

Roni Oesch

# Manuaalitestin suunnittelu ja toteutus automaatiotestiksi



Insinööri (AMK)

Tieto- ja viestintätekniikka

Kevät 2022



**KAMK • University  
of Applied Sciences**

## **Tiivistelmä**

**Tekijä:** Oesch Roni

**Työn nimi:** Manuaalitestin suunnittelu ja toteutus automaatiotestiksi

**Tutkintonimike:** Insinööri (AMK), tieto- ja viestintätekniikka

**Asiasanat:** Testaaminen, Automaatiotestaus, Mobiili, Robot Framework, Appium

Opinnäytetyön aiheena oli suunnitella ja toteuttaa manuaalitestin automaatiotestiksi mobiilialustalle. Työn toimeksiantajana oli Bittium Wireless Oy. Testi testaa Bittium MedicalSuite Mobile -sovelluksen käyttöliittymätoimintoja sekä kardiologisen mittalaitteen Faroksen yhteyttä palveluun.

Testin koodikielenä toimi Robot Framework ja Python, joihin lisänä käytettiin Appiumin kirjastoa. Laitekoonpano pitää sisällään Windows-pohjaisen kannettavan tietokoneen, Faroksen sekä sen testaamiseen käytettävän testipenkin ja Bittium Tough Mobile -matkapuhelimen. Ohjelmistoina olivat PyCharm ohjelmointiympäristönä, testipenkin oman ohjelmiston Linux-käyttöjärjestelmäisessä virtuaalikoneessa ja Appiumin sovellus matkapuhelinsovelluksen tarkastelua varten sekä palvelimena testin aikana.

Projektin haasteista huolimatta lopullinen automaatiotesti onnistui loistavasti. Se täytti kaikki sille asetetut vaatimukset sekä jätti erinomaisen pohjan testin mahdolliselle jatkokehitykselle.

## **Abstract**

**Author:** Oesch Roni

**Title of the Publication:** Planning and Implementation of an Automation Test Based on a Manual Test

**Degree Title:** Bachelor of Engineering, Information and Communications Technology

**Keywords:** Testing, Automation Testing, Mobile, Robot Framework, Appium

The theme of the thesis was to plan and execute an automation test based on an existing manual test on a mobile platform. The project client was Bittium Wireless Oy. The test tests user interface elements of the Bittium MedicalSuite Mobile application and its connection to the cardiologic measuring device Faros.

The programming languages of the test were Robot Framework and Python in addition of Appium Library. Device setup included a Windows laptop, a Faros with its dedicated testbench and a Bittium Tough Mobile cellular phone. Programs included PyCharm as the programming environment, the testbench software running on a virtual machine with Linux as the operating system and the Appium application for inspecting the mobile application and running the server during running the test.

Despite the obstacles, the final automation test was successful. It met the expectations and left an excellent base for future development of the test.

## Sisällys

1	Johdanto .....	1
2	Testaaminen .....	2
2.1	Manuaalitestaaaminen .....	3
2.2	Automaatiotestaaminen .....	3
2.3	Automaatiotestaamisen hyödyt ja haasteet opinnäytetyössä .....	3
3	Laitteistot, ohjelmistot ja niiden suhde toisiinsa.....	5
3.1	Faros .....	6
3.2	Testipenkki testauksen alustana .....	7
3.3	Bittium Tough Mobile -matkapuhelin Bittium MedicalSuite Mobile -sovelluksen alustana .....	8
3.4	Tietokone ja sen ohjelmistot .....	9
4	Testin kulku.....	11
4.1	Testin ajoa edeltävät valmistelut .....	11
4.2	Testin aloitus .....	13
4.3	Mittauksen luominen ja sen kulku .....	14
4.4	Testin lopetus .....	18
5	Päätäntö .....	20
5.1	Haasteet työn toteutuksessa.....	20
5.2	Testin ja sen ympäristön jatkokehitys .....	21
	Lähteet .....	23

## Liitteet

## Termit ja lyhenteet

API	Application Programming Interface, komponenttien välinen raja ohjelmoitavassa järjestelmässä
Appium	Mobiilikehityksessä käytetty ohjelmisto ja kirjasto Robot Frameworkille
Automaatiotestaaminen	Testaamista, jossa testit ajetaan automaattisesti käynnistyksen jälkeen
Bittium HolterPlus	Kardiologinen mittaussarja
Bittium MedicalSuite	Bittiumin palvelu lääketieteelliselle datalle
Bittium MedicalSuite Mobile	Mobiilisovellus edellä mainitulle palvelulle asiakkaan käyttöön
Faros	Kardiologinen mittalaite, joka kiinnitetään rintaan
Manuaalitestaaaminen	Testaamista, jossa testit ajetaan käsin testattavalle tuotteelle
PyCharm	Ohjelmistokehitysympäristösovellus
Robot Framework	Ohjelmointikieli
Testipenkki	Ohjelmistokehityksessä käytetty ulkoinen laite tai laitteisto, joka on alustana testattavalle ohjelmalle tai ohjelmistolle

## 1 Johdanto

Opinnäytetyö on tehty tilauksesta Bittium Wireless Oy:lle työharjoittelun jatkeena, ja sen aihe on automaatiotestin suunnittelu ja toteutus mobiilisovellukselle yrityksen ehdotuksesta. Bittium Wireless Oy on tietoturvallisten laitteita ja palveluja kehittävä yritys. Se toimii Suomessa, Euroopassa, Amerikassa ja Aasiassa. Suomessa toimipisteet ovat Oulu, Tampere, Kajaani, Kuopio ja Espoo. Päätuotteet ovat taktinen sekä tietoturallinen kommunikaatio, langattomat verkkototeutukset ja biosignaalien mittaust. Vuonna 2020 liikevaihto oli 78,4 miljoonaa euroa ja henkilöstön määrä keskimäärin 682. [1.]

Tuotetestaaminen on osa jokaista tuotekehityksen elinkaarta. Se voidaan toteuttaa usealla eri tavalla ja eri kohdissa tuotekehitystä. Hyvä tuotetestaus on joustavaa ja mukautuu kehitettävän tuotteen tarpeisiin. Opinnäytetyössä toteutettu testi testaa suurpiirteisesti kohdesovelluksen toimintoja. Testiä on aiemmin ajettu manuaalisesti, mutta se haluttiin automatisoida toistettavuuden vuoksi.

Opinnäytetyössä kehitetty automaatiotesti on pieni osa laajempaa tuotekehityksen testaamista ja siksi se on toteutettu tuoteprojektin vaatimusten mukaisesti. Automatisointi on ajankohtainen työkalu tuotetestaamisessa vapauttaen resursseja pois toistuvien manuaalitestien ajamisesta. Opinnäytetyön ydin oli suunnitella ja toteuttaa valmiina oleva manuaalitestin mahdollisimman automaattiseksi ja helppokäyttöiseksi. Onnistunut testi käy tarvittavat vaiheet läpi käynnistettyä ja sulkee lopulta testin ja siivoaa ympäristön valmiiksi seuraavaa ajoa varten. Sovelluksen ominaisuuksien toiminta on tärkeää testata toistuvasti. Erityisten tärkeää on testata toimivuus ominaisuuksien lisäämisen tai päivittämisen jälkeen. Tällöin rutiinitestin automatisointi säästää paljon aikaa testaajilta testata muita, hienovaraisempia testejä.

Tavoitteena oli luoda yritykselle laadukas automaatiotesti ja dokumentaatio testistä. Tärkeää oli myös oppia käyttämään uusia työkaluja uudessa ympäristössä ja tuottamaan vaatimusten mukainen lopputulos.

## 2 Testaaminen

Testaaminen on korvaamaton vaihe jokaista tuotekehitystä. Testaamisella varmistetaan, että tuote toimii kuten siltä vaaditaan ja saadaan markkinoille paras mahdollinen versio. Ohjelmistokehityksen testauksessa varmistetaan, ettei ohjelmakoodissa tai sen toiminnassa ole virheitä ja että asiakkaalla on paras mahdollinen käyttökokemus ohjelmaa käytettäessä. [2.]

Työvoima ja aika ovat tärkeitä sekä kalliita resursseja tuotekehityksessä. Oikeanlaisen testaus-suunnitelman laatiminen lisää läpinäkyvyyttä projektin etenemiseen ja on ratkaisevaa aikataulussa sekä budjetissa pysymisen kannalta. Jos tuotteen testaaminen on ollut vajavaista, tuotteen julkaisu voi viivästyä tai markkinoille saatetaan tahtomatta päästää viallinen tuote. Erilaisten testausmenetelmien käytösuhde riippuu projektin koosta, kehitettävästä tuotteesta sekä kehityssuunnitelmasta.

Huolimatta testauksen tärkeästä roolista tuotekehityksessä ja sen vaatimasta budjetista, testamista on tutkittu huomattavasti vähemmän kuin muita tuotekehityksen osa-alueita. Tämä saattaa johtaa tuotteen laadun laskemiseen, mikäli testaamiseen ei ostata asennoitua tarpeeksi perusteellisesti. [3, s. 291, 312.]

