# Peltihamsteri-sovellusprojekti

**Mari Kasanen**
**Leevi Liimatainen**
**Marina Mustonen**
**Juhani Sundell**
**Arttu Ylä-Sahra**

# Sovellusraportti

**Versio 0.1.0**

**Julkinen**

**6.6.2019**

**Jyväskylän yliopisto**

**Informaatioteknologian tiedekunta**

| **Hyväksyjä** | **Päivämäärä** | **Allekirjoitus** | **Nimenselvennys** |
|---|---|---|---|
| Projektipäällikkö | __.__.20__ | | |
| Tilaaja | __.__.20__ | | |
| Ohjaaja | __.__.20__ | | |

# Tietoja dokumentista

Tekijät:

Marina Mustonen marina.s.mustonen@gmail.com 044-9733582

Juhani Sundell juhani.k.sundell@student.jyu.fi 044-2666773

**Dokumentin nimi:** Peltihamsteri-projekti, Sovellusraportti

**Sivumäärä**: 34

**Tiedosto:** Peltihamsteri_Sovellusraportti_0.1.0.pdf

**Tiivistelmä:** Peltihamsteri-sovellusprojekti kehitti Syncster-sovelluksen Jyväskylän yliopiston kognitiotieteen ajosimulaatiolaboratoriossa kerättävän datan synkronointiin ja hallintaan. Sovellusraportti kuvaa Syncster-sovelluksen taustaa, käyttöliittymää ja rakennetta. Raportissa kuvataan myös projektissa kehitettyä toista, Android-laitteilla toimivaa syötetietojen keräämiseen tarkoitettua sovellusta. Raportissa käydään myös läpi ohjelmointi- ja testauskäytänteet. Lisäksi raportissa kuvataan sovellusten vaatimusten täyttämistä ja annetaan ohjeita jatkokehitystä varten.

**Avainsanat:** C#, jatkokehitys, käyttöliittymä, ohjelmointikäytänteet, sovellusrakenne, tavoitteet, testaus, vaatimukset, WPF.

## Muutoshistoria

| Versio | Päivä | Muutokset | Tekijä |
|---|---|---|---|
| 0.0.1 | 25.4.2019 | Sovellusraportin laatiminen aloitettiin. | Marina Mustonen |
| 0.0.2 | 30.4.2019 | Hiottiin raportin runkoa, johdantoa, taustaa ja tavoitteita sekä yhteenvetoa | Marina Mustonen |
| 0.0.3 | 22.5.2019 | Käyttöliittymän näkymiä lisätty. | Juhani Sundell |
| 0.0.4 | 27.5.2019 | Käyttöliittymän kuvat kaikista näkymistä. | Juhani Sundell |
| 0.0.5 | 3.6.2019 | Sovelluksen toiminta ja rakenne –lukua täydennetty. | Juhani Sundell |
| 0.0.6 | 4.6.2019 | Tavoitteiden toteutumista ja jatkokehitysideoita täydennetty. | Juhani Sundell |
| 0.0.7 | 5.6.2019 | Touchsterin osio täydennetty, jatkokehitysideoita ja yhteenvetoa täydennetty. Korjattu kirjoitus- ja muotoiluvirheitä. | Juhani Sundell, Mari Kasanen, Leevi Liimatainen |
| 0.1.0 | 6.6.2019 | Viimeistelty kirjoitusasua ja karsittu ylimääräistä selittämistä. | Juhani Sundell, Leevi Liimatainen |

## Tietoja projektista

**Tekijät:**

| | | |
|---|---|---|
| Mari Kasanen | r.mari.s.kasanen@student.jyu.fi | 044-2359811 |
| Leevi Liimatainen | leevi.m.m.liimatainen@student.jyu.fi | 050-5052731 |
| Marina Mustonen | marina.s.mustonen@gmail.com | 044-9733582 |
| Juhani Sundell | juhani.k.sundell@student.jyu.fi | 044-2666773 |
| Arttu Ylä-Sahra | arttu.e.yla-sahra@student.jyu.fi | 044-0668949 |

**Tilaajan edustajat:**

| | | |
|---|---|---|
| Hilkka Grahn | hilkka.grahn@jyu.fi | 040-8053342 |
| Tuomo Kujala | tuomo.kujala@jyu.fi | 0400-247392 |

**Ohjaajat:**

| | | |
|---|---|---|
| Jonne Itkonen | jonne.itkonen@jyu.fi | 050-4432381 |
| Jukka-Pekka Santanen | santanen@jyu.fi | 040-8053299 |

**Yhteystiedot:**

| | |
|---|---|
| Sähköpostilistat | peltihamsteri@korppi.fi, peltihamsteri_opetus@korppi.fi |
| Sähköpostiarkistot | https://korppi.jyu.fi/kotka/servlet/list-archive/peltihamsteri/, https://korppi.jyu.fi/kotka/servlet/list-archive/peltihamsteri_opetus/ |

# Sisältö

# 1 Johdanto

Peltihamsteri-projekti kehitti kevään 2019 Sovellusprojekti-kurssilla Jyväskylän yliopiston kognitiotieteen ajosimulaatiolaboratoriolle sovelluksen tutkimuksissa kerättävän datan synkronointiin ja hallintaan. Ajolaboratoriossa kehitetään testausmenetelmiä teollisuuden ajoneuvokäyttöliittymiä varten, sekä tehdään perustutkimusta kuljettajien tarkkaavaisuudesta, visuaalisesta havainnoinnista ja liikennekäyttäytymisestä. Ajosimulaatiokokeessa saadaan dataa useasta eri laitteesta. Projektiryhmä kehitti sovelluksen, jolla näistä eri laitteista kerättävä data saadaan ajallisesti synkronoitua keskenään sen käsittelyn ja analysoinnin helpottamiseksi.

Sovellusraportissa kuvataan sovelluksen toteutusratkaisuja, suunnitelmien ja vaatimusten täyttymistä sekä sovelluksen puutteita ja jatkokehitysideoita. Tarkemmat vaatimukset sovellukselle on käyty läpi vaatimusmäärittelyssä [1]. Projektiraportti [2] kuvaa miten projekti toteutui tavoitteiden ja käytänteiden suhteen. Rakennekuvauksessa [3] kuvataan sovelluksen rakennetta ja toimintoja eri näkymien kautta. Järjestelmätestaussuunnitelmassa [4] kuvataan ohjelmistolle suoritettuja testejä ja järjestelmätestausraporteissa [5][14][15] testauksen tuloksia. Dokumentin laatimisessa käytettiin apuna sovellusprojektien ohjetta [6] ja Monisiro- ja Isäxi-projektien sovellusraportteja [7][8].

# 2 Termit

Luvussa kuvataan Peltihamsteri-projektiin liittyviä termejä. Dokumentissa esiintyvät aihealueen termit ovat seuraavat:

| | |
|---|---|
| **AOI** | eli Area of Interest on silmänliikekameran ohjelmistoon määritettävät alueet, joista mitataan kuinka usein ja kuinka kauan koehenkilön katse kohdistuu niihin. |
| **Asetustiedosto** | (eng. *Study settings*) on tiedosto, johon tallennetaan kokeen asetukset, jolloin samoja asetuksia voi käyttää useamman eri koehenkilön kanssa. |
| **CSV** | eli Comma-Separated Values on tiedostomuoto, jolla tallennetaan yksinkertaista taulukkomuotoista tietoa tekstitiedostoon. |
| **Datan jäsentäminen** | on arvojen erottelua ja tunnistamista raakadatasta. |
| **Kanoninen aikaleima** | on jokaiselle datariville laskettava aikaleima, jonka perusteella data synkronoidaan tulostiedostoon. |
| **Koe** | (eng. *Study*) on yhden koehenkilön kanssa suoritettava 1—2 h kestävä ajosimulaatiokerta, jossa koehenkilö suorittaa tehtäviä ja suorituksesta kerätään dataa. |
| **Laitemoduuli** | vastaanottaa ja jäsentää dataa tietyltä laitteelta tai sen ohjelmistolta. |

| | |
|---|---|
| **Lua-skripti** | on tilaajan käyttämä kooditiedosto ajosimulaattoridatan mittaustulosten valikointiin. |
| **Sticky key** | on muuttuja, joka saa arvon tulostiedoston jokaisella rivillä. Arvo on sama kuin ylemmällä rivillä, jos laitteen lähettämässä datassa ei tapahdu muutosta. |
| **Sydämensyke** | (eng. Heartbeat) on yksittäisen laitemoduulin vastaanottama datamäärä viimeisen puolen sekunnin ajalta. |
| **Tehtävä** | (eng. *Task*) on kokeen aikana koehenkilön suorittama yksi ajosuorite. |
| **Toissijainen tehtävä** | on tehtävän aikana koehenkilön suorittama muu toiminto kuin ajaminen, esimerkiksi sovelluksen käyttäminen testilaitteella. |
| **Tulostiedosto** | on kehitettävän sovelluksen muodostama CSV-tiedosto, johon on yhdistetty ja synkronoitu valituilta laitteilta saatavat valitut dataosiot yhden koehenkilön yhdestä tehtävästä. |
| **Välitallennus** | on Syncsterin oma tallennusformaatti, jossa laitekohtainen data on tallennettu omiin tiedostoihinsa. |

Dokumentissa esiintyvät kohdealueen laitteiden ja ohjelmistojen termit ovat seuraavat:

| | |
|---|---|
| **Ajosimulaattori** | on laite, jolla simuloidaan oikean ajoneuvon ajamista virtuaalitodellisuudessa. |
| **Android** | on Linux-pohjainen käyttöjärjestelmä, jota käytetään yleisimmin älypuhelimissa ja tableteissa. |
| **D-Lab** | on tilaajan käyttämän silmänliikekameran ohjelmisto. |
| **DSI-Streamer** | on tilaajan käyttämän EEG-laitteen ohjelmisto. |
| **EEG** | eli elektroenkefalografia tarkoittaa aivosähkökäyrää. |
| **EepSoft** | on tilaajan käyttämän ajosimulaattorin ohjelmisto. |
| **QStates** | on EEG-laitteen kanssa käytettävä ohjelmisto, jolla laskettavia tunnuslukuja käytetään muun muassa koehenkilön kuormituksen arvioimiseen. |
| **Silmänliikekamera** | on laite silmän aseman ja liikkeiden mittaamiseen. |

