympäristössä hiomakoneen kanssa. Testauksen tuloksena asiakkaan puolelta todettiin toiminnallisuuksien olevan vaatimusten mukaiset ja vastaavan suurimmalta osin odotuksia. Asiakastestausta tullaan tekemään lisää ennen laajennuksen julkaisemista muiden sovelluspäivityksien yhteydessä.

Jatkossa mobiilisovelluksen tueksi kehitetään erillinen verkkosovellus, jonka avulla voidaan luoda valmiita testitapauksia mobiilisovelluksen käyttäjille sekä tarkastella testeissä kerättyä tietoa ja testaajien palautetta tuotteista. Mobiilisovelluksen osalta kehityksen tulisi painottua yhä parempaan käytettävyyteen, tekstien lokalisointiin sekä ohjelmointivirheiden korjaamiseen niiden löytyessä. Laadunvarmistuksen tueksi tulisi luoda yksikkötestit, jotka voidaan suorittaa aina kun ohjelmakoodiin tehdään muutoksia, ja näin vähentää tuotantoversioihin päätyviä virheitä.

LÄHDELUETTELO

Bajaj, Piyush (2011). SQL Server – ANSI SQL and T-SQL, What? Why? How? [Verkkodokumentti]. [Viitattu 9.11.2017] Saatavissa: http://www.sqlservergeeks.com/sql-server-ansi-sql-and-t-sql-what-why-how/

Bluetooth SIG (2016). Our History [Verkkodokumentti]. [Viitattu 4.11.2016] Saatavissa: https://www.bluetooth.com/media/our-history

Britch, David (2017). *Enterprise Application Patterns using Xamarin.Forms.* Redmond: DevDiv, .NET and Visual Studio product teams (Microsoft corporation).

CSR plc (2010). Bluetooth low energy [Verkkodokumentti]. [Viitattu 10.1.2017] Saatavissa: https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=227336

Devatus Oy (2017). Ketterämpää ohjelmistokehitystä [Verkkodokumentti]. [Viitattu 28.10.2017] Saatavissa: http://www.devatus.fi/

ECMA-404 (2003). *The JSON Data Interchange Format.* 14 s.

Fielding, Roy (2000). *Architectural Styles and the Design of Network-based Software Architectures.* Irvine: University of California. Väitöskirja. ISBN:0-599-87118-0.

Google and Open Handset Alliance (N.d.) (2017a). Android Reference [Verkkodokumentti]. [Viitattu 16.10.2017] Saatavissa: https://developer.android.com/reference/

Google and Open Handset Alliance (N.d.) (2017b). Android API Guides [Verkkodokumentti]. [Viitattu 10.10.2017] Saatavissa: https://developer.android.com/guide/index.html

Haikala, Ilkka & Jukka Märijärvi (2000). *Ohjelmistotuotanto*. 7. painos. Helsinki: Talentum. 414 s. ISBN 951-762-769-6.

IDC Research Inc. (2017). Smartphone OS Market Share, 2017 Q1 [Verkkodokumentti]. [Viitattu 8.11.2017] Saatavissa: https://www.idc.com/promo/smartphone-market-share/os

ISO 5349-1 (2001). *Mechanical vibration — Measurement and evaluation of human exposure to hand-transmitted vibration.* Part 1: General requirements.

Ivanov, Vladimir (2015). Going with MVVM on Android via Data Binding [Verkkodokumentti]. Saatavissa: http://cases.azoft.com/mvvm-android-data-binding

Kiran, Devi (2014). REST API - Future of SOA [Verkkodokumentti]. [Viitattu 16.11.2017] Saatavissa: http://www.kickstartpros.com/2014/07/rest-api-future-of-soa.html

Lehtonen, T., S. Tuomivaara, V. Rantala, M. Känsälä, T. Mäkilä, T. Jokela, K. Könnölä, M. Kaisti, S. Suomi, M. Isomäki & M. Ylitolva (2014). *Sulautettujen järjestelmien ketterä käsikirja*. Turku: Painosalama Oy. 98 s. ISBN 978-951-29-5838-2.

Maciaszek, Leszek A. & B. L. Liong (2005). *Practical Software Engineering: A Case Study Approach.* Harlow: Pearson Education Limited. 825 s. ISBN 0-321-20465-4.

