Peltihamsteri-sovellusprojekti

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

Sovellusraportti

Versio 0.2.0

Julkinen

13.6.2019

Jyväskylän yliopisto

informaatioteknologian tiedekunta

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

Tietoja dokumentista

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

Dokumentin nimi: Peltihamsteri-projekti, Sovellusraportti

Sivumäärä: 42

Tiedosto: Peltihamsteri_Sovellusraportti_0.2.0.pdf

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

Avainsanat: C#, jatkokehitys, käyttöliittymä, ohjelmointikäytänteet, sovellusrakenne, tavoitteet, testaus, toteutusratkaisut, 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 lisättiin.	Juhani Sundell
0.0.4	27.5.2019	Lisättiin kuvat kaikista näkymistä.	Juhani Sundell
0.0.5	3.6.2019	Sovelluksen toiminta ja rakenne –lukua täydennettiin.	Juhani Sundell
0.0.6	4.6.2019	Tavoitteiden toteutumista ja jatkokehitysideoita täydennettiin.	Juhani Sundell
0.0.7	5.6.2019	Touchsterin osio lisättiin sekä jatkokehitysideoita ja yhteenvetoa täydennettiin. Korjattiin kirjoitus- ja muotoiluvirheitä.	Juhani Sundell, Mari Kasanen, Leevi Liimatainen
0.1.0	6.6.2019	Viimeisteltiin kirjoitusasua ja karsittiin ylimääräistä selittämistä.	Juhani Sundell, Leevi Liimatainen
0.1.1	11.6.2019	Raportin kieliasua korjattiin ohjaajan palautteen perusteella.	Juhani Sundell
0.1.2	12.6.2019	Lisättiin Touchsterin käyttöliittymäkuvaus ja toteutusratkaisujen kehitys. Siivottiin ja järjestettiin lähteet. Täydennettiin testaustulosten raportointia.	Juhani Sundell, Mari Kasanen, Marina Mustonen
0.2.0	13.6.2019	Täsmennettiin datojen yhdistämisen toimintaperiaatetta ja viimeisteltiin tekstin yleistä rakennetta. Lisättiin uusi alaluku 9. lukuun käsittelemään sovelluksen virheitä.	Juhani Sundell, Arttu Ylä-Sahra

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. Sovelluksen vaatimukset ja niiden toteutuminen on käyty läpi vaatimusmäärittelyssä [1]. Projektiraportti [2] kuvaa projektin läpivientiä muun muassa tavoitteiden ja käytänteiden suhteen. Rakennekuvauksessa [3] kuvataan sovelluksen rakennetta ja toimintoja eri näkymien kautta. Järjestelmätestaussuunnitelmassa [4] kuvataan ohjelmistolle suoritettuja testejä. Testauskertojen tulokset on esitetty järjestelmätestausraporteissa [5][6] ja [7]. Hyväksymistestaussuunnitelma [8] ja hyväksymistestausraportti [9] kuvaavat tilaajan suorittamaa hyväksymistestauskertaa ja sen tuloksia. Dokumentin laatimisessa käytettiin apuna sovellusprojektien ohjetta [10] ja Monisiro- ja Isäxi-projektien sovellusraportteja [11] ja [12].

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	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 tuntia 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. t*ask*) 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 tilaajan suorittama testaus sovellukselle, jolla varmistetaan sovelluksen täyttävän tilaajan 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 aivosähkökäyrädataa.

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, joka havaitsee kosketusnäyttöön tehdyt kosketukset. Kosketustiedotkin synkronoidaan muun datan kanssa. Muu data kerätään reaaliajassa, mutta silmänliikekameran AOI-data luetaan jälkikäteen tiedostosta.

Sovelluksen vaatimuksiin kuului myös 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 Syncsterin ja Touchsterin käyttöliittymät

Luvussa kuvaillaan lyhyesti Syncsterin ja Touchsterin käyttöliittymää. Eri näkymien tarjoamat toiminnot on selitetty seikkaperäisemmin sovellusten yhteisessä käyttöohjeessa [13].