Dokumentissa esiintyvät kehitysvälineisiin ja -tekniikoihin liittyvät termit ovat seuraavat:

| | |
|---|---|
| **C#** | on Microsoftin .NET-alustalle kehitetty ohjelmointikieli. |
| **Hyväksymistestaus** | on tilaajien suorittama testaus sovellukselle, jolla varmistetaan sovelluksen täyttävän tilaajien asettamat vaatimukset. |
| **Järjestelmätestaus** | on projektiryhmän suorittama testaus, jossa etsitään virheitä ja puutteita sovelluksen oikeaa käyttöä mallintavassa tilanteessa. |
| **Katselmointi** | on silmäkkäin tapahtuva tulosten tarkastaminen, jossa tarkastetaan toteutukset ja niiden erot alkuperäiseen suunnitelmaan, sekä esitetään parannusehdotuksia. |
| **Käyttöliittymä** | on ohjelman ihmiselle näkyvä osa, jonka avulla käyttäjä ja ohjelma ovat vuorovaikutuksessa keskenään. |
| **Lähdekoodi** | on tietokoneohjelman tekstimuotoinen ohjelmointikielinen listaus. |
| **WPF** | eli Windows Presentation Foundation on .NET-kirjasto, joka muodostaa rajapinnan ja toimii käyttöliittymän pohjana. |
| **Xamarin** | on natiivisovellusten kehittämiseen tarkoitettu kehitystyökalu, jolla koodin saa helposti jaettua eri käyttöjärjestelmien välillä. |

# 3 Taustaa ja tavoitteet

Projektin tilaajana toimi Jyväskylän yliopiston kognitiotieteen ajosimulaatiolaboratorio. Ajosimulaatiolaboratoriossa on vuosina 2007–2019 tehty kokeellisia tutkimuksia, joissa on keskitytty mittaamaan kuljettajan tarkkaavaisuutta ja liikennekäyttäytymistä, sekä esimerkiksi teollisuuden ajoneuvokäyttöliittymien ja muiden laitteiden käytön vaikutusta niihin. Kokeissa kerätään dataa tyypillisesti vähintään kolmelta eri laitteelta (yleensä ajosimulaattori, silmänliikekamera ja toissijaisen tehtävän laite). Myös EEG-laitteelta on tarkoitus kerätä aivosähkökäyrä-dataa.

Datan käsittelyt ja analyysit ovat aiemmin sisältäneet paljon virhealtista manuaalista työtä. Ajosimulaattorista dataa kerätään EepSoft-ohjelmiston ja silmänliikekamerassa D-Lab-ohjelmiston kautta. EEG-laitteella on käytössä kaksi eri ohjelmistoa, sen oma ohjelmisto DSI-Streamer sekä koehenkilön kuormituksen arviointiin tarkoitettu QStates.

Kehitetty Syncster-sovellus vähentää laitteilta saatavan datan yhdistämiseen ja synkronointiin liittyvää manuaalista työtä, mutta ei korvaa laitteiden omia ohjelmistoja. Syncster-sovelluksen päävaatimukset olivat, että sillä voidaan synkronoida dataa ajosimulaattorista, silmänliikekamerasta ja EEG-laitteesta. Lisäksi projektin aikana kehitettiin Android-laitteille sovellus, jolla saadaan kerättyä syötetietoja koehenkilön kokeen aikana käyttämästä laitteesta. Myös syötetiedot synkronoidaan muun datan kanssa. Muu data voidaan kerätä reaaliajassa, mutta silmänliikekameran AOI-data, ts. milloin koehenkilön katse on kohdistunut AOI:ksi määriteltyyn kohteeseen, synkronoidaan jälkikäteen.

Sovelluksen vaatimuksiin kuului myös, että sillä on helppokäyttöinen käyttöliittymä, jonka kautta voi valikoida halutut laitteet ja niiden dataosiot, täyttää kokeen ja tallennuksen tiedot, aloittaa ja lopettaa tallennuksen sekä yhdistää datan yhteen tulostiedostoon. Käyttöliittymässä myös visualisoidaan, tuleeko laitteilta dataa ja tallentuuko se oikein.

# 4 Sovelluksen käyttöliittymä

Luvussa kuvaillaan lyhyesti sovelluksen pääikkunan rakennetta ja sovelluksen eri näkymiä. Käyttöliittymän eri toiminnot on selitetty tarkemmin sovelluksen käyttöohjeessa [10].

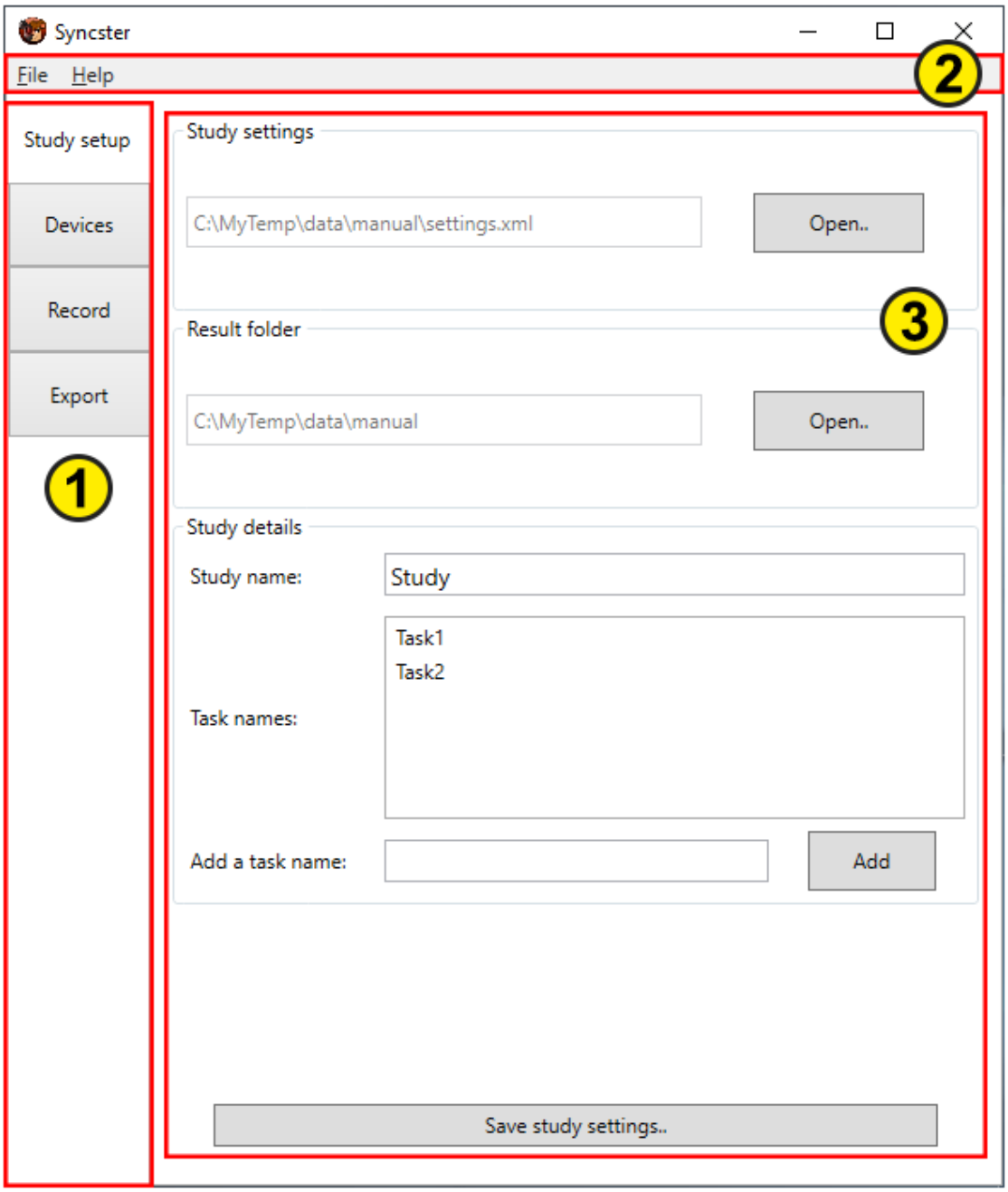
## 4.1 Pääikkunan rakenne

Kuva 4.1 havainnollistaa sovelluksen pääikkunan rakennetta. Siihen on numeroitu käyttöliittymän eri osat, joita ovat:

**1** Navigointipalkki, jonka välilehtien avulla käyttäjä voi siirtyä sovelluksen näkymästä toiseen.

**2** Valikko, jonka kautta voi avata ja tallentaa asetustiedostoja, sekä avata sovelluksen käyttöohjeen.

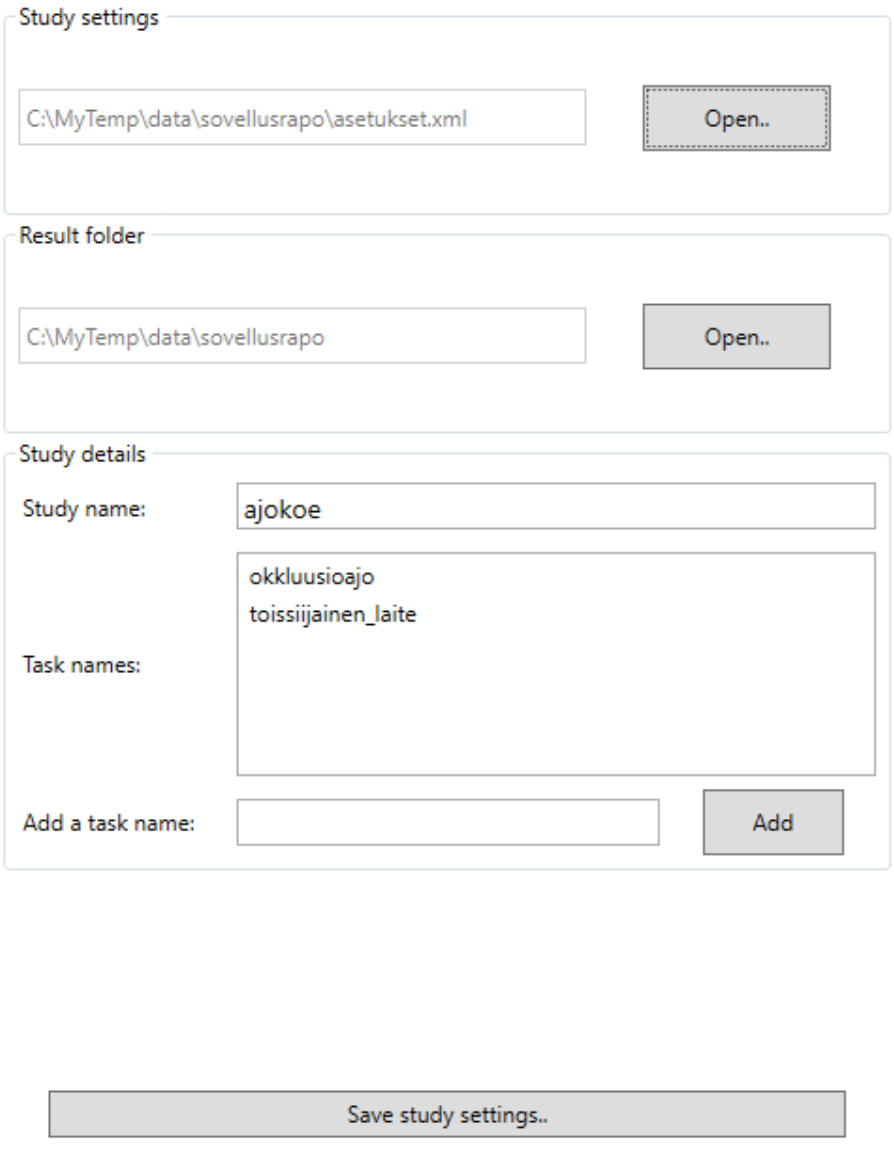
**3** Valittu näkymä.




Kuva 4.1: Sovelluksen pääikkuna.

## 4.2 Kokeen tietojen täyttäminen

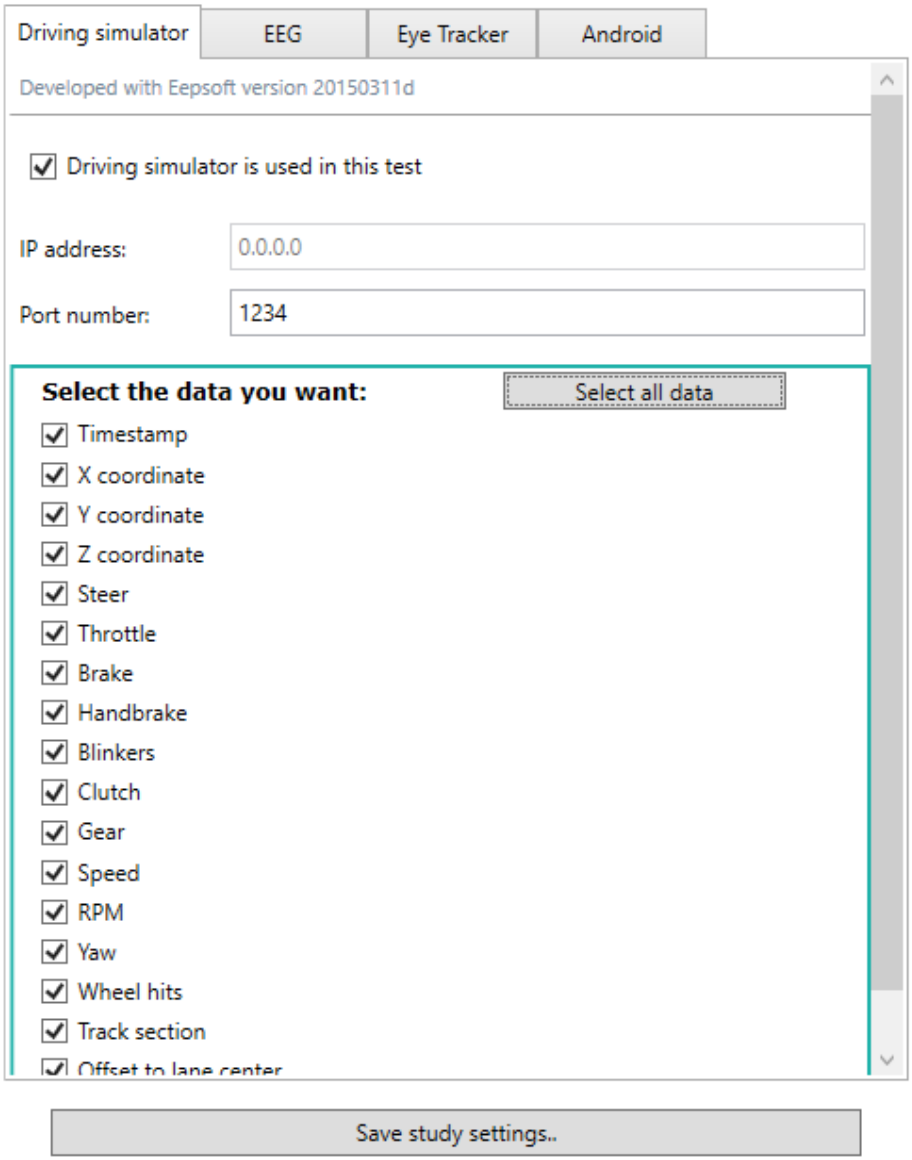
Kuva 4.2 esitetään kokeen asetusten täyttämiseen tarkoitettu näkymä, joka aukeaa *Study setup* –välilehdellä. Näkymässä valitaan kansio, johon kerätty data tallennetaan, sekä syötetään kokeen ja tehtävien nimet. Valitut asetukset voidaan tallentaa asetustiedostoksi *Save study settings..* –painiketta painamalla ja tallennettu asetustiedosto voidaan ladata *Study settings* –kohdasta.




Kuva 4.2: Kokeen asetusten valitsemisnäkymä.

## 4.3 Laitemoduulikohtaiset asetukset

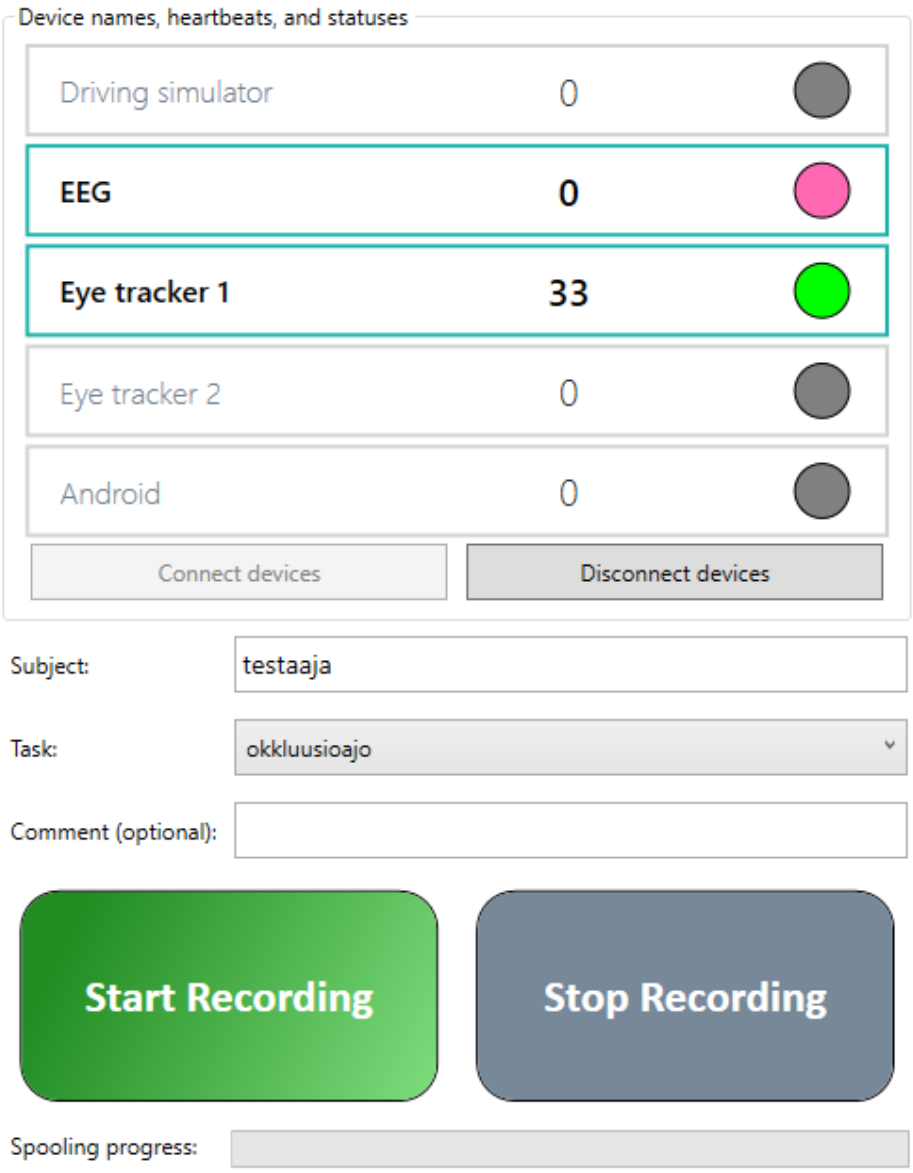
Laitemoduulikohtaiset asetukset tehdään *Devices* –välilehdellä. Jokaiselle moduulille on oma välilehtensä, jossa muun muassa syötetään IP-osoite ja porttinumero, joita moduulin on tarkoitus kuunnella, sekä valitaan halutut dataosiot. Lisäksi jokaisen laitteen välilehdellä on merkintä laitteen oman ohjelmiston versionumerosta, joka on ollut käytössä Syncsteriä kehitettäessä. Kuva 4.3 esittää ajosimulaattorin asetusnäkymän.