Tavallisin muoto testata on V&V, validointi ja verifiointi, jossa tuotetta testataan jatkuvasti kehityksen aikana ja varmistetaan tuotteen oikeellinen toiminta ennen julkaisua. Siihen usein sisältyy kolme rahaa vievää vaihetta: itse ajaminen, muutosten tekeminen ja korjaaminen tuotannon aikana ja jälkimuutokset, mikäli tuote ei toimi toivotulla tavalla julkaisun jälkeen. [3, s. 293.]

Testaamisen muotoja on monenlaisia. Testin ajamisessa voidaan käyttää fyysistä laitteistoa tai simulaatioympäristöä. Tuotetta voidaan testata myös projektin sisällä, suljetussa tai avonaisessa käyttäjätestauksessa. Riippuen tilanteesta testaajilla on eritasoinen pääsy tarkastelemaan tuotetta. Kun testaaja pääsee näkemään tuotteen sisältöön, kuten esimerkiksi ohjelmistokoodiin, kutsutaan testaamista valkolaatikkotestaamiseksi. Mustalaatikkotestaamisessa testaajalla on pääsy ainoastaan tuotteen toiminnallisuuteen, kuten kella tahansa loppukäyttäjällä. Mahdollisuuksien mukaan voi käyttää myös harmaalaatikkotestausta, joka on näiden kahden välimuoto. [2.]

## 2.1 Manuaalitestaaaminen

Manuaalisesti testaaminen käytännössä tarkoittaa ihmisen ajavan käsin tuotteelle suunnitellut testit, joiden tarkoituksena on löytää kehityksessä sattuneita virheitä.

Manuaalitestaus on korvaamaton vaihe tuotekehityksessä vielä toistaiseksi, sillä testaajan kirjoittama raportti tilanteista antaa huomattavasti enemmän informaatiota kuin tietokoneen antama loki. Manuaalitestaaaja pystyy myös huomattavasti tarkemmin vastaamaan käyttäjän kokemusta kuin tietokone.

Uudet implementaatiot ja ominaisuudet vaativat tarkan testaamisen, mikä tuo paljon toistoa manuaalitestaaajalle. Muutokset ohjelmistoissa vaikuttavat usein myös aiemmin lisättyihin ominaisuuksiin, joten on välttämätöntä varmistaa tuotteen toimivuus kokonaisuutena.

## 2.2 Automaatiotestaaminen

Automaatiotestaaminen on yhä yleistynyt osa tuotetestaamista. Manuaalitestauksen sijaan automaatiotestaaja ohjelmoi testit, jotka tietokone sitten ajaa itsenäisesti. Testien lopuksi ohjelmisto palauttaa tulokset sekä mahdollisen ylimääräisen ennalta määrätyn informaation.

Regressiotestaaminen tarkoittaa yksinkertaisuudessaan testien uudelleenajamista toistuvasti. Testit voidaan uudelleenajaa jatkuvasti tai esimerkiksi aina kun ohjelmistoa on päivitetty tai muokattu. Ajamalla standardisoidut testit uudelleen varmistutaan, etteivät muutokset ole aiheuttaneet virheitä alkuperäisiin ominaisuuksiin. Tällaiseen tarkoitukseen automaatiotestaaminen on erittäin optimaalinen ratkaisu, mikäli kehityselinkaari on pitkä. [2.]

## 2.3 Automaatiotestaamisen hyödyt ja haasteet opinnäytetyössä

Erityisesti suurissa projekteissa jatkuva testaaminen on pakollinen osa laadukasta tuotekehitystä, jolloin automaatiotestauksen tarve kasvaa. Mikäli manuaalitestaukseen käytettävä aika ylittää selkeästi automaatiotestien laatimiseen ja huoltoon käytetyn ajan, on automaatiotestaamisesta hyötyä projektille.



Tietokoneet ovat täsmällisempiä ja tarkempia kuin ihmiset. Ohjelmiston pitämä automaattinen kirjanpito virheilmoituksista ja muista vaadituista merkinnöistä on suuri apu tuotteen kehittäjille, ja ne valmistuvat nopeammin ja luotettavammin kuin manuaalinen raportointi. Automaatiotestien hoitaessa suuren määrän säännöllistä toistoa, resurssit ohjautuvat tuotteen kehitykseen. Tällöin vapautuu työntekijöitä mahdollisesti mielekkäämpiin ja palkitsevimpiin työtehtäviin, jotka myös pitävät yllä urakehitystä.

Tässä opinnäytetyössä laadittu ja toteutettu automaatiotesti ei ollut osa tuotekehityksen suunnitelmaa, vaan ominaisuuksia testattiin manuaalisesti. Testi tuli toimivaksi ja oleelliseksi osaksi tuotekehitystä ja oli erittäin hyödyllinen projektille.

Automaatiotestaus tuo mukanaan myös haasteita. Testien kehitys ja hiominen vaativat aikaa ja ylläpitämistä. Vaikka automaatiotesti raportoi tuloksista tarkasti, se ei ainakaan toistaiseksi vielä täysin korvaa ihmistä testausprosessissa. Järjestelmällisesti suorittava ohjelmisto saattaa jättää raportoimatta virheen, joka ei esiinny sen testissä. Ennalta ohjelmoitu automaatiotesti ei välttämättä vastaa käyttäjäkokemusta ohjelmasta, sillä tietokone ei pysty arvioimaan ohjelmiston helpokäyttöisyyttä.

Tämä opinnäytetyö käsittelee automaatiotestaamista mobiililla, jonka isoimpia haasteita on hitaus. Testien laajentuessa ajankäytön suunnittelu pitää ottaa huomioon. Laajempi ja pidempi testi on myös herkempi satunnaisille ajosta riippuville virheille, jotka eivät välttämättä yllä itse ohjelmistoihin. Kosketushäiriöt tai tavallista pidemmät latausajat saattavat kaataa testin kesken ajon, jolloin loppuosa testistä jää ajamatta.

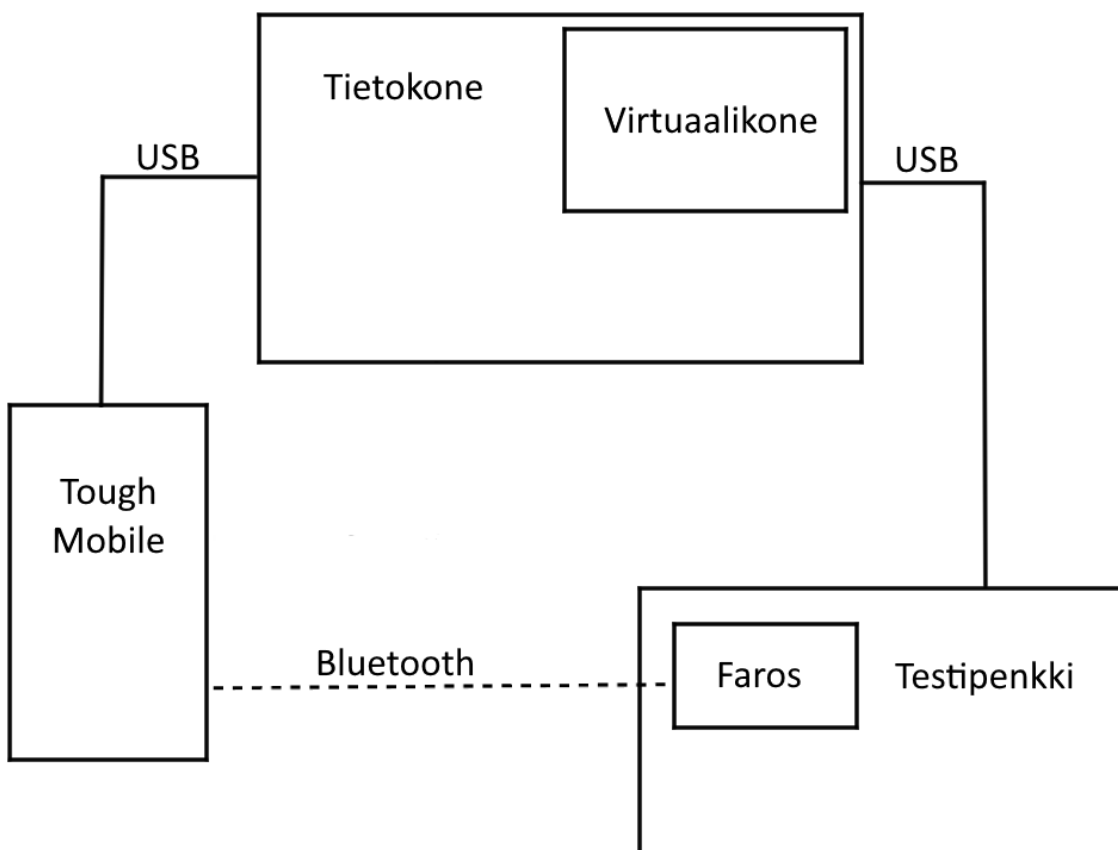
Opinnäytetyössä toteutetusta testistä tuli lopulta hyvin herkkä ja haavoittuvainen muutokselle. Erityisesti mobiililla testaamisen hitauden vuoksi osa testin ominaisuuksista olivat haastavia luoda. Mittauksen luomisessa avainasemassa olevaan ylläpitotilaan pääsee napauttamalla Bitiumin logoa useasti kotinäkylässä. Appiumin kirjastossa ei kuitenkaan ole kommentoa useasti klikkaamiselle ja Click-avainsanan toistaminen vaaditun määrän kestää moninkertaisesti liian pitkän ajan. Toteutus lopulta käyttää näytön koordinaatteja sijainnin löytämiseksi, joka on todella haavoittuvainen laitteen näytön resoluution muuttumiselle. Vastaavanlainen ratkaisu piti valita upotettuun ruutuun, joka vaatii skrollauksen alaspäin. Tämä ratkaisu käyttää pyyhkäisyyn näytön prosentuaalisia koordinaatteja, jotka on valittu mahdollisimman joustavaksi resoluution vaihtamista varten.