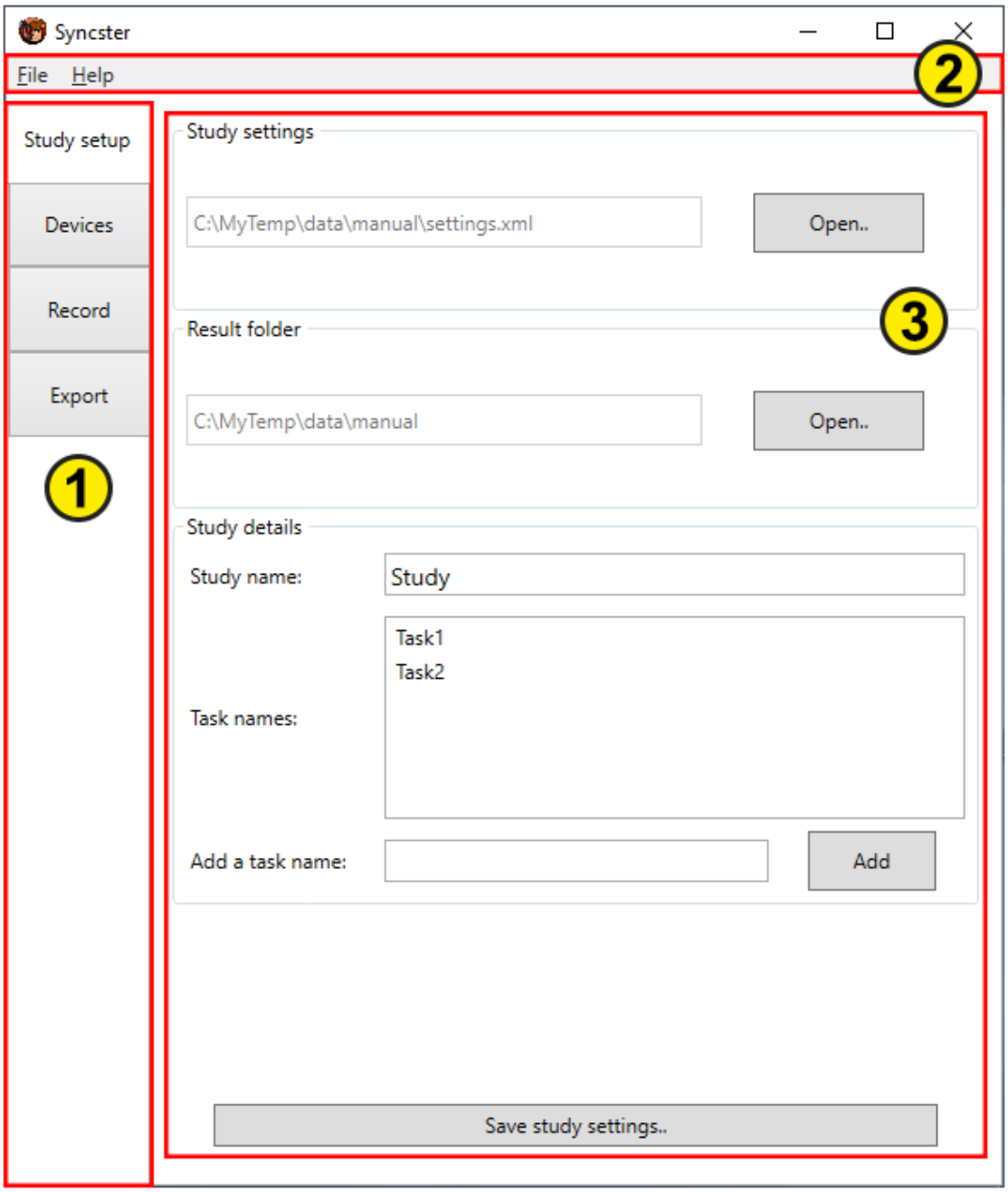
4.1 Syncsterin pääikkunan rakenne

Kuva 4.1 havainnollistaa sovelluksen pääikkunan rakennetta. Siihen on numeroitu seuraavat käyttöliittymän osat:

1 Navigointipalkin välilehtien avulla käyttäjä voi siirtyä sovelluksen näkymästä toiseen.

2 Valikon kautta käyttäjä voi avata ja tallentaa asetustiedostoja sekä avata sovelluksen käyttöohjeen.

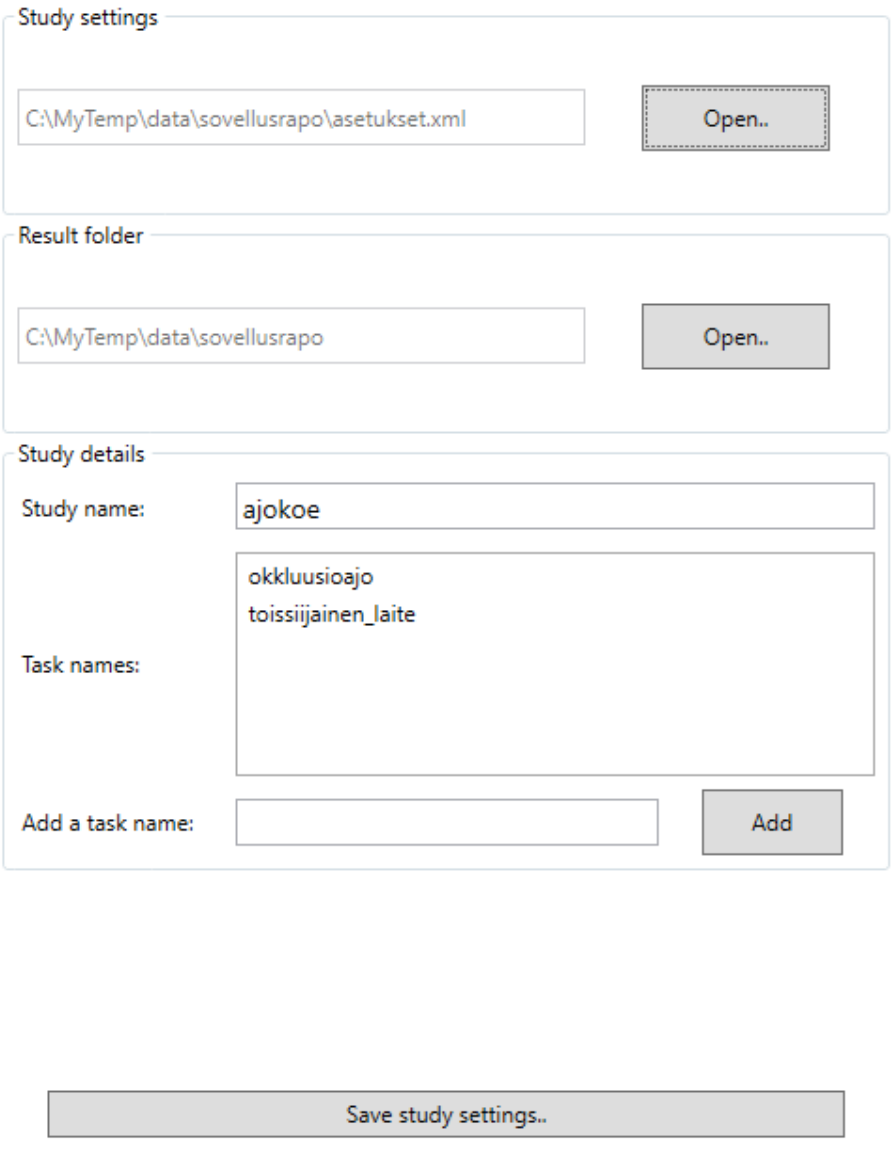
3 Valitussa näkymässä käyttäjä voi suorittaa näkymän tarjoamia toimintoja.



Kuva 4.1: Sovelluksen pääikkuna.