Kuva 4.3: Ajosimulaattorin laitekohtaiset valinnat

## 4.4 Reaaliaikainen tallennus

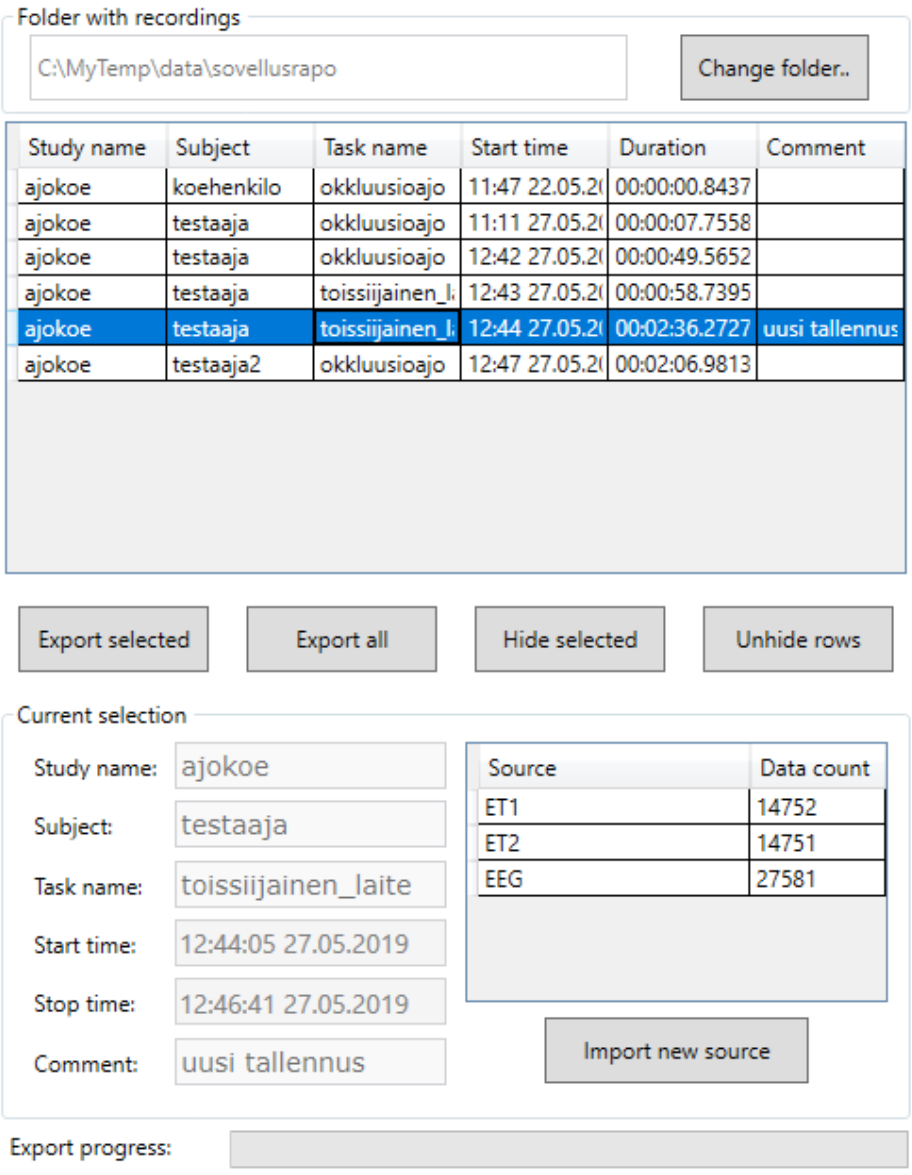
Tallennusnäkymässä käyttäjä voi tallentaa laitteiden lähettämää dataa reaaliaikaisesti. Kuva 4.4 näyttää, kuinka valitut laitemoduulit korostetaan vihreällä reunalla. Moduulin tilaa kuvataan sydämensykkeellä ja värillisellä ympyrällä.




Kuva 4.4: Tallennusnäkymä. Valitut moduulit EEG ja Eye Tracker 1 on yhdistetty, mutta EEG-moduuli ei ole onnistunut yhdistämisessä. Vaaleanpunainen ympyrä kertoo moduulin virhetilanteesta.

## 4.5 Datan tuominen tiedostosta ja vieminen tulostiedostoon

Datan tuominen ja vieminen tulostiedostoon tapahtuu *Export* –välilehdellä. Kaikki valitussa kansiossa sijaitsevat tallennukset esitetään taulukossa, josta käyttäjä voi valita tallennuksen, ja tehdä sille halutut toimenpiteet. Valitusta tallennuksesta näytetään yleistietoja, kuten laitemoduulien keräämien datarivien määrä.




Kuva 4.5: Datan tuomiseen ja viemiseen käytettävä näkymä.

# 5 Sovelluksen toiminta ja rakenne

Luvussa esitellään Syncsterin ja Touchsterin yleistä rakennetta ja tärkeimpiä toteutusratkaisuja. Yksityiskohtaisemmat kuvaukset löytyvät rakennekuvauksesta [3].

Syncster jakautuu Kuva 5.1: Sovelluksen yleisrakenne. mukaisesti kolmeen suurempaan kokonaisuuteen; graafiseen käyttöliittymään, manageriin ja laitemoduuleihin. Graafisen käyttöliittymän tehtävä on vastaanottaa käyttäjän syötteitä ja esittää ohjelman tila. Laitemoduulien tehtävä on puolestaan kerätä ja jäsentää laitteilta tulevaa dataa. Manageri toimii näiden kahden välissä, käynnistäen ja lopettaen tallennuksia käyttäjän käskystä ja välittäen moduulien viestejä käyttöliittymälle. Lisäksi manageri hoitaa välitallennusten muodostamisen ja kirjoittamisen tiedostoon. Kolmijaolla on pyritty varmistamaan selkeät rajat ja vastuunjaot kunkin osa-alueen välillä.


Kuva 5.1: Sovelluksen yleisrakenne.

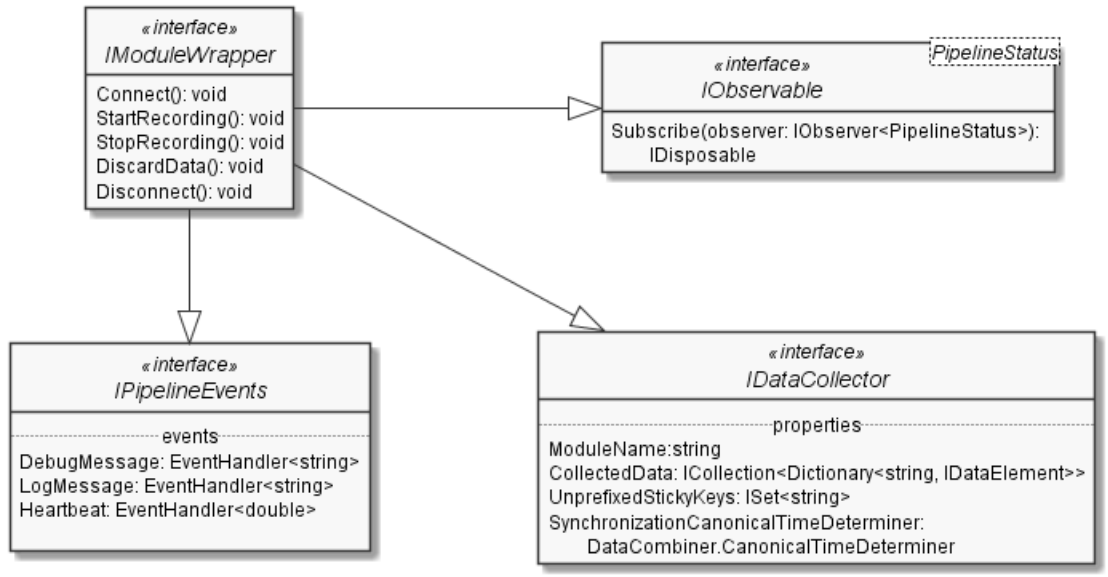
Syncsterin lähdekoodi on jaettu kokonaisuuksien mukaan kolmeen Visual Studio -projektiin: ALBackend, ALManager ja ALGUI. Lisäksi yksikkötesteille on oma ALUnitTests-niminen projekti. Touchsterin lähdekoodi löytyy omasta AndroidTouch-projektistaan. Syncsterin ja Touchsterin lisäksi projektiryhmä kirjoitti `peltihamster_udp.lua`-skriptitiedoston, jolla saadaan kerättyä ja lähetettyä dataa ajosimulaattorilta.

## 5.1 ALBackend

ALBackend-projektin alta löytyvät laitekohtaiset moduulit, joiden tehtävä on vastaanottaa kohdelaitteen lähettämää dataa reaaliajassa ja jäsentää se jatkokä-

sittelyn mahdollistavaan tietorakenteeseen. Moduulit toimivat toisistaan riippumatta, eli tallennuksen aikana voi käytössä olla mikä tahansa moduuliyhdistelmä, eikä sovelluksen toiminta riipu yksittäisestä moduulista.

Kaikki moduulit toteuttavat `IModuleWrapper`-rajapinnan, joka tarjoaa metodit yhteyksien ja tallennuksien käynnistämiseen ja lopettamiseen reaaliajassa. `IModuleWrapperin` kautta moduuli perii Kuva 5.2 mukaisesti myös kolme muuta rajapintaa. `IDataCollector` tarjoaa tarvittavat tiedot välitallennusten luomiseksi. `IPipelineEvents`- ja `IObservable<PipelineStatus>`-rajapinnat mahdollistavat tilapäivitysten ja muiden viestien lähettämisen.