### 3 Laitteistot, ohjelmistot ja niiden suhde toisiinsa

Automaatiotestin suunnitteluun ja luomiseen käytettiin iso määrä laitteita ja ohjelmistoja. Tässä luvussa esitellään kaikki testiin käytetyt laitteistot ja ohjelmistot, niiden toiminta ja suhde toisiinsa.

Projektin keskusyksikkönä toimi Windows-alustallinen kannettava tietokone. Linux-alustaiset ohjelmat ajetaan virtuaalikoneen puolella. Jokainen fyysinen laite oli USB-kaapelin kautta kiinni tietokoneessa, ja osa kommunikoi keskenään Bluetooth-yhteyden avulla [Kuva 1].

Faros on ainoa laitteista, joka ei voi olla latauksessa testin aikana. Tämä lisää testin ajoon manuaalista huoltoa.



Kuva 1: Laitteiden liitokset

### 3.1 Faros

Useiden sydänsairauksien etsimiseen käytetään elektrokardiogrammia, EKG:tä, joka on erinomainen kuvaamaan sydämen rytmiä. Erilaiset rytmihäiriöt näkyvät nauhalla poikkeuksina, joista lääkäri voi tehdä oikeanlaisen diagnoosin. [4.] EKG toimii kokonaisuudessaan 12-kanavaisena, joista puolet mittaa rintakehää ja puolet raajoja. Kanavien elektrodipaikat ovat standardisoidut, sillä niiden sijainnit vaikuttavat sydänpäivän muotoon. [5.]

Faros on Bittium Biosignalsin kehittämä EKG-mittalaite. Sen avulla etsitään piileviä sydänsairauksia ja sydämen toimintahäiriöitä. Faros toimii yhdessä rintaan kiinnitettävien elektrodien ja oireita tarkkailevan sovelluksen kanssa. Laitteet on suunniteltu mahdollisimman pieniksi ja helppokäyttöisiksi; ne ovat kevyitä ja vedenkestäviä toimiakseen usean päivän mittausjaksoissa. [6.]

Faros kiinnitetään rintaan Bittium OmegaSnapin elektrodien avulla, ja Faros lähettää Bluetooth-yhteyden välityksellä sydänpäivää matkapuhelimeen määritellyn ajan. Data tallennetaan kardiologien tarkastelua varten Bittium MedicalSuite -palveluun. Alla havainnollistava kuva laitteista ja niiden tehtävistä. [2.]

**Bittium HolterPlus™ Starter Kit**

The small, lightweight and waterproof Bittium Faros™ ECG recorder is easy to use and set up on a patient.

During an ECG recording, Bittium MedicalSuite™ mobile application enables the patient to report symptoms, activity and sleep via a digital diary.

The recording device transmits the ECG data via bluetooth to the mobile device which, in turn, serves as the gateway for sending the cardiac data to the web portal for analysis. Cardiologist is able to do daily checks and adjust the individual recording length based on the received ECG data.

The patient's recorded ECG information is analyzed with Bittium Cardiac Navigator™ and turned into a comprehensive report, including treatment instructions by the cardiologist. The treating doctor is notified when the analysis is ready and a report is made available via the web service.

**Hardware**

- Bittium Faros™ 360 Waterproof Sensor
- Bittium Faros™ 5-electrode Cable Set
- Bittium MedicalSuite™ Mobile Device
- Bittium OmegaSnap™ ECG Electrode (3 pcs)

**Software**

- Three (3) Month license for Bittium MedicalSuite™ Service Platform
- Three (3) Month license for Bittium Cardiac Navigator™ Online version

Kuva 2: Bittium HolterPlus -esittelykuva Bittiumin verkkosivuilta [1]

### 3.2 Testipenkki testauksen alustana

Testipenkki (englanniksi Test Bed tai Testing Bed) on ohjelmistokehityksessä käytetty ulkoinen laite tai laitteisto, joka on alustana testattavalle ohjelmalle tai ohjelmistolle. Testipenkki voi esimerkiksi olla identtinen keskusyksikkö tietokoneeseen, jolla ohjelmistoa on ohjelmoitu, mutta laite on pääasiassa varattu ohjelmiston testaamiseen. [2.]

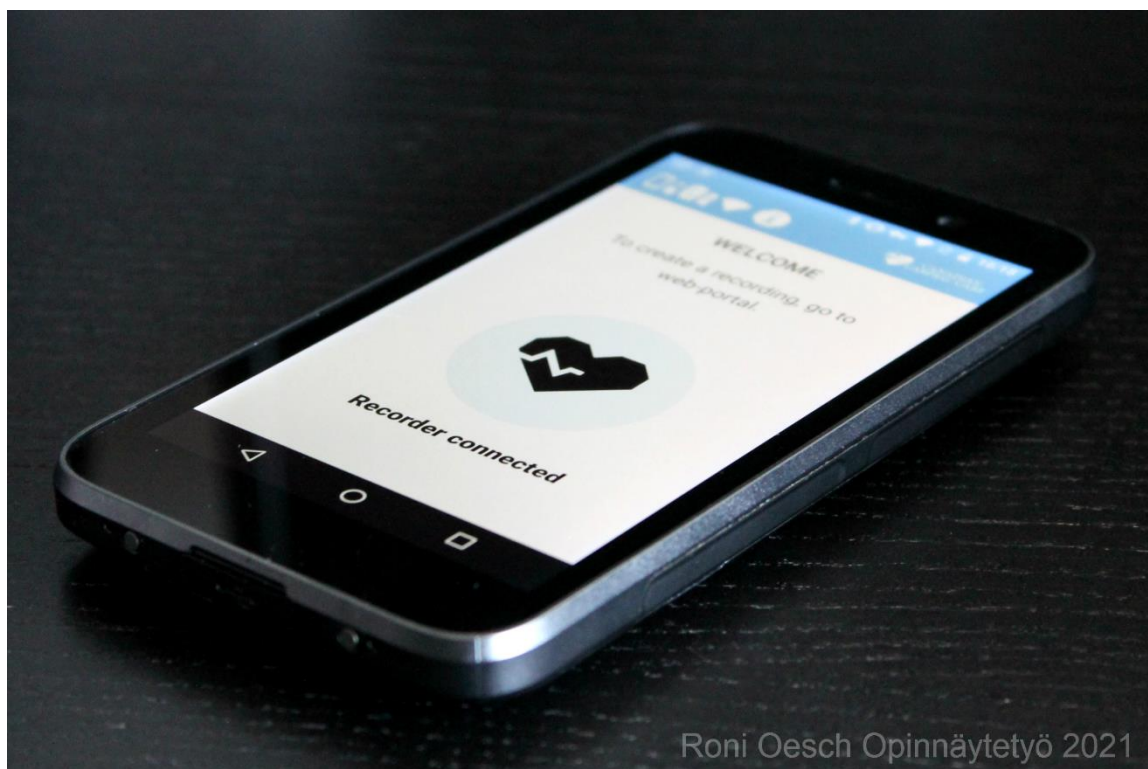
Tässä opinnäytetyössä testipenkki Farokselle on yksinkertainen sydänekäyrää simuloiva laite. Testipenkki esittää sydänelektrodeja ja voi ajaessa simuloida yksi- tai monikanavaista käyrää riippuen käyttäjän asetuksista. Tässä testissä käytetään yksikanavaista asetusta. Testipenkki kiinnitetään USB-johdolla tietokoneeseen, josta sitä halutaan ajaa. Faros kiinnitetään testipenkkiin mikro-USB-kaapelilla alla olevan kuvan mukaisesti. Faros ei saa virtaa kaapelin kautta, vaan se täytyy ladata erikseen ja toimii paristollaan testin aikana. [Kuva 3.]