Mirka Oy (2016a). myMirka digitaalisille palveluille ja liitettävyydelle [Verkkodokumentti]. [Viitattu 1.12.2016] Saatavissa: http://www.mirka.com/fi/fi_mymirka/

Mirka Oy (2016b). e-Field Testing Process. 2 s. Julkaisematon.

Mirka Oy (2016c). *Product Verification & Validation Report*. 1 s. Julkaisematon.

Mirka Oy (2017a). Historia [Verkkodokumentti]. [Viitattu 3.11.2016] Saatavissa: http://www.mirka.com/fi/fi/-Top-Menu-/Mirkasta/#/Historia

Mirka Oy (2017b). *Abrasive Verification Form.* 1 s. Julkaisematon.

Oliana, Rubina (2016). *The Internet of Things; The Next Big Thing for New Product Development?*. University of Twente. The Faculty of Behavioral, Management and Social sciences. Kandidaatintutkielma.

OWASP Foundation (2017). SQL Injection Prevention Cheat Sheet [Verkkodokumentti]. [Viitattu 17.1.2018] Saatavissa: https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet#Defense_Option_2:_Stored_Procedures

Panigrahy, Nilanchala (2015). *Xamarin Mobile Application Development for Android*. 2. painos. Packt Publishing. 298 s. ISBN: 978-1-78528-037-5.

Peppers, J., G. Taskos & C. Bilgin (2016). *Xamarin: Cross-Platform Mobile Application Development.* Packt Publishing. 262 s. ISBN: 978-1-78712-012-9.

Reynolds, Mark (2014). *Xamarin Mobile Application Development for Android.* Packt Publishing. 168 s. ISBN: 978-1-78355-917-6.

Shukkur, Shamlia (2013). Understanding the basics of MVVM design pattern [Verkkodokumentti]. [Viitattu 5.8.2017] Saatavissa: https://blogs.msdn.microsoft.com/msgulfcommunity/2013/03/13/understanding-the-basics-of-mvvm-design-pattern/

Simpson, Edward (2015). Manufacturing Process Verification Versus Validation: Which Do You Need? [Verkkodokumentti]. [Viitattu 3.11.2016] Saatavissa: http://www.labcompare.com/2719-Blog/170873-Manufacturing-Process-Verification-Versus-Validation-Which-Do-You-Need/

Smyth, Neil (2016). *Android Studio 2 Development Essentials.* CreateSpace Independent Publishing Platform. 748 s. ISBN 978-1532853319.

Statista (2018). Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions) [Verkkodokumentti]. [Viitattu 9.1.2018] Saatavissa: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

Suomen Työterveyslääkäriyhdistys ry (2016). Tärinäaltistumisen arviointi työpaikalla ja työterveyshuollossa [Verkkodokumentti]. [Viitattu 5.1.2017] Saatavissa: http://www.ebm-guidelines.com/dtk/ltk/avaa?p_artikkeli=ttl00438&p_haku=hoitosuositus

Systä, Tarja (2009). OHJ-5201 Web-palveluiden toteutustekniikat -luentomateriaali [Verkkodokumentti]. Tampere: Tampereen teknillinen yliopisto. [Viitattu 3.3.2017] Saatavissa: http://www.cs.tut.fi/kurssit/OHJ-5201/materiaali/

Thakur, Aniket (2015). Difference between Dalvik and ART runtimes in Android [Verkkodokumentti]. [Viitattu 8.11.2017] Saatavissa: http://opensourceforgeeks.blogspot.fi/2015/02/difference-between-dalvik-and-art.html

Thompson, Timothy J., P. J. Kline & C. B. Kumar (2008). *Bluetooth Application Programming with the Java APIs Essentials Edition*. 1. painos. Burlington: Morgan Kaufmann. 394 s. ISBN 978-0-12-374342-8.

Tilkov, Stefan & Steve Vinoski (2010). Node.js: Using JavaScript to Build High-Performance Network Programs. In: *IEEE Internet Computing* 14: 6, 80–83. 1089-7801.