Kuva 5.2: IModuleWrapper rajapinta ja sen perinnät

Reaaliaikaisten moduulien peruspohjakomponentti on `AbstractPipeline-Component`, joka käynnissä ollessaan käsittelee sisääntulevaa dataa ja siirtää sen sitten ulostulojonoonsa. Kaksi `AbstractPipelineComponenttia` voidaan putkittaa peräkkäin siten, että jälkimmäinen ottaa vastaan dataa ensimmäisen ulostulosjonosta. Näin datan käsittely voidaan jakaa eri komponenttien välille. Esimerkin tästä näkee rakennekuvauksen ajosimulaattorimoduulin luokkakaaviosta [3].

Jokainen putkikomponentti pyörii omassa säikeessään. Säieturvallisuus on huomioitu käyttämällä datan keräämiseen säieturvallisia `BlockingCollection`- ja `ConcurrentBag`-tietorakenteita. Yksittäinen datarivi esitetään `Dictionaryna`, jonka avaimet vastaavat sarakkeiden nimiä ja arvot ovat `IDataElementtejä`. `IDataElement` on rajapinta sovelluksen tukemille tietotyypeille. Vahvalla tyypityksellä varmistetaan, että sovelluksen vastaanottama data on sitä mitä odotetaankin.

Reaaliaikaisten moduulien ohella ALBackend sisältää `ETFileReader`-komponentin, joka lukee D-Labin tuottaman tallennustiedoston ja jäsentää sen samojen tietotyyppien avulla kuin reaaliaikaset moduulit. Tiedostolukijan tarve tuli selväksi, kun ilmeni, ettei tilaajien kaipaamaa AOI-dataa saada reaaliajassa lähetettyä. Koska muita vastaavia tiedostoslukijoita ei sovelluksessa toistaiseksi ole, ei niitä varten ole luotu yhteistä rajapintaa. Jatkokehityksessä tämä voi kuitenkin osoittautua aiheelliseksi.

Tulostiedoston sisällön tuottaa `DataCombiner`-luokka, joka järjestää eri lähteistä tulleet datarivit kanonisen aikaleiman perusteella ja yhdistää kahden millisekunnin sisällä olevat datarivit samalle tulostiedoston riville. `DataCombiner` huomioi myös sticky keyt lisäämällä uudelle riville edellisen rivin arvon, jos muuttujalle ei muuten ole asetettu arvoa.

## 5.2 ALManager

Siinä missä ALBackend sisältää pääasiassa itsenäisiä apukomponentteja, ALManagerin komponentit vastaavat sovelluksen yleisestä toimintalogiikasta. ALManagerin pääluokka on `Manager`, joka tarjoaa asynkronisen ohjelmointirajapinnan käskyjen välittämiselle laitemoduuleille ja muille komponenteille. `Manager` myös välittää moduulien tilan, sydämensykkeet ja lokiviestit eteenpäin, sekä raportoi välitallennusten ja tulostiedoston muodostamisen etenemisen. ALManagerin kehityksessä on noudatettu Taskeihin pohjautuvaa asynk-

ronista mallia [11] (Task-based Asynchronous Pattern). Asynkronisuuden ansiosta käyttöliittymä reagoi virkeästi käyttäjän komentoihin samalla, kun sovellus hoitaa aikaa vievää datan käsittelyä taustalla.

Moduulien käskyttämisen ohella managerin vastuualueeseen kuuluu datan jälkikäsittely ja välitallennusten tekeminen. Tallennuksen päätyttyä datasta suodatetaan pois ei-kaivatut dataosiot ja lisätään kanoninen aikaleima jokaiselle datariville. Sen jälkeen kunkin moduulin keräämä data tallennetaan moduulin mukaan nimettyyn tiedostoon. Kaikki samaan tallennukseen kuuluvat tiedostot sijoitetaan yhteen kansioon, jonka nimi on muodossa `[kokeen tunniste]_[koehenkilön tunniste]_[tehtävä]_[aikaleima]_[kommentti]`. Lisäksi kansioon tallennetaan `meta.xml`-tiedosto, joka kertoo mitkä tiedostot tallennukseen kuuluvat ja niistä lasketut tarkistussummat, joiden avulla pyritään varmistamaan tiedon eheys vientivaiheessa.

Välitallennusratkaisuun päädyttiin, koska tilaajien kaipaamaa AOI-dataa ei saada reaaliajassa D-Labista. Ratkaisun ansiosta jälkikäteen tuotava data voidaan yksinkertaisesti lisätä uutena tiedostona muiden joukkoon, ja samalla tarvitsee vain päivittää metatiedosto. Lisäksi aikaa vievä datan yhdistäminen voidaan jättää tehtäväksi silloin, kun käyttäjä haluaa.

Välitallennusformaattia voidaan kuvata eräänlaisena näennäis-CSV:nä. Tiedoston otsikkoriville tulee tieto moduulin nimestä ja sticky key-muuttujista. Sen jälkeen jokaiselle riville kirjataan aina yhden datarivin sisältämät arvot, sarakkeiden nimet ja tietotyypit. Näin tiedoston sisältö saadaan palautettua takaisin alkuperäiseen tietorakenteeseen. Kotikutoiseen sarjallistusratkaisuun päädyttiin, koska alun perin käytetty .NET Frameworkin oma binäärisarjallistus osoittautui tuskallisen hitaaksi pitempien tallennuksien yhteydessä. Tiedostokokojen hillitsemiseksi välitallennustiedostot pakataan GZip-formaattiin.

Metatiedosto tuotetaan `HamsterDataSetin` avulla. `HamsterDataSet` on .NET Frameworkin `DataSet`-luokasta peritty tietokantakomponentti, jota Syncster käyttää niin yksittäisen tallennuksen metatietojen sarjallistamiseen

kuin useampien tallennusten esittämiseen käyttöliittymässä. `DataSetin` kirjoitusoperaatiot eivät ole säieturvallisia, mikä huomioitiin kehittämällä `IAsycHamsterDataSet`-rajapinta, joka takaa säieturvalliset metodit tietokannan käsittelylle asynkronisuudesta huolimatta.

## 5.3 ALGUI

ALGUI-projekti sisältää graafisen käyttöliittymän lähdekoodin. Käyttöliittymä on kirjoitettu WPF-kirjaston avulla XAML-merkintäkieltä käyttäen. ALGUI toimii Syncsterin käynnistysprojektina, joka käynnistyessään luo sovelluksen pääikkunan.

Käyttöliittymä huolehtii käyttäjän syötteiden vastaanottamisen ohella myös niiden validoinnista. Validointisääntöjä on laadittu niin tiedostonimille, IP-osoitteille kuin porttinumeroille. Sovellus ilmoittaa epäkelvoista syötteistä Kuva 5.3 osoittamalla tavalla.


Kuva 5.3: Käyttöliittymä ilmoittaa punavärillä virheellisistä syötteistä, kuten tiedostonimiin kelpaamattomista merkeistä.

Käyttöliittymä varmistaa myös, ettei käyttäjä pääse yhdistämään laitemoduuleita tai aloittamaan tallennusta ennen kuin pakolliset tiedot on syötetty sovellukseen. Jos sovellus kohtaa jonkin virhetilanteen, siitä näytetään virheilmoitus erillisessä viestiruudussa.

## 5.4 ALUnitTests

ALUnitTests-projektiin on sisällytetty kaikkien projektien yksikkötestit. Testit käyttävät MSTests v2 -ohjelmistokehystä. Varsinaisten testien lisäksi projekti sisältää muutamia yleiskäyttöisiä apuluokkia testaamisen helpottamiseen. `DummyPipelineComponenttia`, `DelayablePipelineComponenttia` ja

`DummyDataCollectoria` voidaan käyttää putkikomponenttien ja datan kerääjien sijaisolioina, ja `DataGeneratorilla` voidaan tuottaa sattumanvaraista dataa.

## 5.5 Touchster

Touchster on Android-käyttöjärjestelmälle kehitetty sovellus, joka lähettää Syncsterille tiedon siitä, milloin puhelimen kosketusnäyttöä on painettu. Sovellus on kirjoitettu C#:lla käyttäen apuna Visual Studion Xamarin-kehitystyökalua.

Touchsterin yhdistäminen Syncsteriin on toteutettu TCP-yhteyttä käyttäen. Touchster yhdistetään Syncsteriin sen jälkeen, kun Syncsterin moduulit on laitettu käyntiin. Tähän ratkaisuun päädyttiin, koska Touchsterin yhdistäminen Syncsteriin on Syncsterin käytön kannalta helpompaa. Touchster lähettää kosketushetkellä Syncsterille Android-laitteen oman aikaleiman, joka tulee näkyviin myös lopulliseen CSV-tiedostoon.

Kosketuksen tunnistaminen laitteen näytöltä toteutettiin luomalla laitteen näytön kulmaan käyttäjälle näkymätön taso, jonka ulkopuolelta kosketuksia pystytään havaitsemaan. Rajoituksena tälle tavalle on se, että kosketuksesta ei pystytä havaitsemaan sen koordinaatteja ja kestoa. Androidin tiukentuneiden käytänteiden ja sovelluksen vaatimusten takia kosketuksen havaitseminen muilla tavoilla olisi ollut käytännössä hyvin vaikeaa.

## 5.6 Lua-skripti ajosimuulaattoridatan lähettämiseen

Ajolaboratorion ajokokeet suoritetaan EepSoftin ajosimulaattoriohjelmalla. Projektiryhmä kirjoitti Lua-skriptin, joka kerää simulaattorilta dataa esimerkiksi ajoneuvon sijainnista ja nopeudesta ja lähettää sen UDP-protokollaa käyttäen eteenpäin. Skripti annetaan parametrina simulaattorille. Tarkemmat ohjeet skriptin käytöstä löytyvät sovelluksen käyttöohjeesta [10].

# 6 Ohjelmointikäytänteet

Sovelluksen lähdekoodissa on pyritty noudattamaan C#:n käytänteitä. Käytänteiden noudattaminen on tarkastettu teknisen ohjaajan johdolla suoritetuissa katselmoinneissa. Niissä käytiin läpi kaikki ryhmän kirjoittama koodi, mukaan lukien Lua-skripti ja Touchsterin lähdekoodi. Katselmointien huomiot liittyivät pääosin koodin muoto- ja kauneusvirheisiin, kuten epäjohdonmukaisiin nimeämisiin, kirjoitusvirheisiin ja ylipitkiin koodiriveihin. Nämä virheet korjattiin lopulliseen lähdekoodiin. Muuten katselmointien johtopäätös oli, että lähdekoodi noudattaa hyviä ohjelmointitapoja.