4.2 Kokeen tietojen täyttäminen

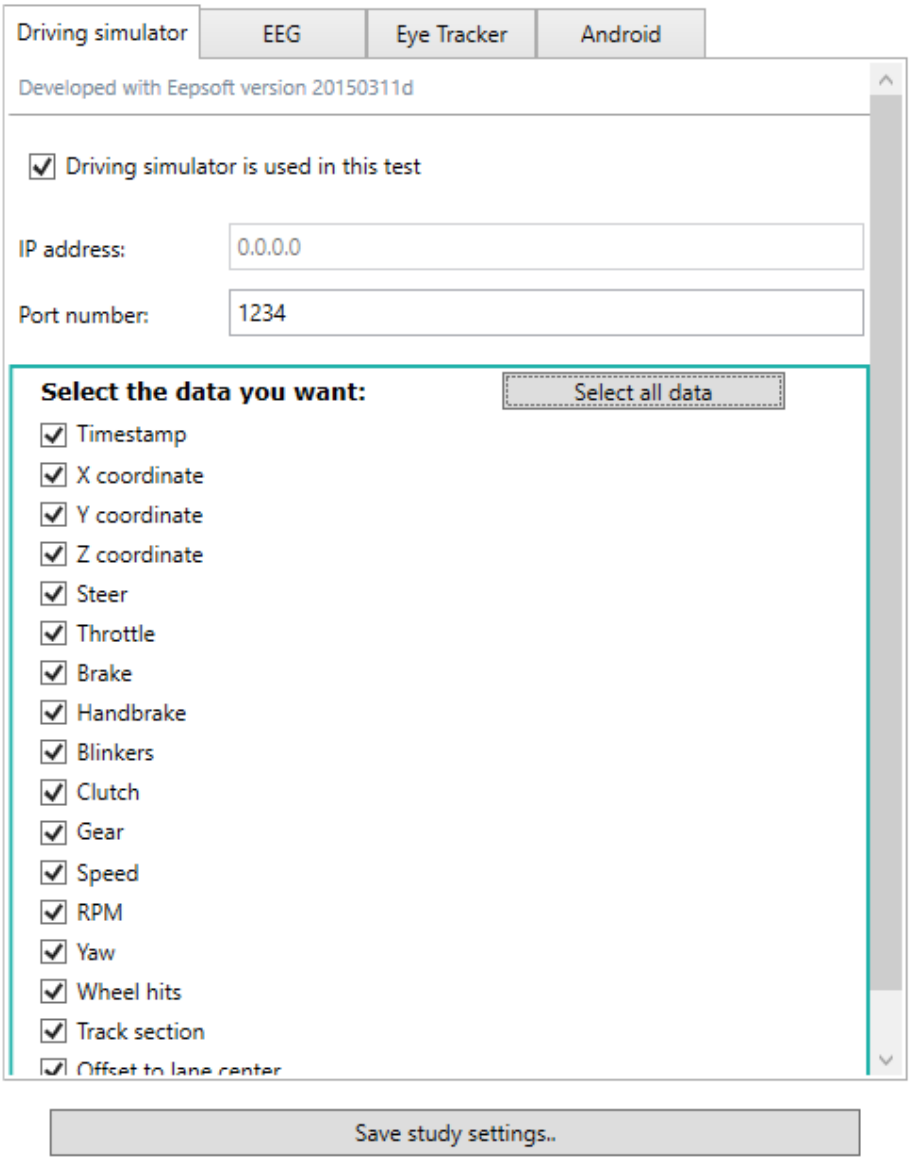
Kuva 4.2 esitetään kokeen asetusten täyttämiseen tarkoitettu näkymä, joka aukeaa *Study setup* -välilehdellä. Näkymässä valitaan kansio, johon kerätty data tallennetaan, sekä syötetään kokeen ja tehtävien nimet. Valitut asetukset voidaan tallentaa asetustiedostoksi *Save study settings..* -painiketta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



Kuva 4.2: Kokeen asetusten valitsemisnäkymä.

4.3 Laitemoduulikohtaiset asetukset

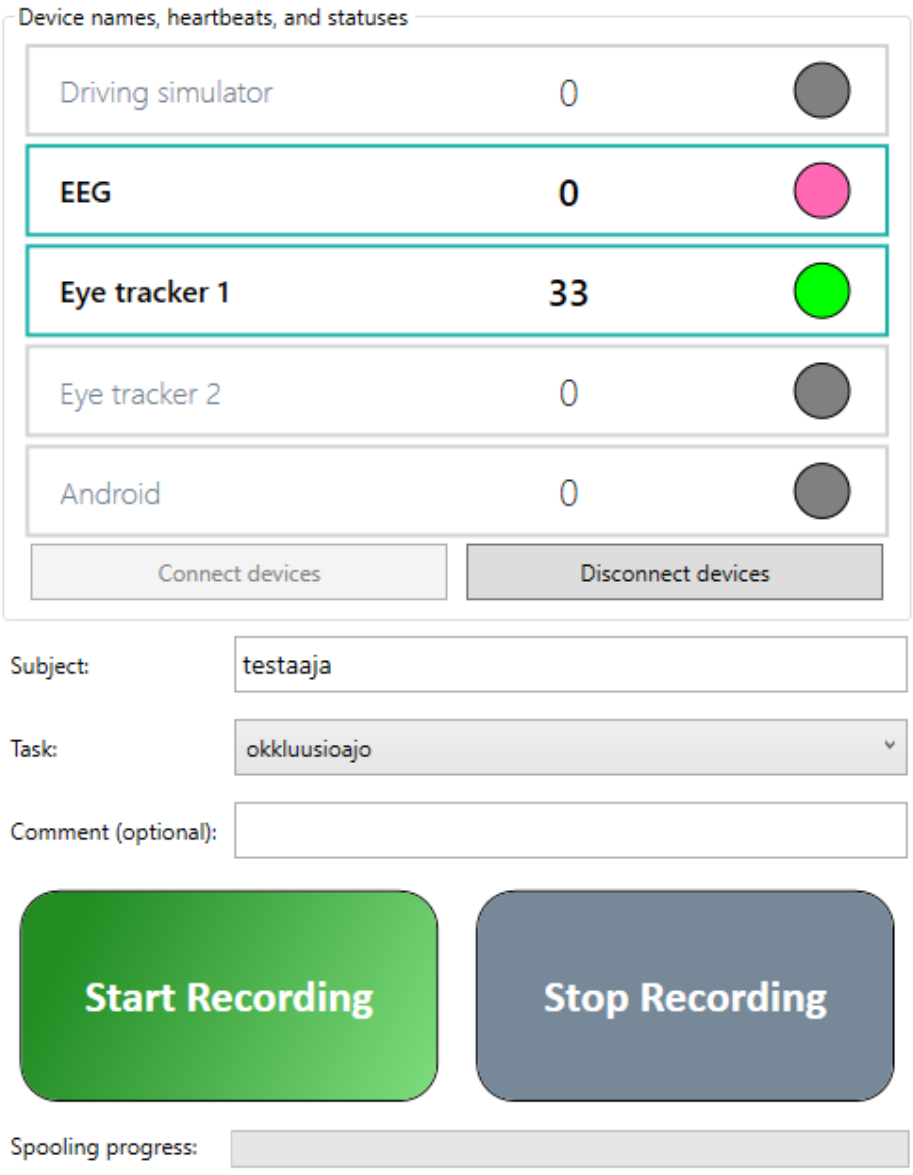
Laitemoduulikohtaiset asetukset tehdään *Devices*-välilehdellä. Jokaiselle moduulille on oma välilehtensä, jossa muun muassa syötetään IP-osoite ja porttinumero, joita moduulin on tarkoitus kuunnella, sekä valitaan halutut dataosiot. Lisäksi jokaisen laitteen välilehdellä on merkintä laitteen oman ohjelmiston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



Kuva 4.3: Ajosimulaattorin laitekohtaiset valinnat.