Kuva 3: Faros kytkettynä testipenkkiin.

### 3.3 Bittium Tough Mobile -matkapuhelin Bittium MedicalSuite Mobile -sovelluksen alustana

Tough Mobile on Bittium Wirelessin kehittämä matkapuhelin [Kuva 4]. Siinä on kaksi käyttöjärjestelmää kaksoiskäynnistyksenä, kovennettu Android henkilökohtaiseen käyttöön ja tietoturvalinen Bittium Secure OS. Matkapuhelin on kehitetty toimimaan tietoturvallisena ja kestäväenä vaihtoehtona muun muassa viranomaisten tiedonvälitykseen. [7.]



Kuva 4: Bittium Tough Mobile, jossa auki Medical Suite Mobile -sovellus

Tässä testissä Bittium Tough Mobile toimi alustana Bittium MedicalSuite Mobile -sovellukselle. Mittauksen aikana Faros ja Bittium Tough Mobile kommunikoivat Bluetooth-yhteyden kautta, joten matkapuhelin on pidettävä yhteyden kantaman matkan päässä. Bittium MedicalSuite on palvelu, joka jakaa tietoturvallisesti informaatiota lääkärin, laitosten ja potilaiden välillä. Palvelu on räätälöitävissä tilanteenmukaisiin tarpeisiin. Testi käyttää palvelua Bittium MedicalSuite Mobile -sovelluksen kautta ja kommunikoi verkkopalveluun API-komentojen avulla.

Asiakkaan käytössä Bittium MedicalSuite Mobile -sovellus on kiinnitettynä matkapuhelimen näyttöön eli sovelluksesta ei pääse muualle puhelimeen. Automaatiotestauksen aikana tämä ominaisuus jouduttiin poistamaan testattavasta versiosta, sillä testin käynnistys vaatii pääsyn puhelimen asetuksiin ja mahdollisuuden käynnistää sovellus uudelleen.

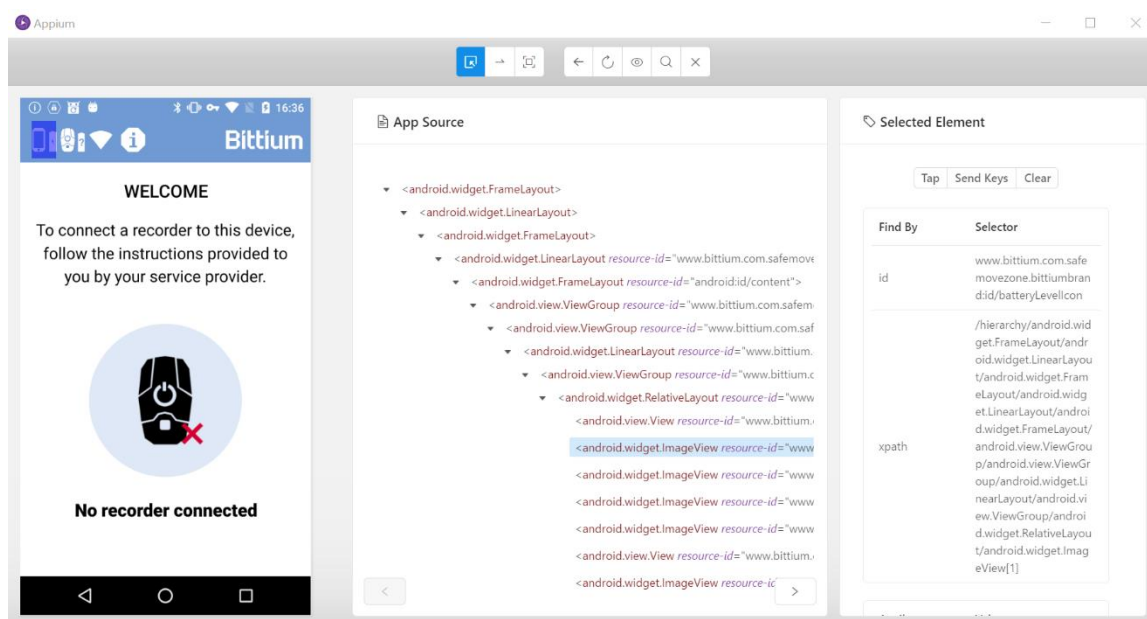
### 3.4 Tietokone ja sen ohjelmistot

Projektin pääyksikkönä toimi Windows-pohjainen kannettava tietokone. Siltä ajettiin testiä sekä kaikkia sen tarpeellisia ohjelmistoja, ja siihen oli yhdistettyä testiin tarvittava laitteisto USB-kaapeleilla.

Testin ohjelmointikielenä toimi Robot Framework, joka on avoimen lähdekoodin automaatioviitekehys. Kieli on kehitetty Python-ohjelmointikielellä mahdollisimman joustavaksi erilaisiin testausympäristöihin ja suunniteltu jatkettavaksi Python- ja Java-kirjastoilla projektin tarpeen mukaan. Syntaksi on rakennettu mahdollisimman helppolukuiseksi ja -käyttöiseksi ihmiselle. Robot Framework on ilmainen käyttää ilman lisenssimaksuja. [8.]

Testin kehitys- ja ajoympäristönä käytettiin PyCharm-ohjelmaa, johon lisättiin Appiumin kirjasto mahdollistamaan mobiilisovelluksen testaamisen. Appium on mobiilisovellusten testaustyökalu kehitetty toimimaan yhdessä Robot Frameworkin kanssa [9]. Appium simuloi sovelluksen käyttöä käyttämällä painikkeita yleensä joko elementin id:n tai xpathin avulla. Appiumin koodikieli on käytännössä samalla formaatilla Robot Frameworkin kanssa. Appium tarvitsee Javan toimiakseen. Appiumin palvelin avataan sovelluksen kautta ja se toimii keskusteluyhteytenä PyCharmin ja matkapuhelimen välillä.

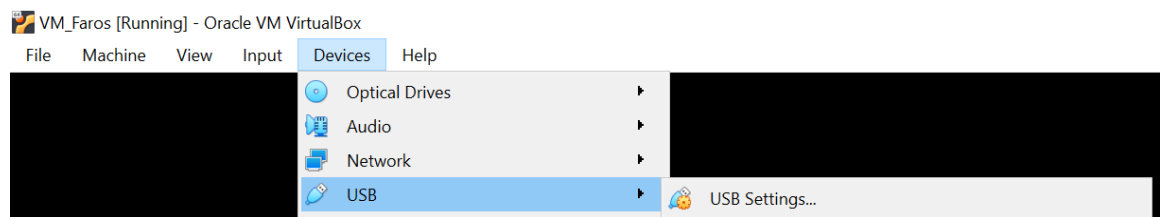
Appium-sovellus avaa visuaalisen kuvan puhelimesta, ja sen käyttöliittymästä näkee sovelluksen painikkeiden ja elementtien tiedot ja muuttujat [Kuva 5]. Testattavaa sovellusta pitää sitten itse manuaalisesti ohjata, jotta jokaisen vaiheen voi antaa testille.



Kuva 5: Appiumin käyttöliittymä esittämässä akkutaso-painiketta

Appiumin sovellus kaataa käyttöliittymäsessio, jos testin ajaa PyCharmista. Tämä vaikeuttaa nopeaa testaamista testin kehityksen lomassa, sillä testin ajettua täytyy sulkea kaatunut sessio ja avata uusi. Uuden session avaamisen jälkeen täytyy ongelmallinen kohta etsiä manuaalisesti uudestaan, jos Appiumin sovellus on ainoa pääsy elementtien nimiin kuten tässä opinnäytetyössä. Luonnollisesti palvelin kuitenkin pysyy päällä, joten testin ajon voi toistaa niin monta kertaa kuin on tarpeen.

Testipenkin ohjaamiseen käytettiin ohjelmistoa, joka toimi ainoastaan Linux-alustalla. Tätä varten jouduttiin lataamaan Oraclen Virtual Machine, sillä tietokoneyksiköitä oli ainoastaan yksi. Tiedostojen siirtäminen virtuaalikoneelle vaati jaetun kansion käyttöä. Windowsin jaettu kansio näkyy myös Linuxin puolella, ja tiedostot voi kansiosta kopioida virtuaalikoneen tiedostoihin kansiosta. Testipenkki vaati myös USB-yhteyden, joten virtuaalikoneelle piti erikseen valita käytettävä USB-portti [Kuva 6]. Mikäli virtuaalikoneen tai fyysisen koneen sammuttaa, tulee testipenkin ohjelmiston asennus ajaa uudestaan.