## 6.1 Muotoilu-, nimeämis- ja kommentointikäytänteet

Lähdekoodi ja kommentit on kirjoitettu englanniksi noudattaen ohjelmointikielen käytänteitä [13]. Nimiavaruuksien, rajapintojen, luokkien, aliohjelmien, ja vakioiden nimet on kirjoitettu Pascal case -tyylillä (esimerkiksi `ColumnHeaderComparer`). Rajapinnat alkavat I-etuliitteellä (`ITwoWaySerializer`). Abstraktit luokat alkavat abstract-sanalla (`AbstractPipelineComponent`). Aliohjelmien lokaalit muuttujat on kirjoitettu camel case –tyylillä (`bufferIndex`). Luokkien yksityisten attribuuttien nimen eteen on lisätty alaviiva (`_prefixSeparatorChar`).

Lähdekoodi on pääosin rivitetty niin, että koodi- ja kommenttirivit ovat korkeintaan 80-90 merkin pituisia. Poikkeuksen tekevät XAML-tiedostot, sillä koettiin, että merkkauskielen rivittäminen ei palvelisi koodin luettavuutta, ja olisi muutenkin hyvin työlästä.

Lisenssiteksti (BSD 3-Clause License) on sijoitettu jokaisen kooditiedoston alkuun lukuun ottamatta Visual Studion automaattisesti tuottamia tiedostoja ja graafisen käyttöliittymän määrittelyyn käytettyjä XAML-tiedostoja.

## 6.2 Kehitysympäristö

Syncsterin ja Touchsterin lähdekoodi on kirjoitettu Visual Studio 2017 -ohjelmalla käyttäen .NET Frameworkin versiota 4.6.1. Xamarin-kehitystyökalun kirjastoja lukuun ottamatta kaikki kehityksessä käytetyt ulkopuoliset kirjastot kuuluvat .NET Frameworkin alle. Versiohallintaan käytettiin Git-ohjelmaa ja YouSource-järjestelmää.

Sovellus on pääasiassa kehitetty projektihuoneen Windows 10 -tietokoneilla. Jotta sovelluksen verkkoyhteysominaisuuksia voitiin kokeilla projektihuoneen koneiden välillä, piti projektihuoneen koneiden palomuuriasetuksiin tehdä yliopiston ryhmäkäytänteet ohittavia sääntöjä koneiden välisen verkkoliikenteen sallimiseksi. Kehitystyöhön käytetyt Android-laitteet yhdistettiin jyu-student-verkkoon, jotta yliopiston yleinen palomuuri ei estänyt yhteyden muodostamista.

Kehitysvaiheessa ryhmällä oli käytössään ajosimulaatiolaboratorion käyttämiä laitteita ja ohjelmistoja. Laitteistoon kuuluivat ratti-poljin-yhdistelmä, EEG-kypärä, silmäliikekamera ja Android-puhelin ja -tabletti. Ohjelmistoista käytössä olivat EepSoftin ajosimulaattori, DSI-Streamer ja D-Lab.

# 7 Testauskäytänteet ja tulokset

Projektisuunnitelman [9] mukaisesti ohjelmaa testattiin yksikkötestauksella, järjestelmätestauksella ja hyväksymistestauksella. Järjestelmätestauskerroilla ja hyväksymistestauksessa käytettiin sovelluksen sen hetkistä release-versiota.

## 7.1 Yksikkötestaus

ALUnitTests-projekti sisältää yhteensä 54 testimetodia, ja niiden koodikattavuus on Visual Studion testianalyysityökalun mukaan 61 % ALBackendin osalta ja 37 % ALManagerin osalta. ALGUI-projektia ja Touchsteria ei ole yksikkötestattu, sillä molempien yksikkötestaaminen koettiin. ALManagerin alhainen testikattavuus selittyy osittain sillä, että se käyttää ALBackendin komponentteja, jotka ovat puolestaan yksikkötestattuja. Testianalyysityökalu ei ota tätä huomioon.

Yksikkötestit jäivät kattavuudeltaan tavoitetasosta. Tämä selittyy osin ajanpuutteella. Isoimmaksi puutteeksi jäi asynkronisuuden testaaminen, jota ei tehty juuri lainkaan. Sovelluksen perustan muodostavat ALBackendin olennaisimmat komponentit on kuitenkin hyvin yksikkötestattu.

Yksikkötestien kirjoittamisessa on paikoin lipsuttu hyvistä ohjelmointikäytänteistä. Ne ovat harvakseltaan kommentoituja, ja usein yhdellä testimetodilla testataan useampaa toimintoa. Testien päivittäminen koodin muuttuessa voikin osoittautua työlääksi.

## 7.2 Järjestelmätestaus

Syncsterille järjestettiin projektin aikana kolme järjestelmätestausta. Kaikki järjestelmätestauskerrat suoritettiin projektihuoneen Windows 10 -tietokoneilla käyttäen Syncsterin sen hetkistä release-versiota. Jokaisesta testauskerrasta laadittiin raportti, jonka liitteeksi lisättiin testauskerralla kerätty data ja käytetyt asetustiedostot. Testauskertojen tulokset on esitetty Taulukko 1. Tarkemmat tulokset löytyvät järjestelmätestausraporteista [5][14][15].

| Testauskerta | Suoritetut | Hyväksytyt | Huomautukselliset | Puutteelliset | Virheelliset | Ohitetut |
|---|---|---|---|---|---|---|
| 26.4.2019 | 35 | 25 | 3 | 5 | 2 | 5 |
| 14.5.2019 | 40 | 30 | 5 | 2 | 3 | 2 |
| 24.5.2019 | 33 | 29 | 1 | 0 | 3 | 2 |

Taulukko 1: Järjestelmätestauskertojen tulokset

Ensimmäinen järjestelmätestaus suoritettiin järjestelmätestaussuunnitelman version 0.3.0 mukaisesti. Tässä vaiheessa Android- ja EEG-moduulit jäivät testaamatta, sillä ensin mainittu ei ollut vielä valmis testattavaksi, ja jälkimmäisen tarvitsemaa EEG-kypärää ei saatu toimimaan. Tässä vaiheessa sovelluksen virheenkäsittely ja validointi olivat monin paikoin puutteellisia, ja moni tärkeä ominaisuus oli vielä toteuttamatta.

Toisella testauskerralla noudatettiin järjestelmätestaussuunnitelman versiota 1.0.0. Testauksessa löydetyt virheet olivat pieniä, eikä sovellus enää kaatuillut puutteelliseen virheenkäsittelyyn. Ainoa merkittävä puute oli käyttöohjeen puuttuminen.

Kolmas järjestelmätestaus suoritettiin, kun kaikki suunnitellut ominaisuudet oli saatu toteutettua. Testaussuunnitelma oli sama kuin toisella kerralla. Testaussuunnitelmasta poikettiin testaamalla sovellusta eri kieliasetuksilla, minkä huomattiin aiheuttavan epäluotettavaa käytöstä välitallennustiedostojen lukemisessa.

## 7.3 Hyväksymistestaus

Hyväksymistestaus suoritettiin 17.5.2019 ajosimulaatiolaboratoriossa aitoa koetilannetta mukaillen. Testauksen suorittivat tilaajat projektiryhmän jäsenten toimiessa valvojina, koehenkilöinä ja teknisenä tukena. Testauksen tulokset löytyvät hyväksymistestausraportista [16]

Sovellus täytti tilaajien tarpeet. Käyttöliittymässä havaittiin muutamia mitättömiä käyttömukavuuteen vaikuttavia ongelmia. D-Lab toimi ailahtelevasti, eikä datan tuomista pystytty testauskerralla testaamaan. Datan tuominen kuitenkin hyväksytettiin tilaajalla jälkikäteen. Samalla havaittiin lokalisointiin liittyviä ongelmia, jotka sitten testattiin kolmannessa järjestelmätestauksessa.

## 7.4 Testauksen yhteenveto

Yleisesti ottaen testaus tuki sovelluksen laadunvarmistusprosessia, sillä jokaisella testauskerralla havaittiin ohjelmasta virheitä ja puutteita, jotka sitten saatiin korjattua.

Dokumentoitujen testauskertojen lisäksi sovellusta on myös koeteltu muun muassa rasittamalla sitä suurella datamäärällä ja muokkaamalla välitallennuksia käsin. Näillä keinoilla on pystytty havaitsemaan ja korjaamaan ongelmia sovelluksen suorituskyvyssä ja virheenkäsittelyssä.

Projektiryhmä ei onnistunut keksimään kaksisia keinoja ajallisen synkronoinnin testaamiseen. Tarkkuutta yritettiin muun muassa testata kahden testaajan voimin metronomin tahdistuksella. Toinen testaaja paineli puhelimen näyttöä, toisen vedellessä rattipolkimen liipaisimia. Toisella kerralla synkronointia koestettiin silmäliikekameran ja puhelimen avulla. Testaaja sulki silmänsä samalla kun kosketti puhelimen näyttöä. Nämä testauskerrat olivat liian lyhyitä perustavanlaatuisten johtopäätösten tekemiseen, mutta ne antoivat kuitenkin näyttöä siitä, että synkronointi onnistuu alle vaaditun 100 millisekunnin tarkkuuden. Synkronoinnin täsmällisyyttä olisi kuitenkin hyvä testata lisää.

Myös verkkoyhteyksien viiveistä koituvat haitat jäivät testaamatta. Ne olivat osa järjestelmätestaussuunnitelmaa, mutta muut osiot veivät niille varatun ajan. Epäluotettavien verkkoyhteyksien testaaminen vaatisi etukäteisvalmisteluja testausympäristön suhteen, joten sille voitaisiin laatia kokonaan oman testaussuunnitelmansa.

# 8 Tavoitteiden toteutuminen

Sovellusprojektin lopputuloksena oli toimiva sovellus, joka sai tilaajien hyväksynnän. Tärkeimmät vaatimukset saatiin toteutettua, eikä sovelluksessa ole merkittäviä puutteita. Osa ratkaisuista olisi kuitenkin kaivannut enemmän hiomista.