4.4 Reaaliaikainen tallennus

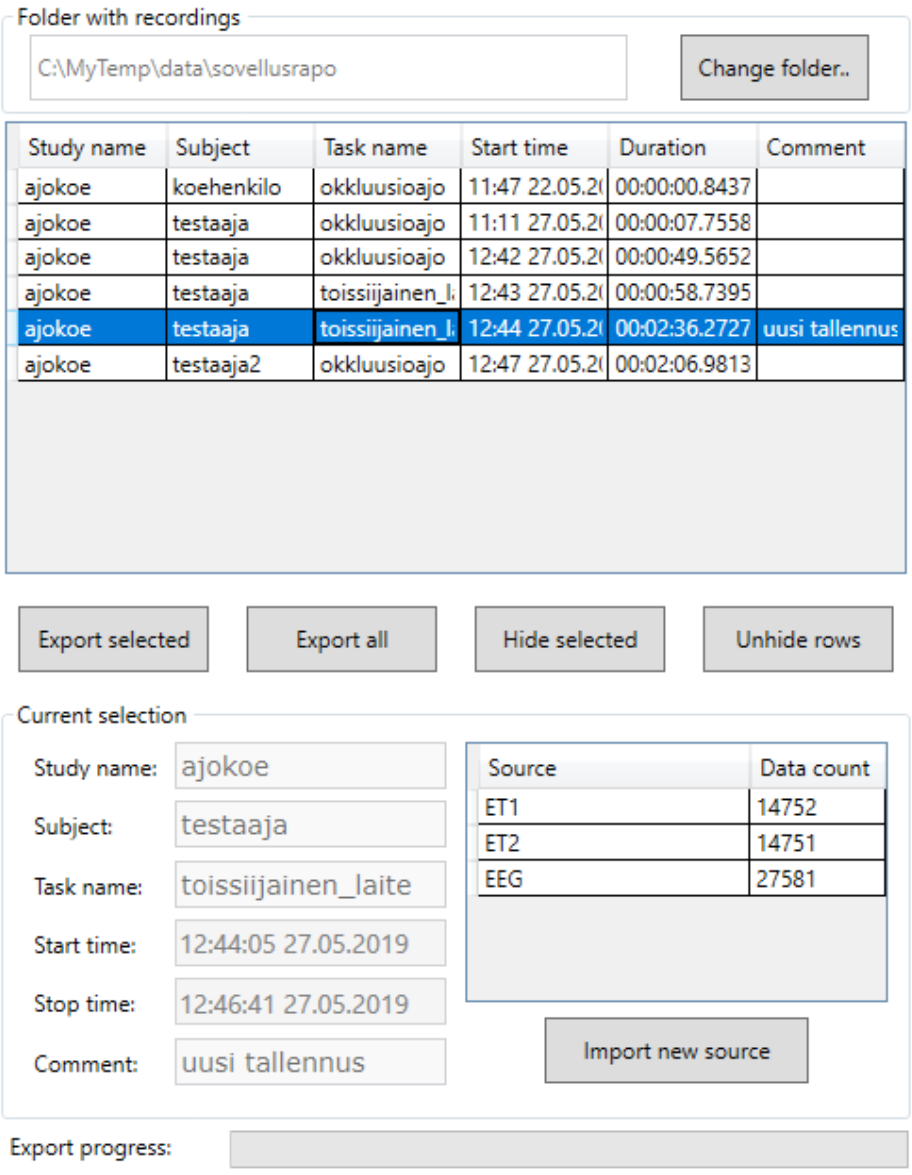
Tallennusnäkymässä käyttäjä voi tallentaa laitteiden lähettämää dataa reaaliaikaisesti. Valitut moduulit korostetaan ja niiden tila esitetään sydämensykkeellä ja värillisellä ympyrällä. Kuva 4.4 valitut moduulit EEG ja Eye Tracker 1 on yhdistetty, mutta EEG-moduuli on kohdannut virhetilanteen, mistä kertoo vaaleanpunainen ympyrä.



Kuva 4.4: Tallennusnäkymä, jossa näkyy myös moduulien tila.

4.5 Datan tuominen tiedostosta ja vieminen tulostiedostoon

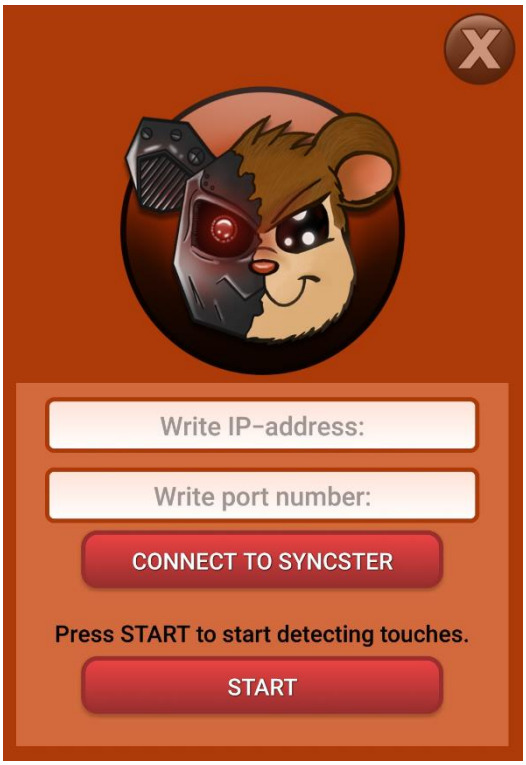
Uuden datan tuominen tallennukseen ja tallennuksen vieminen tulostiedostoon tapahtuu *Export*-välilehdellä. Kaikki valitussa kansiossa sijaitsevat tallennukset esitetään taulukossa, josta käyttäjä voi valita tallennuksen ja tehdä sille halutun toimenpiteen. Valitusta tallennuksesta näytetään yleistietoja, kuten laitemoduulien keräämien datarivien määrä.



Kuva 4.5: Datan tuomiseen ja viemiseen käytettävä näkymä.

4.6 Touchsterin päänäkymä

Android-sovellus Touchsterin päänäkymä on esitetty Kuva 4.6. Ikkunaan kirjoitetaan Syncsteriä pyörittävän tietokoneen IP-osoite ja porttinumero, jonka kautta Syncster vastaanottaa dataa. Kun Touchster on yhdistetty ja kosketusten tunnistaminen on aktivoitu, puhelimen muita sovelluksia voi alkaa käyttää normaalisti. Touchster piirtää muiden sovellusten päälle pienen näkymättömän tason, joka ei vaikuta sovellusten käyttämiseen, mutta joka mahdollistaa kosketusten havaitsemisen.



Kuva 4.6: Touchsterin päänäkymä.

5 Sovelluksen toiminta ja rakenne

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

5.1 Syncsterin yleisrakenne

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.