Kuva 6: Testipenkin USB:n valitseminen virtuaalikoneesta

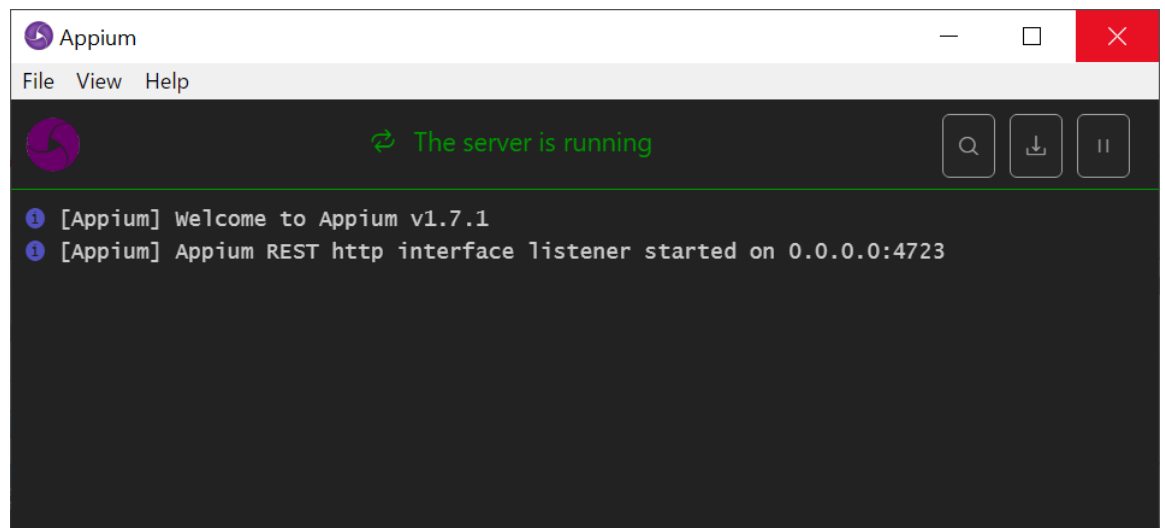
## 4 Testin kulku

Testi on monivaiheinen ja tässä luvussa käsitellään testin kulku vaihe vaiheelta alkuvalmistelusta lopetukseen. Kulku jakautuu karkeasti kolmeen vaiheeseen: testin valmistelu, sen kulku ja lopetusvaihe. Testin kulku ja lopetusvaihe ovat molemmat automatisoituja, mutta ne eroavat toisistaan rakenteellisesti ja lopetusvaihe lähettää pääasiassa kutsuja Bittium MedicalSuite -palvelimelle matkapuhelimen sijaan.

### 4.1 Testin ajoa edeltävät valmistelut

Ennen testin ajoa ympäristö tulee valmistella testin toimivuuden takaamiseksi. Ajoa edeltävä valmistelu jakaantuu kahteen osaan: matkapuhelin tarvitsee Appiumin toimintavalmiiksi ja Faros testipenkkiä ajavan ohjelmiston käyntiin. Ympäristö vaatii alustavaa valmistelua myös ennen testin ajoa edeltäviä valmisteluja, kuten ohjelmistojen asentaminen. Ne käydään läpi luvussa 3.

Testi käyttää Appiumin palvelinta kommunikoidakseen mobiilisovellukseen. Palvelin käynnistetään Appiumin antamalla vakioasetuksilla. Palvelimen ja puhelimen tiedot on ennalta kovakoodattu testiin ja Appiumin puolesta täytyy ainoastaan käynnistää serveri. [Kuva 7.]



Kuva 7: Appiumin palvelin käynnistettynä

Matkapuhelimen sijaan Faros tarvitsee sydänekäyräsignaalin toimiakseen testin aikana ja tähän käytetään testipenkkiä. Testipenkin ohjelmisto on Linux-pohjainen, joten sen käynnistäminen



riippuu laitekokoonpanosta. Tässä projektissa kaikki ohjelmistot ajettiin samalta tietokoneyksiköltä, joten testipenkin ohjelmisto ajetaan virtuaalikoneelta. Ohjelmisto siirrettiin virtuaalikoneelle Windowsin jaetun kansion avulla sitä pystyttäessä.

Käynnistäessä virtuaalikone uudestaan testipenkin ajurit pitää myös asentaa uudelleen. Ohjeet löytyvät kansion README-tiedostosta ja ajurit asennetaan yhdellä komennolla komentoriville. Ajurien asentamisen jälkeen asetetaan jännite sekä taajuus jokaiselle elektrodille. Viimeisenä asetetaan olemassa oleva sydänpäätiedostoa sisältävä tiedosto simuloimaan sydänpäätiedoston mittausta.

Kuvassa 8 näkyvät kaikki komentorivillä ajettavat komennot testipenkin ympäristön valmistelun vuoksi.

```

faros@faros-VirtualBox:~/Documents/Testipenkki/kernel$ sudo sh ./install.sh
[sudo] password for faros:
'Makefile' -> '/var/tmp/usb-skel/Makefile'
'usb-skeleton.c' -> '/var/tmp/usb-skel/usb-skeleton.c'
make -C /lib/modules/5.4.0-77-generic/build M=/var/tmp/usb-skel modules
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-77-generic'
  CC [M] /var/tmp/usb-skel/usb-skeleton.o
Building modules, stage 2.
MODPOST 1 modules
  CC [M] /var/tmp/usb-skel/usb-skeleton.mod.o
  LD [M] /var/tmp/usb-skel/usb-skeleton.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-77-generic'
faros@faros-VirtualBox:~/Documents/Testipenkki/kernel$ cd ..
faros@faros-VirtualBox:~/Documents/Testipenkki$ sudo ./test_kalif /dev/skel0 -k0r128
faros@faros-VirtualBox:~/Documents/Testipenkki$ sudo ./test_kalif /dev/skel0 -k1r128
faros@faros-VirtualBox:~/Documents/Testipenkki$ sudo ./test_kalif /dev/skel0 -k2r128

```

Kuva 8: Komentorivikuva kaikista tarvittavista komennosta virtuaalikoneen valmistelusta

Komento `sudo sh ./install.sh` asentaa ohjelmiston ajurit pääkäyttäjänä ohjelmistojen rajoituksien vuoksi. Usein `sudo` pyytää käyttäjän salasanan ensimmäisen kerran ajettaessa. Komentoikkuna tulostaa ruudulle informaatiota asennuksesta hetken ennen komennon `cd ..` ajamista. Komento `cd ..` palauttaa sijainnin yhden kansion aiemmin kansiohierarkiassa.

Kolme viimeistä komentoa komentorivillä asettaa taajuudet testipenkin kolmelle kanavalle. Testipenkkiä ohjaa `test_kalif`-ohjelma, ja sen sisältämä `skel`-komento määrittelee käytettävän tietokoneyksikön USB-portin. Tässä työssä käytettävä USB-portti oli numerolla 0, ja portin numero on helppo tarkistaa luvussa 3.4 esitellystä tavasta valita USB-portti testipenkille virtuaalikoneesta. Laitteiden kokoonpanoa muuttaessa on tärkeää tarkistaa oikeanlaisten komentojen käyttö.

Lopuksi komentoikkunasta ajetaan valmis sydänpäätiedosto README-tiedostosta löytyvällä komennolla. Tämä komento ei ole näkyvissä yllä olevassa kuvassa, vaan komento ajetaan näiden komentojen jälkeen. Testin kehityksessä on käytetty README-tiedostossa mainittua sydänpäätiedostoa.

## 4.2 Testin aloitus

Testi käynnistetään ajamalla se PyCharmin konsoli-ikkunasta. Komentona käytetään `./[testin kansio]/AutomationTest.robot`, joka käynnistää testin ajon. Kun komento on ajettu, on hyvä painaa Faroksen virtapainiketta, jotta varmistetaan sen olevan päällä ja aktiivisena. Tästä eteenpäin testi on täysin automatisoitu ja vaatii manuaalista työtä ainoastaan vikatilanteissa.

Käynnistyessään Appium tarkistaa käyttöoikeutensa puhelimen asetuksista. Tämä näkyy visuaalisesti asetukset-ruudun välähdyksenä näytöllä. Appiumin tarkistettua asetukset itse testi voi alkaa.

Sovelluksen käynnistyessä ensimmäisellä ajokerralla se pyytää hyväksymään käyttöehtoja. Käyttöehtojen hyväksyminen jää sovelluksen välimuistiin, joten käyttöehdot kysyvä ponnahdusikkuna ei ilmesty seuraavissa ajoissa. Appium kuitenkin pyyhkii Bittium MedicalSuite Mobile -sovelluksen välimuistin, mikäli sille avataan sessio Appiumin sovelluksessa tietokoneella. Tällöin Bittium MedicalSuite Mobile -sovellus kysyy käyttöoikeudet uudestaan, kun testi ajetaan. Testi on suunniteltu hyväksymään käyttöehdot, mikäli niitä pyydetään; mutta ei kaada ajoa, jos ikkuna ei koskaan ilmesty.