## 8.1 Vaatimusten täyttyminen

Sovelluksen vaatimukset luokiteltiin viiteen prioriteettiluokkaan: pakollinen, tärkeä, valinnainen, idea ja ei toteuteta. Kaikki 9 pakollista ja 21 tärkeää vaatimusta saatiin toteutettua [1].

Pakollisten vaatimusten mukaisesti Syncster mahdollistaa tallennuksessa käytettävien laitteiden valitsemisen, eikä toisaalta pakota valitsemaan mitään tiettyä laitetta.

Sovellus toimii Windows 10 –käyttöjärjestelmässä ja sen käyttöliittymä on kirjoitettu englanniksi. Käyttöliittymä mahdollistaa tallennuksen aloittamisen ja lopettamisen.

Laitemoduulit keräävät dataa sitä lähettäviltä laitteilta tai niiden ohjelmistoilta. Moduulit jäsentävät keräämänsä datan rakenteelliseen muotoon ja siirtävät datan erilliselle komponentille datojen yhdistämistä ja tiedostoon kirjoitusta varten. Jälkimmäinen vaatimus toteutuu pienen mutkan kautta välitallennusten takia.

Valinnaisista vaatimuksista toteuttamatta jäi vaatimus 6.8, jossa määriteltiin, ettei tallennusta voi aloittaa ennen kuin vähintään yksi dataosio on valittuna. Käytännössä tämä tarkoittaa sitä, että jos käyttäjä valitsee EEG- tai DS-moduulin ja suodattaa pois kaikki dataosiot, lopulliseen tulostiedostoon tulee vain sarake kanonisia aikaleimoille, kehysnumeroille ja alikehysnumeroille. Vaatimus jäi toteuttamatta, koska se todettiin liian työlääksi saatavaan hyötyyn nähden.

Idea-prioriteetin vaatimuksista toteuttamatta jäi D-Labin etäkäyttöön liittyneet vaatimukset 8.6 ja 8.7. Näiden toteuttamiseen tarvittavat ohjeet saatiin laitevalmistajalta vasta projektin loppusuoralla, joten niiden kehittämiselle ei jäänyt aikaa.

Ei toteuteta –prioriteetin vaatimukset olivat sellaisia, jotka todettiin vaatimusanalyysin edetessä tarpeettomiksi. Tähän kategoriaan lukeutuivat muun muassa suomenkielinen käyttöliittymä ja käyttöohje.

## 8.2 Epätyydyttävät ratkaisut toteutuksessa

Yksi sovelluksen tärkeimmistä tavoitteista oli tehdä uusien moduulien lisäämisestä mahdollisimman kivutonta jatkokehittäjälle. Tavoitteen ei voida katsoa täyttyneen täysin tyydyttävästi, sillä kuten teknisestä ohjeesta [12] ilmenee, uuden moduulin kytkeminen manageriin ja graafiseen käyttöliittymään sisältää useita vaiheita. Tämä johtuu siitä, että käyttöliittymä ja moduulien ajonaikainen luominen ovat pitkälti kovakoodattu vain nykyisiä moduuleja varten.

Toinen jatkokehitettävyyttä heikentävä seikka on yleiskäyttöisen pohjakomponentin puute tavupuskurien lukemiselle. Silmäliikekamera- ja ajosimulaattorimoduuli vastaanottavat rivimuotoista dataa, eli ne lukevat sisääntulevaa tietovirtaa aina siihen asti, kunnes rivinvaihto tulee vastaan. Tätä varten on olemassa `TCPUPDRawLineReader`, joka hoitaa rivien lukemisen ohella myös yhteyden käynnistykset, lopetukset ja virheiden käsittelyt. EEG- ja Android-moduulit, jotka puolestaan lukevat tietyn määrän tavuja puskuriin, joutuvat kumpikin toteuttamaan itse oman puskurin lukemisen.

`HamsterDataSet` otettiin käyttöön lyhyen prototyypittelyn jälkeen, koska se osoitti olevansa helposti sekä sarjallistettavissa että sidottavissa käyttöliittymään. Kehitystyön edetessä sen ylläpito aiheutti kuitenkin huomattavan määrän ylimääräistä ponnistelua säieturvattomuuden takia. Lisäksi erinäisten virhetilanteiden välttämiseksi sovellus joutuu luomaan tilapäisiä `HamsterDataSet`-olioita useissa eri kohdissa, mikä ei ole kovinkaan tyydyttävä ratkaisu.

Lähdekoodi sisältää jonkin verran päälle liimattuja viritelmiä, jotka palvelevat vain tiettyä moduulia. Tällainen on esimerkiksi `Exporter`-luokan `AdjustET-Timestamps`-metodi, joka tahdistaa tiedostosta luetun silmäliikekameradatan aikaleimat reaaliaikaisen silmäliikekameradatan kanssa. Vastaavalle toiminnallisuudelle voisi jatkossa olla tarvetta muidenkin moduulien kanssa, joten yleiskäyttöisempi ratkaisu olisi parempi. Reaaliaikaisen silmäliikekameradatan kanonisen aikaleiman määrittelyä voisi myös tarkentaa, sillä nyt kanoniseksi aikaleimaksi on määritelty datarivin saapumisaika sovellukseen. Kuitenkin kameran ohjelmisto ei lähetä jokaista datariviä säännöllisesti, vaan joskus niitä tulee useampi kerralla.

Android- ja EEG-moduuli eivät yritä yhdistää uudelleen kohdelaitteeseen, jos yhteys on katkennut. EEG:n kohdalla tämä on perusteltua, sillä sen kanoninen aikaleima riippuu DSI-Streamerin lähettämästä suhteellisesta aikaleimasta, joka nollaantuu ohjelman kaatuessa. Androidin kohdalla vastaavaa rajoitetta ei kuitenkaan ole. Pienenä kauneusvirheenä Android-moduulin vastaanottamien datarivien määrä näkyy vientinäkymässä kaksinkertaisena. Tämä johtuu siitä, että kosketus on määritelty sticky key –muuttujaksi, joten se pitää nollata lisäämällä ylimääräinen kosketus edellisen perään.

## 8.3 Toteutuksen haasteet

Sovelluksen kehityksessä jouduttiin huomioimaan monia asioita. Laitteissa ja niiden ohjelmistoissa oli merkittäviä eroja, joten jokainen laitemoduuli vaati omia erityisratkaisujaan. Datan tuominen jälkikäteen monimutkaisti sovellusta entisestään. Moni asia selvisi yrityksen ja erehdyksen kautta. Esimerkiksi välitallennusformaatti koki merkittävän muodonmuutoksen ennen nykyiseen ratkaisuun päätymistä.

Ryhmän jäsenten C#-taidot olivat projektin alussa vähän ruosteisia, joten asioiden opettelu vei aikansa. Rajallinen aikataulu yhdistettynä laajaan kokonaisuuteen johti paikoin kovakoodattuihin ratkaisuihin.

# 9 Ideoita jatkokehitykseen

Luvussa esitellään kehitysideoita, jotka jäivät projektiryhmältä toteuttamatta ja joilla nykyisen sovelluksen käyttökokemusta voitaisiin parantaa. Ohjeet kehitysympäristön pystytyksestä ja uusien moduulien lisäämisestä löytyvät puolestaan sovelluksen teknisestä ohjeesta [12].

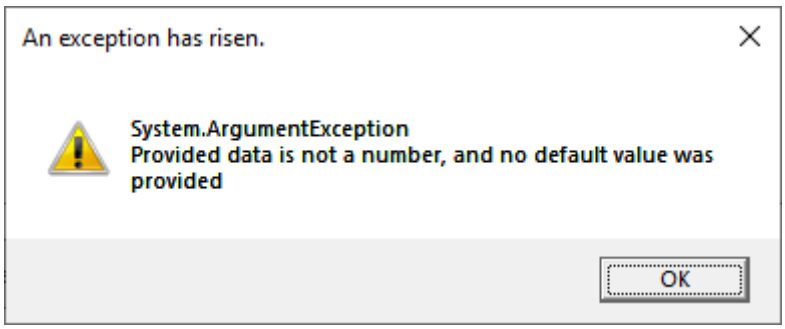
## 9.1 Olemassa olevien ominaisuuksien kehittäminen

Luvussa 8.3. mainittua kovakoodausta pitäisi pyrkiä vähentämään. Nykyinen asetusolio (`SerializableSettings`) voitaisiin koostaa useammasta laitekohtaisesta asetusoliosta. Käyttöliittymää varten voitaisiin luoda yleiskäyttöisiä pohjaelementtejä asetusten ja laitteiden tilojen esittämistä varten.

Syncsterille ei toistaiseksi ole määritelty suorituskykyyn liittyviä vaatimuksia. Jatkokehityksessä voitaisiin huomioida ainakin muistinkäyttö, sillä nykyään sovellus syö paljon muistia vientivaiheessa eri yhdistettäessä. Yleisesti sovelluksen muistinkäyttö säilyy kohtuullisena, sillä ainoastaan tallennusten metatietoja säilytetään koko ajan muistissa. Alhainen muistinkäyttö on myös toivottavaa, jos Syncsteriä halutaan käyttää samalla koneella kuin D-Labia.

Sovelluksen muistinkäyttöä hillitään tällä hetkellä kutsumalla eksplisiittisesti roskienkeruuta isoimpien operaatioiden jälkeen. Parempi ratkaisu olisi IDisposable-rajapinnan laajempi käyttö. Tämän hetkisistä moduuleista vain ajosimulaatiomoduuli toteuttaa kyseisen rajapinnan.

Syncsterin virheenkäsittely on varsin kattavaa, eikä sovellusta saa kovinkaan herkästi kaatumaan. Käyttäjälle näytettävät virheilmoitukset eivät kuitenkaan usein kuvaa ongelmatilanteita kovinkaan selkokielisesti (esimerkkinä Kuva 9.1). Viestien esittämiseen käytettyjä modaalisia ikkunoita voitaisiin korvata käyttäjäystävällisemmällä ratkaisulla.