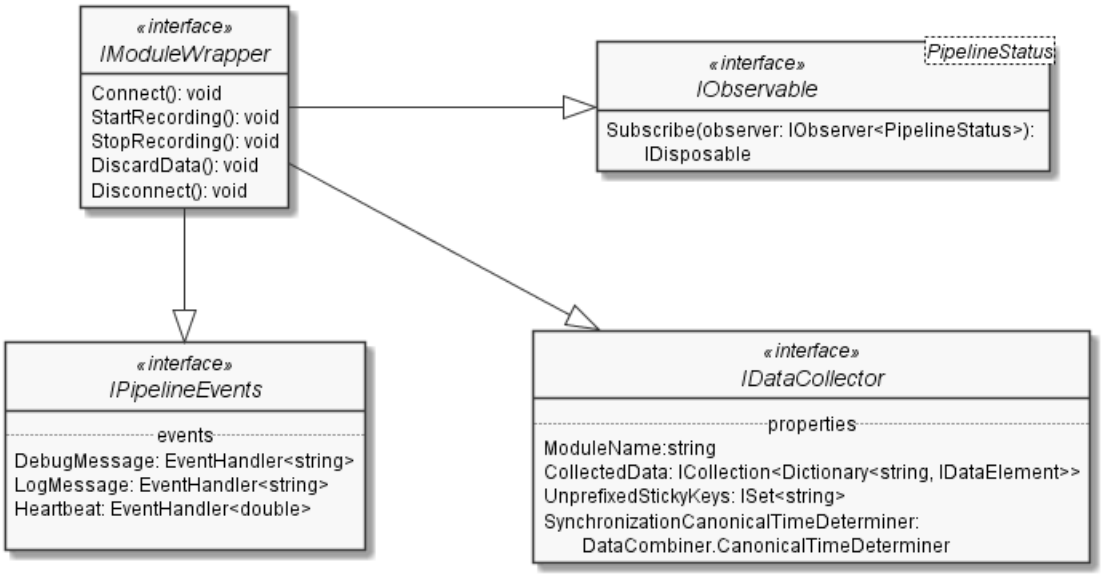
Kuva 5.1: Sovelluksen yleisrakenne.

Syncsterin lähdekoodi on jaettu kokonaisuuksien mukaan kolmeen Visual Studio -projektiin: ALBackend, ALManager ja ALGUI. Kaksi ensin mainittua ovat luokkakirjastoja, jotka sisältävät sovelluksen keskeisimmät komponentit. Viimeksi mainittu on Windows-sovellus, joka sisältää käyttöliittymän lähdekoodin. Yksikkötesteille on oma ALUnitTests-niminen projekti, ja Touchsterin lähdekoodi löytyy AndroidTouch-projektista. Syncsterin ja Touchsterin lisäksi projektiryhmä kirjoitti `peltihamster_udp.lua`-skriptitiedoston, jolla saadaan kerättyä ja lähetettyä dataa ajosimulaattorilta.

5.2 ALBackend

ALBackend-projektin alta löytyvät laitekohtaiset moduulit, joiden tehtävä on vastaanottaa kohdelaitteen lähettämää dataa reaaliajassa ja jäsentää se jatkokäsittelyn 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 tekemiseksi. `IPipelineEvents` ja `IObservable<PipelineStatus>` 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 `AbstractPipelineComponent`tia 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 `Dictionary`na, 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 muodostaman tallennustiedoston ja jäsentää sen samojen tietotyyppien avulla kuin reaaliaikaiset moduulit. Tiedostolukijan tarve tuli selväksi, kun tilaajan kaipaamaa AOI-dataa ei saatu reaaliajassa lähetettyä. Koska muita vastaavia tiedostolukijoita 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. Jos eri lähteistä tulleet datarivit ovat alle kahden millisekunnin sisällä toisistaan, ne yhdistetään samalle tulostiedoston riville. Kahden millisekunnin tarkkuus toteutetaan pyöristämällä kanoninen aikaleima lähimpään parilliseen millisekuntiin. Jos useammalla yhden lähteen datarivillä on sama pyöristetty aikaleima, ne järjestetään tulostiedostossa eri riveille ja numeroidaan alikehysnumeron mukaan. `DataCombiner` huomioi myös sticky keyt lisäämällä uudelle riville edellisen rivin arvon, jos muuttujalle ei ole asetettu sillä rivillä arvoa.

5.3 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 Microsoftin dokumentaatiossa [14] kuvattua Taskeihin pohjautuvaa asynkronista mallia (Task-based Asynchronous Pattern). Asynkronisuuden ansiosta käyttöliittymä vapautuu reagoimaan käyttäjän komentoihin samalla, kun sovellus hoitaa aikaa vievää datan käsittelyä taustalla.

Moduulien käskyttämisen ohella `Manager`in vastuualueeseen kuuluu datan jälkikäsittely ja välitallennusten tekeminen. Tallennuksen päätyttyä datasta suodatetaan pois valitsematta jätetyt 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 nimessä on alaviivoin eroteltuna *kokeen* ja *koehenkilön tunnisteet*, *tehtävä*, *aikaleima* ja vapaavalintainen *kommentti*. Jottei saman nimisiä tallennuksia pääse syntymään, aikaleima merkitään millisekunnin tarkkuudella. Lisäksi kansioon tallennetaan `meta.xml`-tiedosto, joka sisältää käyttöliittymässä näytettävien metatietojen lisäksi myös välitallennustiedostoista lasketut MD5-tarkistussummat. Tarkistussummien avulla pyritään varmistamaan tiedon eheys vientivaiheessa.

Välitallennusratkaisuun päädyttiin, koska tilaajan 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 suorittaa 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 kaksoispisteellä erotettuna. 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 muodostetaan `HamsterDataSet`in avulla. `HamsterDataSet` on .NET Frameworkin `DataSet`-luokasta peritty tietokantakomponentti, jota Syncster käyttää niin yksittäisen tallennuksen metatietojen sarjallistamiseen kuin useampien tallennusten esittämiseen käyttöliittymässä. `DataSet`in kirjoitusoperaatiot eivät ole säieturvallisia, mikä huomioitiin kehittämällä `IAsycHamsterDataSet`-rajapinta, joka takaa säieturvalliset metodit `HamsterDataSet`in käsittelylle asynkronisuudesta huolimatta.

5.4 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 ja `Manager`-instanssin, jonka kautta kommunikointi laitemoduulien kanssa onnistuu. Lisäksi käyttöliittymä ylläpitää omaa `IAsyncHamsterDataSet`-oliotaan, johon tehtyjen tallennusten metatiedot lisätään, ja joka on sidottu vientinäkymän taulukkoon.

Käyttöliittymä huolehtii käyttäjän syötteiden vastaanottamisen ohella myös niiden validoinnista. Validointisääntöjä on laadittu tiedostonimille, IP-osoitteille ja porttinumeroille. Sovellus ilmoittaa epäkelvoista syötteistä, kuten tiedostonimiin kelpaamattomista merkeistä Kuva 5.3 osoittamalla tavalla.

Kuva 5.3: Käyttöliittymä ilmoittaa punavärillä virheellisistä syötteistä.

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.

5.5 ALUnitTests

ALUnitTests-projektiin on sisällytetty kaikkien projektien yksikkötestit. Testit käyttävät MSTests v2 –ohjelmistokehystä, jonka käyttöohjeistus löytyy Microsoftin dokumentaatiosta [15]. Varsinaisten testien lisäksi projekti sisältää muutamia yleiskäyttöisiä komponentteja testaamisen helpottamiseen. Apuluokkia `DummyPipelineComponent`, `DelayablePipelineComponent` ja `DummyDataCollector` voidaan käyttää putkikomponenttien ja datan kerääjien sijaisolioina. `DataGenerator`illa puolestaan voidaan tuottaa sattumanvaraista dataa.

