TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Taavi Sutt 142600IAPB

MOODUL TTÜ100 TUDENGISATELLIIDI ELEKTRITOITESÜSTEEMILE UUE TARKVARA LAADIMISEKS

Bakalaurusetöö

Juhendaja: Evelin Halling
MSc

Tallinn 2018

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Taavi Sutt

21.05.2018

Annotatsioon

Töö eesmärgiks on luua moodul TTÜ100 tudengisatelliidi elektritoitesüsteemile uue tarkvara laadimiseks. Moodul arendati missioonijuhtimistarkvara osana.

Mooduli koosneb kasutajaliidesest ja *back-end* poolest.

Töös kirjeldatakse missioonijuhtimistarkvara arhitektuuri, elektritoitesüsteemi, mooduliga seotud nõudeid, realisatsiooni, testimist ja edasist tööd.

Töö tulemusena valmib moodul, millega on võimalik elektritoitesüsteemile laadida uus tarkvara ja lahendust testitakse prototüübi abil.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 25 leheküljel, 8 peatükki, 15 joonist, 2 tabelit.

Abstract

Module for updating TTU100 student satellite's Electrical Power Supply software

TTU100 student satellite's project was introduced in the autumn of 2014. Its main objective is to develope a nanosatellite and deploy it to orbit. Also to create a ground station from which the satellite mission could be controlled. The project offers students experience in working with technologies related to space missions.

Work on the mission control software began in 2017 when the architecture was planned. Mission control software includes many different functionalities of which one are activites related to Electrical Power Supply. At the moment, mission control software lacks the ability to update Electrical Power Supply's software.

The main purpose of this thesis is to develope a module for updating TTU100 student satellite's Electrical Power Supply software. The module will be a component of mission control software.

Thesis will give a general overview of the mission control software architecture. It will describe Electrical Power Supply and its communication protocol. Also address the requirements which need to be considered for developing the module. Main focus of the thesis will be on giving the overview of how the module was created and tested.

As a result a working module will be created from which the software can be updated. The solution will be tested on a prototype.

The thesis is in Estonian and contains 25 pages of text, 8 chapters, 15 figures, 2 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, „programmiliides mille abil saab liidestada tarkvaraga uusi komponente“ [1, p. 5].
EPS	*Electrical Power Supply*, elektritoitesüsteem, vastutab satelliidi akude ja elektri eest.
GitLab	Keskkond Git repositooriumitele.
HDLC	*High-Level Data Link Control*, „bitipõhise piiratud kaadriga andmeside standardprotokoll“ [2].
REST	*Representational State Transfer*, veebiteenuste loomistel kasutatav tarkvaraarhitektuuri stiil.
SPA	*Single Page Application*, „rakendus, kus kogu kasutajapoolne rakenduse osa on laetud brauserisse ühe lehe laadimisega“ [1, p. 5].
TLE	*Two-line element set*, „kaherealine andmeformaat, objekti asukoha kirjeldamiseks kosmoses“ [1, p. 5].

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

TTÜ100 tudengisatelliidi programmiga alustati 2014. aasta sügissemestril. Projekti eesmärgiks on valmistada nanosatelliit ja see orbiidile lennutada.Satelliidi missiooni juhtimiseks tuleb luua ka maajaam. Programm pakub tudengitele võimalust osaleda kosmoseprojektis, kus omandatakse kosmosetehnoloogia valdkonnaga seotud teadmisi ja kogemusi. [3]