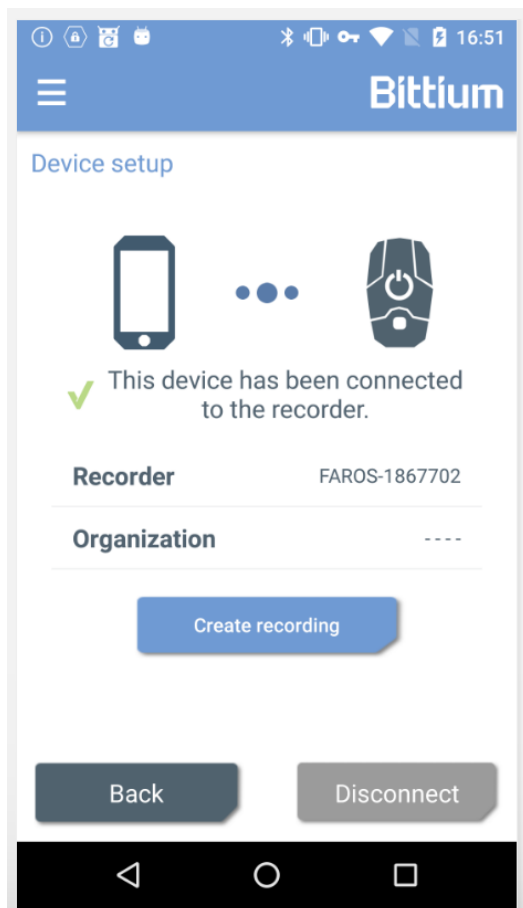
Testin alussa tarkistetaan matkapuhelimen akun taso. Mikäli matkapuhelimen akun taso on alle 30 %, testi keskeyttää kaiken ajon. Testi tallentaa akun tason lokiin ja tulostaa sen tuloksiin. Olisi eduksi tarkistaa myös Faroksen akku, sillä se ei lataudu testaamisen aikana. Ohjelmistossa on kuitenkin puuttuva ominaisuus, ja Faroksen akun tila näkyy vasta testin aikana. Tästä syystä Faroksen akun tila tarkistetaan ja tulostetaan vasta testin päätyttyä.

Ennen mittauksen luomista testi yhdistää Faroksen Bittium MedicalSuite Mobile -sovellukseen. Faroksen sarjanumero on kovakoodattu testiin, jotta testi ottaa käyttöön valitun laitteen. Laite on helppo vaihtaa: testin tiedostosta löytyy kovakoodattu kohta käytetyn laitteen sarjanumerolle ja kohta on kommentoitu selkeästi. Muuttamalla arvot toisen laitteen sarjanumeroon käytössä oleva laite on vaihdettu. Alla esitelty *Keywords.robot*-tiedostosta löytyvä Faroksen testin kehityksessä käytetty sarjanumero. [Kuva 9.]

```
# Change this if you use a different faros
${FarosID}          FAROS-1867702
${dev_id}           1867702
```

Kuva 9: Faroksen sarjanumero koodissa

Faros yhdistyy Bittium MedicalSuite Mobile -sovellukseen Bluetoothin avulla ja yhteys vaatii ajoittain aikaa muodostuakseen [Kuva 10]. Testiin on tästä syystä asetettu ylimääräistä aikaa ennen kuin se aikakatkaisee yhdistämisyrityksen. Faroksen onnistuneen yhdistämisen jälkeen testi aloittaa mittauksen luomisen.



Kuva 10: Sovellus on yhdistänyt Faroksen ylläpitonäkymässä

#### 4.3 Mittauksen luominen ja sen kulku

Kun testi on palannut takaisin sovelluksen kotinäkömään yhdistettyään Faroksen, mittaus on valmis luotavaksi. Testi palaa ylläpitonäkymään painamaan "luo mittaus"-painiketta [Kuva 10]. Mittausta varten tulee täyttää potilastiedot, jotka ilmestyvät matkapuhelimen ruudulle [Kuva 11].

**Bittium**

Create recording

1 — 2 — 3

Patient details\* Anamneses Recording details\*

Patient ID

\_\_\_\_\_

Patient ID must be at least 4 characters long and lower/uppercase letters, numbers and underscores are allowed. If Patient ID is left blank the system will generate it (for example PID0000A).

Age (years)\*

\_\_\_\_\_

Weight (kg)\*

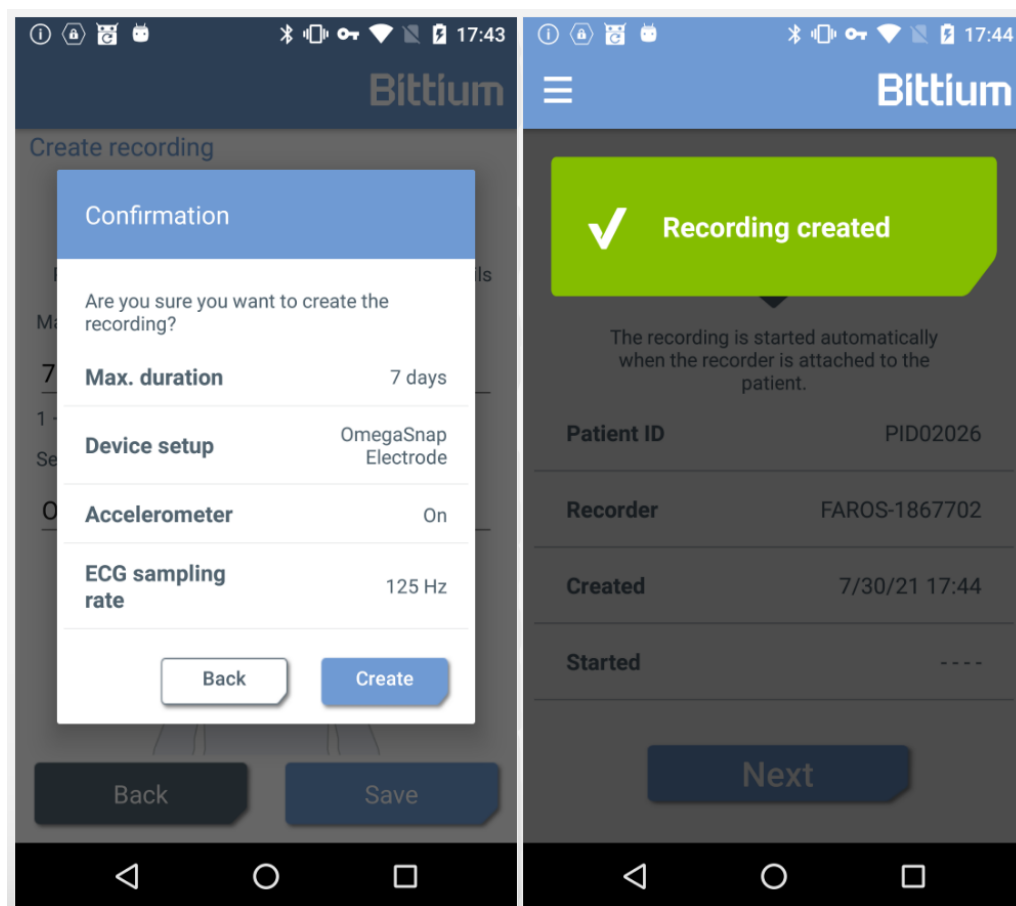
\_\_\_\_\_

Back Next

Kuva 11: Potilastietojen näkymä ensimmäisellä sivulla

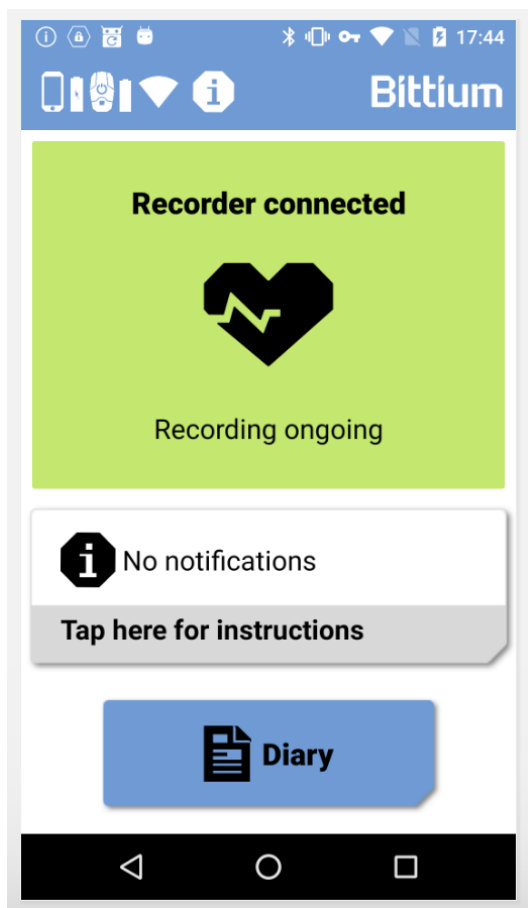
Lähes kaikkiin valintoihin avautuu rullavalikko, jota on haastava ohjata hallitusti Appiumin komennoilla. Haastetta lisää testin mahdollinen käyttö eri mallisilla matkapuhelimilla. Testi myös laajeni huomattavasti rullavalikoiden ohjailmiseen tarvittavilla avainsanoilla. Tästä syystä testi käyttää jokaisessa kohdassa oletusasetuksia, eli se hyväksyy rullavalikosta automaattisesti aukeavan arvon.