5.6 Touchster

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

Touchsterin yhdistäminen Syncsteriin on toteutettu TCP-yhteyttä käyttäen. Touchster voidaan yhdistää Syncsteriin sen jälkeen, kun Syncsterin Android-moduuli on käynnistetty. 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 tietoturvakäytänteiden ja sovelluksen vaatimusten takia kosketuksen havaitseminen muilla tavoilla olisi ollut käytännössä hyvin vaikeaa.

5.7 Lua-skripti ajosimulaattoridatan lähettämiseen

Ajolaboratorion ajokokeet suoritetaan EepSoftin ajosimulaattoriohjelmalla. Projektiryhmä kirjoitti Lua-skriptin, joka kerää simulaattorilta dataa esimerkiksi ajoneuvon sijainnista ja nopeudesta. Kerätty data lähetetään sen UDP-protokollaa käyttäen Syncsterille. Skripti annetaan parametrina simulaattorille. Tarkemmat ohjeet skriptin käytöstä löytyvät sovelluksen käyttöohjeesta [13].

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 Microsoftin dokumentaatiossa [16] määritettyjä käytänteitä. 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 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 muodostamia 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. Kehitystyössä käytetyt Android-laitteet yhdistettiin yliopiston langattomaan 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 sekä Android-puhelin ja -tabletti. Ohjelmistoista käytössä olivat EepSoftin ajosimulaattori, DSI-Streamer ja D-Lab.

7 Testauskäytänteet ja tulokset

Sovellusta testattiin yksikkötestauksella, järjestelmätestauksella ja hyväksymistestauksella. Yksikkötestejä käytettiin kehityksen tukena varmistamaan kulloinkin kehityksen kohteena olevan ominaisuuden oikea toiminta. Järjestelmätestauksessa sovellusta testattiin kokonaisuutena virheitä ja puutteita etsien [4]. Hyväksymistestauksessa tilaaja varmisti sovelluksen täyttävän sille asetetut vaatimukset [8].

7.1 Yksikkötestaus

ALUnitTests-projekti sisältää yhteensä 54 testimetodia. Niiden koodikattavuus on Visual Studion testianalyysityökalun mukaan 61 % ALBackendin osalta ja 37 % ALManagerin osalta. ALGUI-projektia ja Touchsteria ei ole yksikkötestattu, sillä käyttöliittymien yksikkötestaus on tehotonta ja turhaa. 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.

Yksikkötestejä käytettiin pääasiassa testivetoisen kehityksen apuvälineinä. Testit kirjoitettiin ennen testattavan ominaisuuden kehittämistä ja niitä ajettiin sitä mukaa, kun kehitystyö eteni. Näin havaitut virheet voitiin korjata nopeasti ja samalla varmistettiin, että ominaisuus toimii halutulla tavalla.

7.2 Järjestelmätestaus

Syncsterille järjestettiin projektin aikana kolme järjestelmätestauskertaa. Kaikki testauskerrat suoritettiin projektihuoneen Windows 10 –tietokoneilla. Sovelluksesta käytettiin pääasiassa optimoitua release-käännöstä, mutta virheiden paikallistamiseksi saatettiin vaihtaa debug-käännökseen. Jokaisesta testauskerrasta laadittiin raportti, jonka liitteeksi lisättiin testauskerralla kerätty data ja käytetyt asetustiedostot. Testitapausten tilojen lukumäärät on esitetty Taulukko 1. Tarkemmat tulokset löytyvät järjestelmätestausraporteista [5][6][7].

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 26.4.2019 järjestelmätestaussuunnitelman version 0.3.0 mukaisesti. Android- ja EEG-moduulit jäivät testaamatta, sillä ensin mainittu ei ollut vielä valmis testattavaksi, ja jälkimmäisen tarvitsemaa EEG-kypärää ei saatu toimimaan. Sovelluksen virheenkäsittely ja validointi osoittautuivat monin paikoin puutteellisiksi, ja moni tärkeä ominaisuus oli vielä toteuttamatta.

Toisella testauskerralla 14.5.2019 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 24.5.2019, kun kaikki suunnitellut ominaisuudet oli saatu toteutettua. Testaussuunnitelma oli sama kuin edellisellä kerralla, mutta siitä poikettiin testaamalla sovellusta käyttöjärjestelmän eri kieliasetuksilla. Tämä oli aiheuttanut epäluotettavaa käytöstä välitallennustiedostojen lukemisessa hyväksymistestauksen aikana. Testauskerralla onnistuttiin paikantamaan lokalisointiongelmien syy, joka sitten korjattiin sovelluksen julkistettuun versioon.

Kolmannella järjestelmätestauskerralla havaitut *virheelliset* testitapaukset ja niiden perusteella suoritetut toimenpiteet on esitetty Taulukko 2. *Huomatukselliseen* johtopäätökseen päädyttiin testitapauksessa 3.2b, joka koski laitteiden omien sovellusten muodostamien CSV-muotoisten tiedostojen tuomista. Huomautus johtui siitä, että kyseinen toiminto on toteutettu vain D-Lab-datalle.

Koodi	**Testitapaus**	**Toimenpide**
2.5b	Tallennusta ei voida aloittaa, jollei vähintään yhtä dataosiota ole valittuna.	Sovittiin tilaajan kanssa, että tämä jätetään korjaamatta.
3.2c	Käyttäjälle näytetään virheilmoitus, jos sovellukselle syötetty tiedosto ei sovellu sovelluksen käyttöön, tai tiedoston sisältämää dataa ei voida synkronoida muuhun dataan.	Korjattiin tarkentamalla tiedoston sisällölle tehtävää tarkastusta.
6.1	Sovellus varoittaa verkkoyhteyden katkeamisesta ja yrittää yhdistää uudelleen mikäli tarpeen.	Korjattiin silmäliikekameramoduulin osalta. EEG-moduulin kohdalla jätetään korjaamatta, sillä uudelleenkäynnistys sekoittaisi aikaleiman laskennan. Android-moduuli jätetään jatkokehitykseen. Ajosimulaatiomoduuli toimii halutulla tavalla.

Taulukko 2: virheelliset testitapaukset ja tehdyt toimenpiteet.

7.3 Hyväksymistestaus

Hyväksymistestaus suoritettiin 17.5.2019 ajosimulaatiolaboratoriossa aitoa koetilannetta mukaillen. Testauksen suorittivat tilaajan edustajat projektiryhmän jäsenten toimiessa valvojina, koehenkilöinä ja teknisenä tukena. Testauksen tulokset löytyvät hyväksymistestausraportista [9].

Hyväksymistestauksen perusteella sovellus täytti tilaajan tarpeet. Käyttöliittymässä havaittiin muutamia mitättömiä puutteita, jotka heikensivät sovelluksen käyttömukavuutta. Puutteet korjattiin testauksesta saadun palautteen perusteella.