Kuva 9.1: Sovelluksen esittämä virheilmoitus, kun käyttäjä tuo tiedoston, jonka arvot eivät vastaa oletustyypitystä.

Jos kokeen tunnistetta tai tallennuskansiota ei ole syötetty kokeen asetuksissa, tallennusnäkymässä ei näy mitään indikaattoria sille, miksi tallennusta ei voida aloittaa. Kun laitemoduulit on yhdistetty, tallennusnäkymästä ei myöskään enää pääse takaisin koeasetuksiin tarkistamaan kokeen tunnistetta.

Jos käyttäjällä ei ole kirjoitusoikeutta valittuun tallennuskansioon, sovellus ei ilmoita tästä käyttäjälle kansion valinnan jälkeen. Virheilmoitus tulee vasta, kun sovellus yrittää tallentaa kyseiseen kansioon.

Tehtävien tunnisteita ei validoida, kun ne luetaan asetustiedostosta. Jos käyttäjä lisää asetustiedostoon käsin tunnisteita, jotka eivät sovellu tiedostonimeen, virheilmoitus esitetään vasta siinä vaiheessa, kun sovellus yrittää tehdä tallennusta virheellisen tehtävän nimellä.

Laitemoduulien ja managerin lähettämille lokiviesteille ei toistaiseksi tehdä mitään. Niiden esittämiselle voitaisiin varata jokin tila käyttöliittymästä.

Silmäliikekameramoduuli ei tiedä etukäteen, mitä dataosioita D-Labista on valittu lähetettäväksi. Datan tunnistamiseen käytetään `ETSchemaElementParser`-luokkaa. D-Labin päivitykset, lisenssin muutokset tai eri silmäliikekameran käyttö saattavat vaatia Syncsterin lähdekoodin muokkausta, jotta tunnistaminen toimii jatkossakin. Ylläpidon helpottamiseksi sovellus voisi lukea tunnistuksessa käytetyt asetukset käsin muokattavasta tiedostosta.

Syncster tukee tällä hetkellä silmäliikedatan vastaanottamista kahden portin kautta, mutta D-Lab kykenee lähettämään dataa useammankin portin kautta. Syncsterin rajoitus johtuu käyttöliittymästä, jossa dynaamisesti muuttuvan moduulien määrän esittäminen koettiin liian haastavaksi toteuttaa. Tarve useamman portin käytölle tulisi kartoittaa ja tehdä sen pohjalta muutoksia sovellukseen.

Sovelluksen tuontiominaisuus tukee tällä hetkellä vain D-Labin tuottamia tallennustiedostoja. Jos muutakin dataa halutaan tulevaisuudessa lukea tiedostosta, tulisi jatkokehittäjän kehittää yhteinen rajapinta tiedostolukijoille ja tehdä tarvittavat muutokset `Importer`-luokkaan ja graafiseen käyttöliittymään, jotta oikea lukija tulee valituksi. `ETFileReaderiä` voi käyttää lähtökohtana uusien tiedostolukijoiden toteuttamisessa.

Tallennusnäkymässä laitemoduulin tilaindikaattori näyttää vihreää moduulin ollessa *Running*-tilassa. TCP-yhteyttä käyttävien moduulien kohdalla tämä ei kuitenkaan vielä kerro, onko yhteys muodostunut vai ei, mikä olisi käyttäjälle erittäin hyödyllinen tieto. Tämän muuttaminen ei kuitenkaan ole yksinkertaista, sillä se vaatii joko uutta tilapäivityslogiikkaa tai merkittäviä muutoksia moduulien tai putkikomponenttien toiminnassa.

## 9.2 Uusien ominaisuuksien kehittäminen

Vaatimusmäärittelyn vaatimukset 8.6 ja 8.7 käsittelevät silmäliikekameran D-Lab-ohjelmiston etäkäyttöä. Etäkäytön avulla Syncster voisi aloittaa ja lopettaa D-Labin tallennuksen automaattisesti, mikä vähentäisi jonkin verran kokeenvetäjän nykyisiä toimenpiteitä. D-Labin tallennusta tarvitaan, jotta sovellukseen saadaan tuotua AOI-data. Projektiryhmä sai etäkäytön ohjeet laitevalmistajan tuotetuesta niin myöhään, että ominaisuutta ehdittiin vain lyhyesti prototyypittelemään.

AOI-datan automaattiseen lähettämiseen joko reaaliajassa tai jälkikäteen ei löytynyt ratkaisua projektin aikana. Se saattaa olla mahdollista, jos tilaajien nykyistä D-Lab-lisenssiä laajennetaan. Ainakin skriptaustyökalut sisältävä lisenssi saattaisi olla tutustumisen arvoinen. Toinen, joskin hyvin kyseenalainen tapa, olisi hakea AOI-data D-Labin pyörittämästä MongoDB-tietokannasta. Tätä ominaisuutta ei ole D-Labin puolelta dokumentoitu, eikä projektiryhmä osaa esittää kantaa idean toteutuskelpoisuuteen.

Käyttöliittymä ei kerro kuinka kauan tämän hetkinen tallennus on kestänyt. Tämä voitaisiin korjata lisäämällä tallennusnäkymään kello, joka mittaa kulunutta aikaa.

# 10 Yhteenveto

Peltihamsteri-projekti kehitti kevään 2019 Sovellusprojekti-kurssilla Jyväskylän yliopiston kognitiotieteen ajosimulaatiolaboratoriolle Syncster-sovelluksen, jonka avulla synkronoidaan ajallisesti ajosimulaatiokokeen eri laitteista saatava data ja kirjoitetaan ne tulostiedostoon. CSV-muotoisen tulostiedoston sisältöä on helppo käsitellä ja analysoida tilastoanalyysityökaluilla.

Syncsterille kehitettiin helppokäyttöinen käyttöliittymä, jonka kautta käyttäjä voi valita haluamansa laitteet ja niiden dataosiot, täyttää kokeen ja tallennuksen tiedot, aloittaa ja lopettaa tallennuksen sekä yhdistää eri laitteiden datan yhteen tulostiedostoon, mukaan lukien jälkikäteen tuotava AOI-data.

Projektissa kehitettiin myös Touchster-sovellus syötetietojen keräämiseen Android-laitteilta. Sovellus rekisteröi, milloin laitteen kosketusnäyttöön kosketaan. Ajosimulaatiolaboratorion kokeissa sitä käytetään mittaamaan kuinka usein ja milloin koehenkilö koskee toissijaisen tehtävän laitteeseen. Android-laitteen syötetiedot kerätään reaaliajassa ja synkronoidaan muilta laitteilta saadun datan kanssa Syncster-sovelluksella.

Projektin aikana toteutettiin kaikki pakolliset ja tärkeät vaatimukset, jotka sovellukselle asetettiin. Sovellus on myös jatkokehityskelpoinen, jos ajolaboratorio päivittää laitekantaansa.

Sovellukselle tehtiin yksikkötestejä, kolme järjestelmätestausta ja hyväksymistestaus. Projektin tekninen ohjaaja tarkasti koodin kaksi kertaa. Sovellus sai hyväksynnän niin tilaajilta kuin tekniseltä ohjaajaltakin.

# Lähteet

[1] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, "Peltihamsteri-projekti, Vaatimusmäärittely", Jyväskylän yliopisto, Informaatioteknologian tiedekunta, 31.5.2019.

[2] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, "Peltihamsteri-projekti, Projektiraportti", Jyväskylän yliopisto, Informaatioteknologian tiedekunta, 5.6.2019.

[3] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, "Peltihamsteri-projekti, Rakennekuvaus", Jyväskylän yliopisto, Informaatioteknologian tiedekunta, TBA.

[4] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, "Peltihamsteri-projekti, Järjestelmätestaussuunnitelma", Jyväskylän yliopisto, Informaatioteknologian tiedekunta, 14.5.2019.

[5] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, "Peltihamsteri-projekti, Järjestelmätestaus", Jyväskylän yliopisto, Informaatioteknologian tiedekunta, 26.4.2019.

[6] Jukka-Pekka Santanen, ”Tietotekniikan sovellusprojektien ohje”, saatavilla PDF-muodossa, Jyväskylän yliopisto, informaatioteknologian tiedekunta, 30.1.2017.

[7] Severi Jääskeläinen, Samuel Kaiponen, Heta Rekilä ja Sinikka Siironen, ”Monisiro Project, Application Report”, Jyväskylän yliopisto, informaatioteknologian tiedekunta, 25.6.2018.

[8] Sauli Flinkman, Jere Junttila, Tuukka Jurvakainen ja Anette Karhu, ”Isäxi-sovellusprojekti, Sovellusraportti”, Jyväskylän yliopisto, informaatioteknologian tiedekunta, 19.6.2017.

[9] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, "Peltihamsteri-projekti, Projektisuunnitelma", Jyväskylän yliopisto, Informaatioteknologian tiedekunta, 4.4.2019.

[10] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, ”Syncster, Instruction manual”, Jyväskylän yliopisto, Informaatioteknologian tiedekunta, 5.6.2019

[11] Ron Petrusha et al., ”Task-based asynchronous pattern (TAP)” `https://docs.microsoft.com/en-us/dotnet/standard/asynchronous-programming-patterns/task-based-asynchronous-pattern-tap`, Microsoft 26.2.2019.

[12] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, ”Peltihamsteri, Technical manual”, Jyväskylän yliopisto, Informaatioteknologian tiedekunta, 5.6.2019.

[13] Krzysztof Cwalina et al., “Naming Guidelines” `https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/naming-guidelines, Microsoft` 22.10.2008.

[14] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, "Peltihamsteri-projekti, Järjestelmätestaus", Jyväskylän yliopisto, Informaatioteknologian tiedekunta, 14.5.2019.

[15] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, "Peltihamsteri-projekti, Järjestelmätestaus", Jyväskylän yliopisto, Informaatioteknologian tiedekunta, 24.5.2019.

[16] Tuomo Kujala, "Peltihamsteri-projekti, Hyväksymistestausraportti", Jyväskylän yliopisto, Informaatioteknologian tiedekunta, 17.5.2019.