Testi ohittaa toisen sivun täyttämisen kokonaan sen ollessa vapaaehtoinen mittauksen toimimiselle. Tästä syystä testi ohjaa sovelluksen suoraan kolmannelle sivulle painamalla yläreunassa näkyvää kolmosta kahden hyväksymispainikkeen sijaan. Suoraan kolmannelle sivulle siirtyminen säästää testistä toistuvan avainsanan. Tämä optimoi testiä, sillä jokainen avainsana lisää siihen helposti useamman sekunnin ajoaikaa, mikä kasaantuu nopeasti ylimääräistä avainsanoista. Testin täytettyä valinnat mittauksen luominen vahvistetaan ja se voi alkaa [Kuva 12].



Kuva 12: Mittauksen valinnat asetettuaan, mittaus luodaan

Sovelluksella voi toisinaan mennä hetki Faroksen sydänpäätalven löytämiseen, joten testiin on lisätty ylimääräistä aikaa ennen aikakatkaisua. Tästä syystä on hyvä varmistaa, että Faros ja matkapuhelin ovat Bluetooth-kantaman etäisyydellä toisistaan. Kun sydänpäätalven on löytynyt ja laitteet kommunikoivat keskenään, mahdollisuus kirjata päiväkirjaan ilmestyy kotinäkömään [Kuva 13].

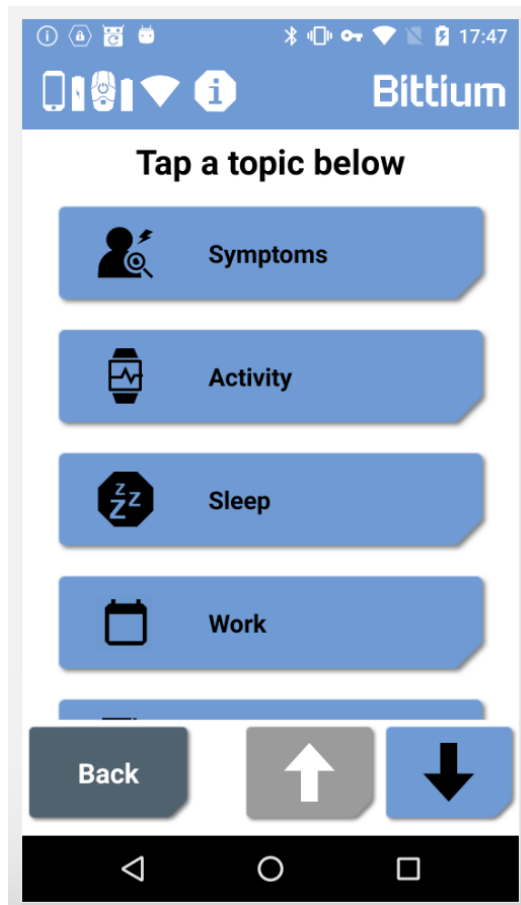


Kuva 13: Sovellus on löytänyt Faroksen sydänpääläpän

Mittauksen aikana potilas voi kirjata ylös oireitaan ja aktiviteettejaan sovelluksen päiväkirjaan. Testi kirjaa ylös kohdan jokaisesta mahdollisesta neljästä vaihtoehdosta. [Kuva 14.] Oireista sovellus kirjaa huimauksen ja kaikki muut *sleep*-avainsanalla pidemmät ajat nukkuessa tai töissä.

Testi täyttää jokaisen kohdan kerran ja järjestyksessä: oireet, aktiivisuus, nukkuminen ja työt. Koodiin on laitettu pitkät odotusajat kolmen jälkimmäisen kohdalla, sillä se vastaa paremmin käyttökokemusta. Nämä on selkeästi kommentoitu testin koodiin, joten odotusajan *sleep*-komentoa on helppo editoida tai muuttaa, jos sen pituutta halutaan muokata tai testi vaatii manuaalista jatkokehitystä.

Lopuksi testi tarkistaa, että kaikki kirjaukset löytyvät sovelluksen mittauksen lokista vertaamalla niitä ruudulla oleviin avainsanoihin. Aktiviteetit kirjattuaan se lopettaa mittauksen.



Kuva 14: Päiväkirjan kirjausvaihtoehdot

#### 4.4 Testin lopetus

Kirjattuaan ja tarkastettuaan neljä päiväkirjamerkintää testi on valmis siirtymään lopetusvaiheeseen. Mittausta ei voi sulkea Bittium MedicalSuite Mobile -sovelluksen käyttöliittymästä, joten testi lähettää usean peräkkäisen API-kutsuja Bittium MedicalSuite -palveluun. Testi luo session palvelimelle, jotta lähetetyt API-kutsut toimivat yhdessä askeleittain. Kaikki API-kutsut löytyvät tiedosta *API\_library.py*.

Ensimmäinen API-kutsu on kirjautumispyyntö *post*-kutsukomennolla Bittium MedicalSuite -palveluun. Kutsu käyttää kovakoodattuja kirjautumistietoja, joten ne tulee vaihtaa, mikäli testiin tai käyttäjätunnuksiin tekee muutoksia. Palvelin vastaa kutsuun koodilla 200, "OK", mikäli sisäänkirjautuminen onnistui. Testi tulostaa tiedon onnistumisesta lokiin, jotta vikatilanteita on helpompi valvoa.

Kirjauduttaan sisään testi etsii palvelusta oikean mittauksen. Tähän se käyttää aiemmin mittauksesta talteen otettua potilas-ID:tä ja testin luomaa sessiota. Testi lähettää *get*-pyynnön näillä tiedoilla saadakseen takaisin json-tiedoston ja parsiakseen sieltä aktiivisena olevan mittauksen ID:n. Palvelin vastaa myös tähän kutsuun koodilla 200, joka tulostetaan lokiin.

Measurement ID:n saatuaan testi lähettää *put*-pyynnön mittauksen lopettamiseksi. Onnistuneeseen pyyntöön palvelu vastaa koodilla 202, "Accepted", ja sulkee mittauksen. Sulkeminen näkyy myös Bittium MedicalSuite Mobile -sovelluksen käyttöliittymässä ja vaatii testiä painamaan vielä muutamaa painiketta sovelluksen puolella. Testi odottaa Faroksen tietojen tallennuksen ja käy sen jälkeen tarkistamassa sekä kirjaamassa ylös sen akkutason. Lopuksi testi lähettää viimeisen API-pyyntöön palvelimelle Faroksen irrottamiseksi mittauksesta. Palvelin vastaa koodilla 200 ja tulostaa tiedon onnistuneesta irrotuksesta lokiin.



## 5 Päätäntö

Opinnäytetyön sisältö oli minulle lähes täysin uutta. Työn tekeminen oli hyppy tuntemattomaan, mutta opin asiat nopeasti projektista saatavalla avustuksella. Minulle oli uutta Robot Frameworkin ja Appiumin käyttäminen, automaatiotestaaminen, osa virtuaalikoneen toiminnoista, PyCharmin ohjelmointiympäristö sekä mobiilialustan kehittäminen.

Projekti oli hyvä oppimiskokemus. Opin paljon asioita ohjelmistotestaamisesta, erityisesti mobiililla ja automaatiotestaamisen eroista eri alustoilla. Appiumin tunnetumpi sisarkirjasto, Selenium, on käytössä PC-ympäristössä. Havaintona isoin ero näiden kahden käytössä omilla alustoillaan osoitti mobiililla testaamisen hitauden. Siinä missä Selenium tarvitsee sleep-komentoja, jotta nettisivu tai sovellus pysyy sen perässä, Appiumin testi vaatii pidennettyjä timeout-komentoja, sillä testin elementtejä etsivät komennot eivät kerenneet löytämään tarvitsemaansa painiketta tarpeeksi nopeasti.

Opinnäytetyön tekeminen opetti myös paljon aikatauluttamista. Suurien ja isotöisten projektien tekeminen on aina ollut minulle haaste, ja opinnäytetyö ei poikennut tästä. Projektin tekeminen oli kuitenkin tärkeä aikatauluttaa, jotta sen saa tehtyä järkevän ajan puitteissa. Työn toteutus ei vastannut täydellisesti sen suunnitelmaa, mutta se opetti silti pitkäjänteiseen työhön.

Kokonaisuudessaan opinnäytetyö onnistui juuri kuten pitikin. Se tulee osaksi projektia ja kaikki vaatimukset täyttyivät. Suurin hyöty opinnäytetyön tekemisestä tuli projektille. Automaatiotesti tuli osaksi tuotetestausta ja jätti erinomaisen pohjan testauksen jatkokehitykselle.