D-Lab toimi testauskerran aikana ailahtelevasti, eikä datan tuomista pystytty testaamaan. Datan tuominen kuitenkin hyväksytettiin tilaajalla jälkikäteen. Samalla havaittiin lokalisointiin liittyviä ongelmia, jotka sitten testattiin täsmennetysti kolmannessa järjestelmätestauksessa.

7.4 Testauksen tuloksien 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 luotettavia keinoja ajallisen synkronoinnin testaamiseen. Tarkkuutta testattiin muun muassa metronomin tahdittamana siten, että toinen testaaja paineli puhelimen näyttöä toisen vedellessä samaan aikaan rattipolkimen liipaisimia. Synkronointia koestettiin myös silmäliikekameran ja puhelimen avulla. Testaaja sulki silmänsä samalla, kun kosketti puhelimen näyttöä. Nämä testauskerrat antoivat alustavaa näyttöä siitä,

että synkronointi onnistuu alle vaaditun 100 millisekunnin tarkkuuden. Ne olivat kuitenkin aivan liian lyhyitä perustavanlaatuisten johtopäätösten tekemiseen. Synkronoinnin täsmällisyyttä olisikin hyvä testata lisää ja täsmällisemmin.

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 jatkokehityksessä.

8 Tavoitteiden toteutuminen

Sovellusprojektin lopputuloksena oli toimiva sovellus, joka sai tilaajan hyväksynnän. Tärkeimmät vaatimukset saatiin toteutettua, eikä sovelluksessa ole merkittäviä puutteita. Osa toteutusratkaisuista 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.

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 kanonisille 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 huhtikuun lopulla, jolloin projekti oli ehtinyt jo loppusuoralle. Siinä vaiheessa ei enää ollut aikaa suunnitella ja toteuttaa kokonaan uusia ominaisuuksia.

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 helppoa ja suoraviivaista jatkokehittäjälle. Tavoitteen ei voida katsoa täyttyneen täysin tyydyttävästi, sillä kuten teknisestä ohjeesta [17] 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`-putkikomponentti, joka hoitaa rivien lukemisen ohella myös yhteyden käynnistykset, lopetukset ja virheiden käsittelyt. EEG- ja Android-moduulit puolestaan lukevat tietyn määrän tavuja puskuriin, joten ne joutuvat kumpikin toteuttamaan itse oman puskurin lukemisen.

`HamsterDataSet`-luokka 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 päälleliimattuja viritelmiä, jotka palvelevat vain tiettyä moduulia. Tällainen on esimerkiksi `Exporter`-luokan `AdjustETTimestamps`-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 Toteutusratkaisujen kehittyminen

Projektiryhmä suunnitteli ja kehitti alusta alkaen kaksi täysin uutta sovellusta, joten toteutusratkaisuja pohdiskellessaan ryhmällä oli paljon vapauksia ja liikkumavaraa. Kehitystyö alkoi prototyypittelyvaiheella, jolloin tutustuttiin laitteisiin ja niiden lähettämiin datavirtoihin, ja kokeiltiin alustavia moduuliratkaisuja. Putkikomponentti ja sen käyttämät tietotyypit todettiin jo varhain toimiviksi ideoiksi, joten ne pysyivät pääasiassa muuttumattomina projektin alusta lähtien.

Välitallennusten tarve ilmeni, kun D-Labin AOI-dataa ei saatu reaaliajassa lähetettyä muun silmäliikedatan kanssa. Datan tuominen jälkikäteen tarkoitti sitä, että sovelluksen tuli osata käsitellä aiemmin tekemäänsä tallennusta. Jotta välitallennusten tekeminen olisi mahdollisimman nopeaa, päätettiin jokaisen moduulin keräämä data kirjoittaa omiin tiedostoihinsa. Jälkikäsittelyn joustavuutta pyrittiin parantamaan kirjoittamalla välitallennustiedostot sellaiseen muotoon, josta data voidaan palauttaa takaisin alkuperäisiin tietotyyppeihin.

Sopivan sarjallistamisratkaisun löytäminen vei aikansa. Alkuperäinen toteutus käytti XML-sarjallistusta, mutta sen tuottamat tiedostot paisuivat valtavan suuriksi. Pienempiä tiedostokokoja saatiin aikaan binäärisarjallistuksella. Se kuitenkin osoittautui hyvin hitaaksi pitempien tallennuksien yhteydessä. 10-minuuttisen tallennuksen lukeminen tiedostoista vei aikaa lähes saman verran kuin itse tallennuksen kesto. Lopulta ryhmä päätyi kehittämään oman sarjallistusratkaisunsa, joka mahdollisti nopeamman tallennusten kirjoittamisen ja lukemisen pienemmillä tiedostoilla.

Graafisen käyttöliittymän tarve oli alun perin määritelty hyvin pieneksi, joten sen kehittäminen jätettiin projektin alussa taka-alalle. Sovellus kuitenkin osoittautui toiminnoiltaan sen verran monimutkaiseksi, että sen käyttämiseen tarvittiin selkeää ja helppokäyttöistä käyttöliittymää. Käyttöliittymä eli sitä mukaa, kun uusia toimintoja lisättiin, ja tilaajan ja projektiryhmän väliset näkemyserot kapenivat.

8.4 Toteutuksen haasteet

Sovelluksen kehityksessä jouduttiin huomioimaan monia asioita. Laitteissa ja niiden ohjelmistoissa oli merkittäviä eroja, joten jokainen laitemoduuli vaati omia erityisratkaisujaan. Laitteet olivat osittain uusia myös tilaajallekin, joten projektiryhmä joutui myös testaamaan niiden toimintaa oman sovelluksen ohella. Moni asia selvisi yrityksen ja erehdyksen kautta. Datan tuominen jälkikäteen monimutkaisti sovellusta entisestään.

Androidin kehitysympäristö muuttuu kiivasta tahtia versionumeroiden päivittyessä ja tietosuojarajoitteiden tiukentuessa. Tämä johti siihen, että moni aiemmin toiminut mallikoodinpätkä oli jo ehtinyt vanhentua, eikä niistä ollut hyötyä Touchsterin kehityksessä.

Ryhmän jäsenten C#-taidot olivat projektin alussa vähän ruosteisia, joten asioiden opettelu vei aikansa. Oman haasteensa toi myös graafisen käyttöliittymän alustavan tarpeen alimitoitus. 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 [17].

9.1 Sovelluksen virheiden korjaaminen

Syncsterin ydinominaisuudet on hyvin testattu ja toimiviksi todettu. Sovelluksen virheenkäsittely on varsin kattavaa, eikä sitä saa kovinkaan herkästi kaatumaan, varsinkin jos sovellusta käyttää siivosti käyttöohjetta noudattaen. Joissain tilanteissä sovelluksen toiminta on kuitenkin käyttäjän kannalta epäloogista, ja jopa virheellistä. Alaluvussa luetellaan epäkohtia, joihin tulisi jatkokehityksessä kiinnittää huomiota.