Työterveyslaitos (2014a). Käsitärinä [Verkkodokumentti]. [Viitattu 19.10.2016] Saatavissa: https://www.ttl.fi/tyoymparisto/altisteet/tarina/kasitarina/

Työterveyslaitos (2014b). Käsitärinälle altistuvien terveysseuranta työterveyshuollossa [Verkkodokumentti]. [Viitattu 19.10.2016] Saatavissa: http://www.partner.ttl.fi/fi/tyoymparisto/tarina/tarinan_torjunta/sivut/default.aspx

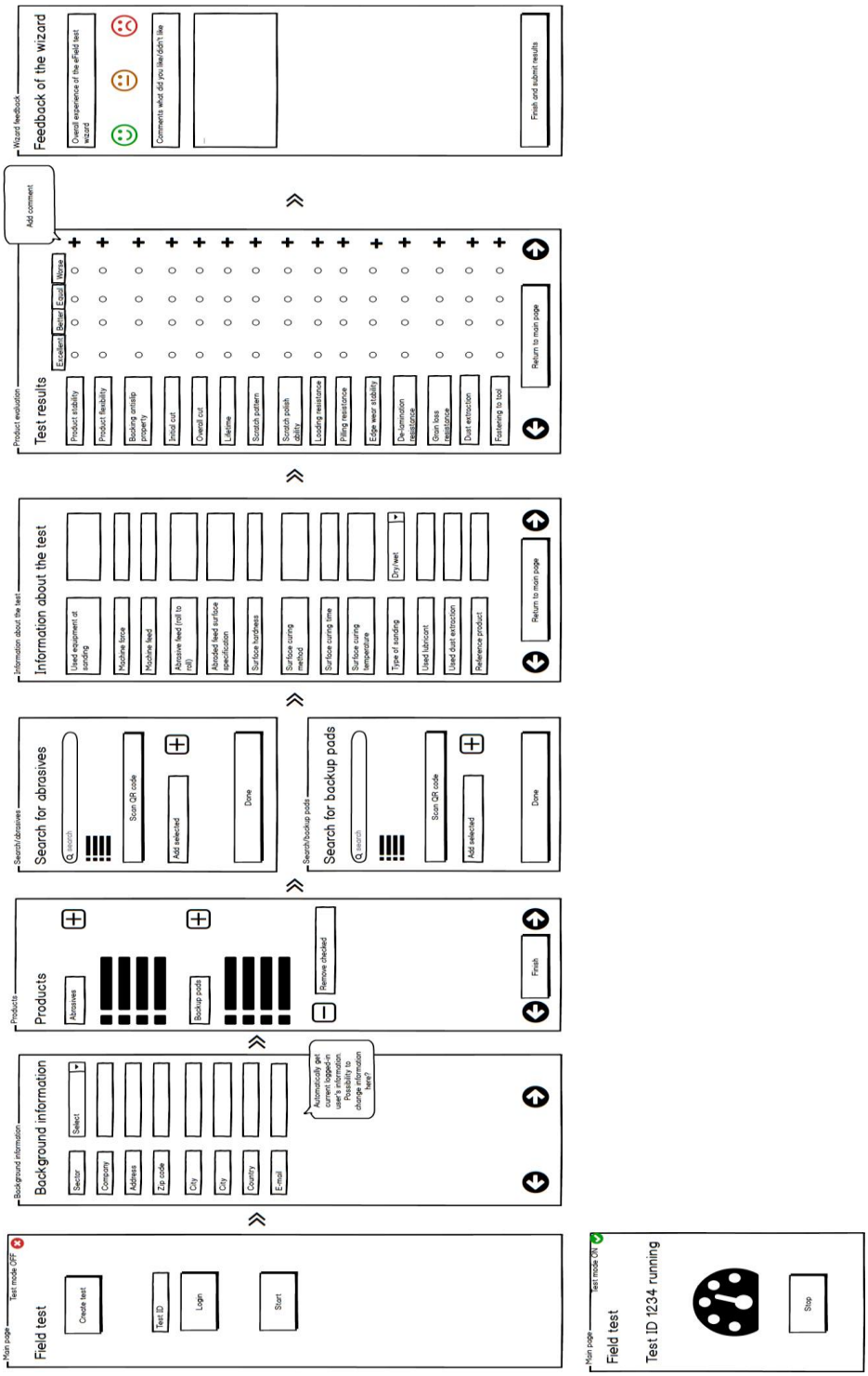
Työterveyslaitos (2014c). Tärinän torjunta [Verkkodokumentti]. [Viitattu 19.10.2016] Saatavissa: http://www.ttl.fi/fi/tyoterveyshuolto/ammattitaudit/tavallisimpia_ammattitauteja/tarinatauti/Documents/Tärin%C3%A4n%20terveysseurantaohjeet2.pdf

Ward, James, J. Clarkson, D. Bishop & S. Fox (2002). *Good Design Practice For Medical Devices And Equipment – Design Verification.* Cambridge: University of Cambridge Engineering Design Centre & University of Cambridge Institute for Manufacturing. 91 s. ISBN 1-902546-12-1.