### 5.1 Haasteet työn toteutuksessa

Työ haastoi minua alusta lähtien ollessaan suurimmaksi osaksi vierasta asiaa ja se oli kokonaisuudessaan haastavampi kuin mihin olin varautunut. Opinnäytetyö ei sisältänyt kovin montaa entuudestaan tuttua elementtiä, ja matkan varrella oli useampi tietotekninen itse työhön liittymätön ongelma. Virtuaalikoneen asennus ja valmistelut tuottivat vaikeuksia, sillä en ollut aiemmin kiinnittänyt fyysisiä laitteita siihen tai jakanut tiedostoja Windowsin ja Linuxin virtuaalikoneen välillä, kun testipenkin ohjelmisto tuli siirtää sinne. Ainoastaan muutama opinnäytetyössä käytetyistä työkaluista oli tuttu etukäteen. Myös se kuvaa opinnäytetyön onnistumista, kun vieraista lähtökohdista pystyy tuottamaan onnistuneen projektin.

Erityisesti virtuaalikoneympäristön pystyttäminen oli haastavaa. Tarkkaa käyttöjärjestelmän versiota ei ollut kirjattu dokumentaatioon ylös, joten käytin uusinta mahdollista versiota 64-bittisestä Linux Ubuntusta. Tämä toi haasteita testipenkin ohjelmiston käytössä. Ohjelma *test\_kalif*, joka ajaa testipenkkiä, oli lukittu käyttöoikeuksien taakse. Konsolissa ajaminen antaa myös saman virheilmoituksen, jos testipenkki ei ole kiinni laitteessa, josta komentoa ajetaan.

Ajatus Robot Frameworkin ja Appiumin käytöstä työkaluina lähes koko opinnäytetyön tekemiseen tuntui aluksi liian isolta palalta oppia kerralla, mutta molemmat olivat intuitiivisia ja käyttäjävälillä ottaa haltuun. Robot Frameworkilla on omat haasteensa, erityisesti ehdollisten steppien tekeminen testiin on tarpeettoman haastavaa, mutta niiden puute tekee testistä hyvin haavoittuvaisen ja herkän.

Mobiilisovelluksen kehityksessä ei ollut huomioitu automaatiotestaamisen haasteita, joten myös hyvin suunnitellusta testistä tuli lopulta herkkä. Osa mobiilisovelluksen elementeistä oli valittu välittämättä automatisoidusta käytöstä, joten joustava testisuunnittelu on hankalaa. Jatkokehitysideoita osaan haasteista ja havainnoista käsitellään seuraavassa luvussa.

Testausautomaatioympäristön suunnitelmallisuuden puute näkyi muun muassa elementtien nimeämisessä sovelluksessa. Osa painikkeista oli nimetty camel casella (EsimerkkiSana) ja osa snake casella (esimerkki\_sana), mikä hankaloitti testin pitämisen mahdollisimman kompaktina ja joustavana muutoksille. Takaisinpaluuta varten luotu avainsana joutui pitämään sisällään kaksi eri versiota, jotta jokainen kohta testistä toimii oikein.

Appium kaataa käyttöliittymäsessio, jos testin ajaa PyCharmista. Tämä vaikeuttaa nopeaa testaamista, sillä testin ajettua täytyy sulkea kaatunut sessio ja avata uusi. Uuden session avaamisen jälkeen täytyy ongelmallinen kohta etsiä manuaalisesti uudestaan. Luonnollisesti palvelin pysyy kuitenkin päällä, joten testiä voi ajella niin monta kertaa kuin huvittaa.

## 5.2 Testin ja sen ympäristön jatkokehitys

Projekti onnistui hyvin ja täytti kaikki vaatimukset, mutta jätti silti tilaa jatkokehitykselle. Osa jatkokehitysideoista koskee itse testiä ja osa testin ulkopuolista ympäristöä.

Tällä hetkellä testi tekee kaiken vaaditun, mutta jatkokehitystä varten testiä kannattaa laajentaa automaatiotestauksen monipuolistamiseksi. Testiin voi tuoda lisäsisältöä ja vakautta vikatilanteiden varalta.

Eniten lisäsisältöä jatkokehitykseen tuo potilastietojen valinta. Tällä hetkellä testi valitsee oletusvaihtoehdon, sillä rullavalikkoa on monimutkaista siirtää hallitusti haluttuun arvoon. Tästä syystä automaatiotesti ei saa testattua tätä ominaisuutta sovelluksesta, vaan se kuuluu edelleen manuaalitestaukseen. Mahdollinen hyvä tapa olisi toteuttaa avainsana, joka siirtää rullavalikkoa yhden yksikön tai avainsana, jonka muuttujaa kertomalla saa laskettua helposti tarvittaman siirtomäärän. Tähän voisi toimia esimerkiksi Swipe-avainsana, jota käytetään rullaamaan potilastietojen upotettua rullausikkunaa, sillä ominaisuus ei toimi muiden ruutua rullaavien avainsanojen kanssa upotuksen vuoksi.

Toteutus kannattaa suunnitella mahdollisimman joustavaksi muokata, jotta tietoja on helppo vaihtaa tarvittaessa. Yksi avainsana nostaa testin ajoa useammalla sekunnilla, mikä kasaantuu nopeasti pidemmäksi ajaksi, joten monimutkaisemman implementaation vuoksi testin ajo on hyvä suunnitella yöllä ajettavaksi.

Näin laaja jatkokehitys kuitenkin herkistää testiä huomattavasti. Yhden avainsanan epäonnistuminen johtaa lähes kaikkien seuraavienkin epäonnistumiseen, sillä jokainen avainsana olettaa olevansa oikeassa näkymässä elementtien löytymiseksi. Testiä olisi hyvä vahvistaa virhetilanteiden varalta tai muokata loppusiivousta niin, että testi lopetetaan ja poistetaan virhetilanteista huolimatta.

Loppusiivouksessa API-kutsut lähetetään ainoastaan mittauksille, joiden tila on *active*. Erittäin hyvä lisäys tähän olisi lisätä ehdollinen API-kutsu myös mittauksille tilassa *created*, jos testin kulku epäonnistuu ennen mittauksen käynnistämistä sen luomisen jälkeen. Tällöin testiä ei tarvitsisi manuaalisesti sulkea virhetilanteissa, vaan sekin hoituisi automaattisesti ja testin voisi vain käynnistää uudestaan.

Mikäli myös Bittium MedicalSuite Mobile -sovellusta halutaan parantaa, valittaisiin yksi nimeämistapa ja noudatettaisiin sitä. Tämä vähentäisi useampien samankaltaisten avainsanojen luontia sekä tekisi olemassa olevista yhteensopivampia. Tämä mahdollisesti helpottaisi myös testin kehittämistä, jotta jatkossa voisi olettaa yleisimpien painikkeiden nimiä helpommin manuaalisen tarkistamisen sijaan.

Mittauksen lopettava API-kutsu saa Bittium MedicalSuite -palvelusta vastauksen 202, "Accepted". Dokumentaation mukaan kutsun tulisi vastata 200, "OK", kuten kahden muunkin vastauksellisen kutsun. Ideaalisesti vastaus muutettaisiin 200:ksi vastaamaan dokumentaatiota ja olemaan yhtäläinen kahden muun kutsun kanssa.

## Lähteet

- 1 Bittium Wireless Oy kotisivut, Company Overview  
Saatavilla: <https://www.bittium.com/about-bittium/facts-figures/company-overview>
- 2 Ohjelmistotestauksen käsikirja, Jussi Pekka Kasurinen  
ISBN: 978-952-291-102-5 EPUB
- 3 Khadija Tahera, David C. Wynn, Chris Earl, Claudia M. Eckert: Research in Engineering Design, Testing in the incremental design and development of complex products  
DOI: 10.1007/s00163-018-0295-6
- 4 Terveyskirjasto, Terveysthuollonerikoislääkäri Hannaleena Eerola: Sydänsairauksia, joissa EKG:sta on hyötyä  
Artikkelin tunnus: snk03211  
Saatavilla: <https://www.terveyskirjasto.fi/snk03211/sydansairauksia-joissa-ekgsta-on-hyotya>
- 5 Terveyskirjasto, Terveysthuollonerikoislääkäri Hannaleena Eerola: EKG (sydänfilmi)  
Artikkelin tunnus: snk03210  
Saatavilla: <https://www.terveyskirjasto.fi/snk03210/ekg-sydanfilmi>
- 6 Bittium Wireless Oy kotisivut, Bittium Faros  
Saatavilla: <https://www.bittium.com/medical/bittium-faros>
- 7 Bittium Wireless Oy kotisivut, Bittium Tough Mobile  
Saatavilla: <https://www.bittium.com/secure-communications-connectivity/bittium-tough-mobile>
- 8 Robot Framework  
Saatavilla: <https://robotframework.org/>
- 9 Appium Library, Appiumin kirjasto, dokumentaatio  
Saatavilla: <https://serhatbolsu.github.io/robotframework-appiumlibrary/Appium-Library.html#library-documentation-top>

Liitteet