Android-moduuli ei yritä automaattisesti yhdistää uudelleen sen jälkeen, kun yhteys on kerran katkennut. Uuden yhteyden muodostaminen vaatii kaikkien muidenkin laitemoduulien yhteyksien katkaisemista ja uudelleenkäynnistämistä.

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.

Käyttäjä voi aloittaa uuden tallennuksen viemisen ennen edellisen viemisen päättymistä. Tämä sekoittaa viemisen edistymistä mittaavan palkin. Jos käyttäjä vie saman tallennuksen kahteen kertaan, saattaa toinen vientioperaatio keskeytyä tiedostolukitukseen.

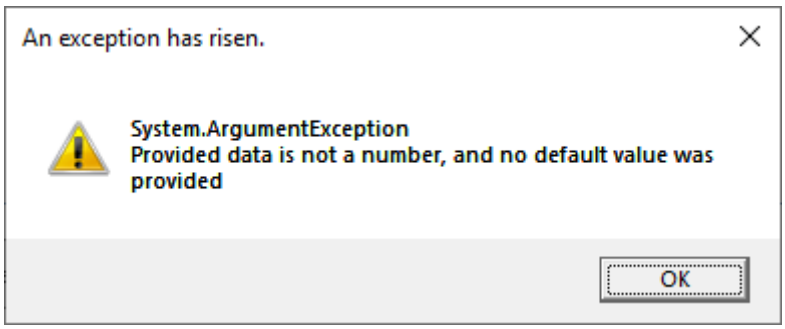
9.2 Olemassa olevien ominaisuuksien kehittäminen

Luvussa 8.2 mainittua kovakoodausta pitäisi pyrkiä vähentämään. Nykyinen `SerializableSettings`-asetusolio 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 datoja 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 eniten muistia vievien operaatioiden jälkeen. Parempi ratkaisu olisi `IDisposable`-rajapinnan laajempi käyttö. Tämän hetkisistä moduuleista vain ajosimulaatiomoduuli toteuttaa kyseisen rajapinnan.

Käyttäjälle esitettävät virheilmoitukset eivät useinkaan kuvaa ongelmatilanteita kovinkaan selkokielisesti. Esimerkiksi käyttäjän tuodessa tiedoston, jonka arvot eivät vastaa oletustyypitystä, sovellus antaa Kuva 9.1 mukaisen teknisen virheilmoituksen. Viestien esittämiseen käytettyjä modaalisia ikkunoita voitaisiin korvata käyttäjäystävällisemmällä ratkaisulla. Voitaisiin myös kehittää jokin ratkaisu laitemoduulien ja managerin lähettämien lokiviestien esittämiselle käyttöliittymässä



Kuva 9.1: Sovelluksen esittämä virheilmoitus.

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. `ETFileReader`-luokkaa 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.

9.3 Uusien ominaisuuksien kehittäminen

Vaatimukset 8.6 ja 8.7 käsittelevät silmäliikekameran D-Lab-ohjelmiston etäkäyttöä [1]. 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. Projektiryhmällä ei ollut aikaa suunnitella, kuinka ominaisuus integroitaisiin osaksi nykyistä sovellusta, mutta lyhyt prototyypittely ainakin osoitti etäkäytön toimivan.

AOI-datan automaattiseen lähettämiseen joko reaaliajassa tai jälkikäteen ei löytynyt ratkaisua projektin aikana. Se saattaa olla mahdollista, jos tilaajan 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 toteutuskelpoisuudesta.

Vaatimus 1.3 tunnuslukujen laskemisesta datasta jätettiin toteuttamatta [1], sillä tilaaja katsoi voivansa suorittaa tarvittavat tilastotieteelliset laskutoimenpiteet käsittelemällä tulostiedostoa muilla työkaluilla. Jos tunnuslukuja halutaan kuitenkin jatkossa laskea Syncsterillä, voidaan tarvittavia metodeja lisätä `CollectedDataExtensions`-luokkaan, ja kutsua niitä datan jälkikäsittelyvaiheessa. Tämä tapa tosin soveltuu lähinnä yksittäisen moduulin datan käsittelyyn. Yhteisten tunnuslukujen laskeminen useammasta moduulista vaatisi todennäköisesti uusien komponenttien kehittämistä, ja käyttöliittymästäkin pitäisi varata niiden esittämiselle jokin tila.

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 antaman datan yhteen tulostiedostoon.

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 laadittiin useita yksikkötestejä, kolme järjestelmätestauskertaa ja yksi hyväksymistestauskerta. Projektin tekninen ohjaaja tarkasti koodin kaksi kertaa. Sovellus sai hyväksynnän niin tilaajalta kuin tekniseltä ohjaajaltakin.

Lähteet

[1] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, "Peltihamsteri-projekti, Vaatimusmäärittely", Jyväskylän yliopisto, informaatioteknologian tiedekunta, 11.6.2019.

[2] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, "Peltihamsteri-projekti, Projektiraportti", Jyväskylän yliopisto, informaatioteknologian tiedekunta, 10.6.2019.

[3] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, "Peltihamsteri-projekti, Rakennekuvaus", Jyväskylän yliopisto, informaatioteknologian tiedekunta, 12.6.2019.

[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, Järjestelmätestausraportti", Jyväskylän yliopisto, informaatioteknologian tiedekunta, 26.4.2019.

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

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

[8] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, "Peltihamsteri-projekti, Hyväksymistestaussuunnitelma", Jyväskylän yliopisto, informaatioteknologian tiedekunta, 25.5.2019.

[9] Tuomo Kujala, "Peltihamsteri-projekti, Hyväksymistestausraportti", Jyväskylän yliopisto, informaatioteknologian tiedekunta, 17.5.2019.

[10] Jukka-Pekka Santanen, ”Tietotekniikan sovellusprojektien ohje”, saatavilla PDF-muodossa http://www.mit.jyu.fi/palvelut/sovellusprojektit/projohje.pdf, Jyväskylän yliopisto, informaatioteknologian tiedekunta, 30.1.2017.

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

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

[13] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, ”Syncster, User manual”, Jyväskylän yliopisto, informaatioteknologian tiedekunta, 11.6.2019

[14] 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.

[15] Nicolò Carandini et al., ”Unit testing C# with MSTest and .NET Core” https://docs.microsoft.com/en-us/dotnet/core/testing/unit-testing-with-mstest, Microsoft 8.9.2017

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

[17] Mari Kasanen, Leevi Liimatainen, Marina Mustonen, Juhani Sundell ja Arttu Ylä-Sahra, ”Peltihamsteri, Technical manual”, Jyväskylän yliopisto, informaatioteknologian tiedekunta, 10.6.2019.