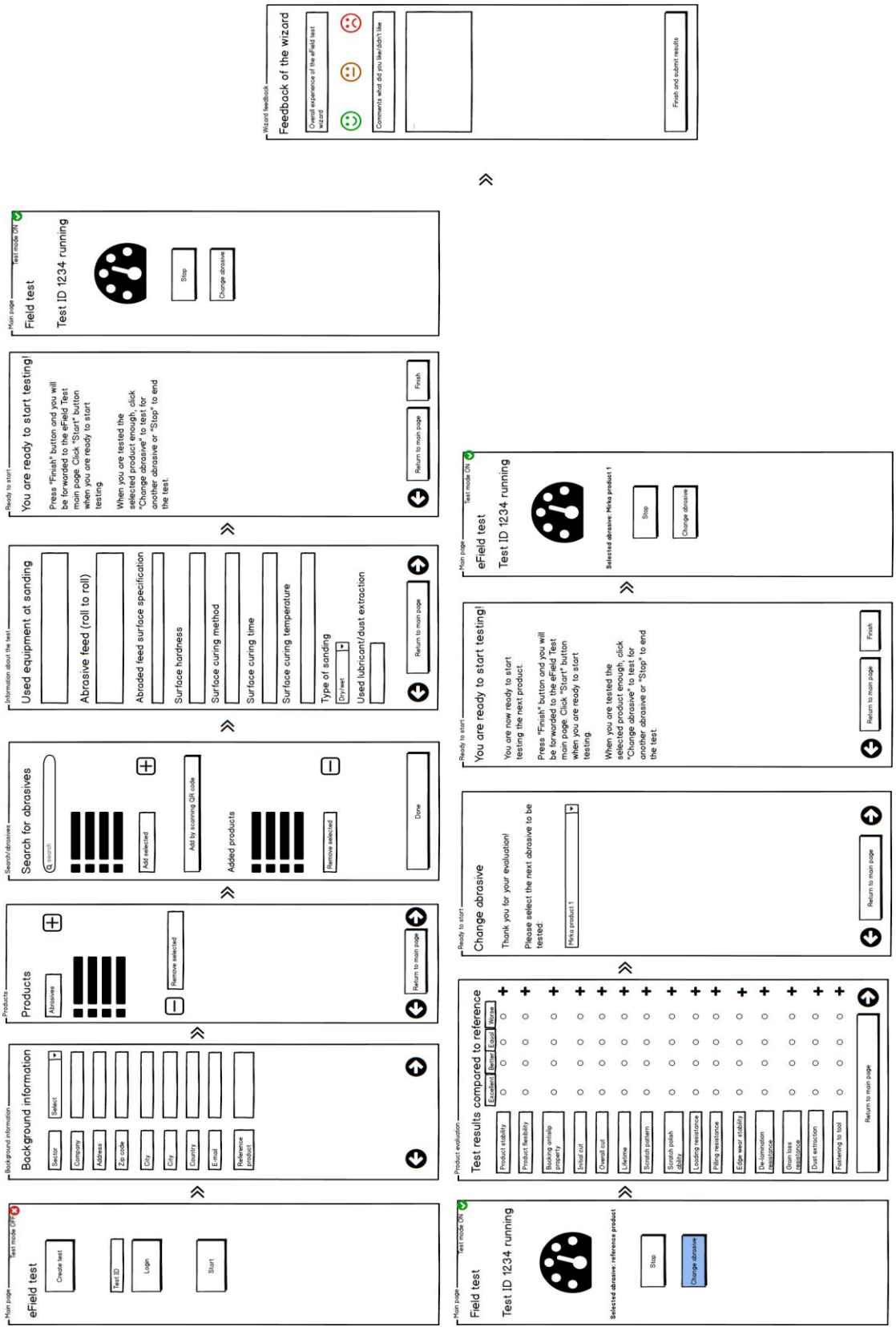
Xamarin Inc. (2016). Xamarin Platform [Verkkodokumentti]. [Viitattu 28.11.2016] Saatavissa: https://www.xamarin.com/platform

LIITTEET

LIITE 1. Ensimmäinen käyttöliittymäsuunnitelma



LIITE 2. Toinen käyttöliittymäsuunnitelma



LIITE 3. eField Test -moduulin näkymien ja näkymämallien luokkakaavio