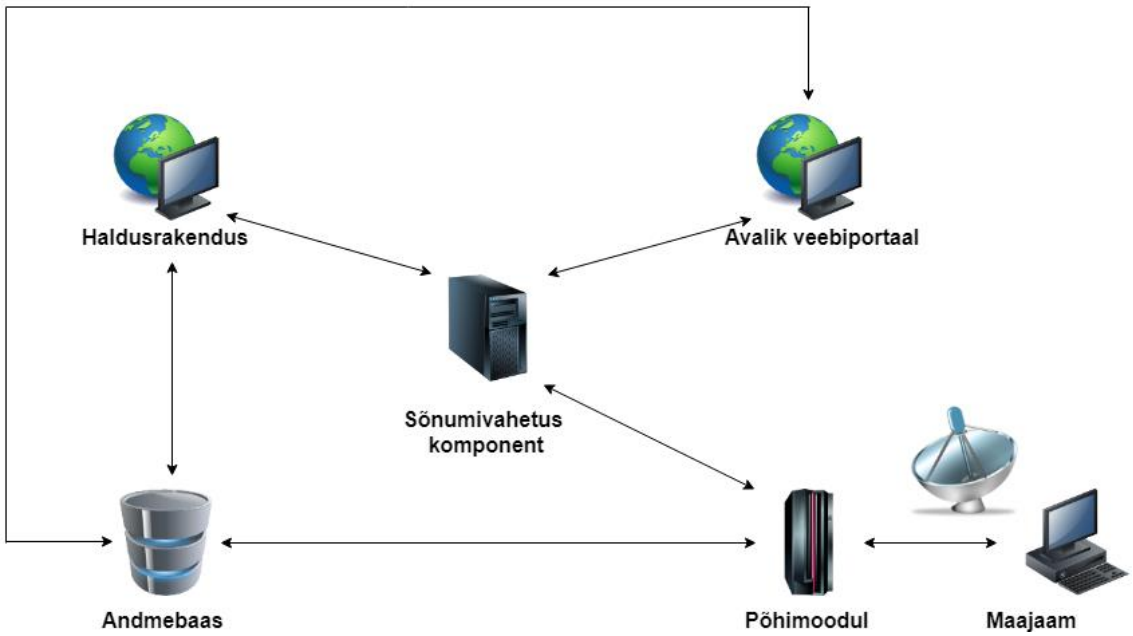
Missioonijuhtimistarkvara loomist TTÜ100 tudengisatelliidile alustati 2017 aastal, kui Siim Romanovi tegi sellest oma magistritöö ja pani paika tarkvara arhitektuuri. Missioonijuhtimistarkvara sisaldab palju erinevaid funktsionaalsusi nagu näiteks satelliidile käskude saatmist, satelliidi poolt tehtavate piltide alla laadimist, elektritoitesüsteemi olukorra jälgimist. Elektritoitesüsteem vastutab satelliidi elektriga varustamise eest. Kuna missiooni kestvuseks on kavandatud paar aastat. Seega võib tekkida olukord, kus elektritoitesüsteemile on vaja missiooni vältel üleslaadida uus tarkvara. Missioonijuhtimistarkvaras puudus praeguseni võimalus elektritoitesüsteemile uut tarkvara laadida.

Käesoleva töö eesmärgiks on luua missioonijuhtimistarkvarasse moodul elektritoitesüsteemile uue tarkvara laadimiseks. Mooduli tegemisel tuleb arvestada elektritoitesüsteemist tulenevate piirangutega, kui ka teistest moodulile püstitatud nõuetest.

Töö teises peatükis kirjeldatakse missioonijuhtimistarkvara arhitektuuri. Kolmandas peatükis räägitakse elektritoitesüsteemist ja selle kommunikatsiooni protokollist. Neljandas peatükis kirjeldatakse loodava mooduli nõudeid. Viies peatükk keskendub mooduli realisatsioonile. Kuuendas peatükis räägitakse, kuidas testiti loodud lahendust, missugused probleemid testimisel ilmnesid ning kuidas need lahendati. Viimases peatükis räägitakse edasisest tööst mooduliga.

2 Missioonijuhtimistarkvara arhitektuur

TTÜ100 tudengisatelliidi missioonijuhtimistarkvara arhitektuuri töötas välja TTÜ tudeng Siim Romanov enda magistritöö raames, 2017. aastal. Töö lõpptulemusena lõi ta arhitektuuri, mis koosneb neljast suurest komponendist: andmebaasist, sõnumivahetuskomponendist, haldusrakendusest ja põhimoodulist. Arhitektuurile lisandub ka viies suur komponent, avalik veebiportaal. Avalik veebiportaal on loomisel teise tudengi poolt. Joonisel nr 1 on kujutatud missioonijuhtmistarkvara arhitektuuri.



Joonis 1. Missioonijuhtimistarkvara arhitektuur [1, p. 24]

2.1 Andmebaas

Missioonijuhtimistarkvara kasutab andmete hoidmiseks PostgreSQL andmebaasi. Arhitektuuri looja valis PostgreSQL andmebaasi, kuna tal oli sellega varasemalt rohkem kogemusi, kui alternatiivse valikuga, milleks oli MySQL andmebaas. Andmebaasimuudatuse haldamiseks on kasutusel Liquibase[1] tarkvara. Arhitektuuri looja

[1] http://www.liquibase.org/

põhjendab magistritöös [1, p. 25] selle tarkvara valikut järgmiselt: „Liquibase koos Git versioonihaldustarkvaraga võimaldab jälgida andmebaasi muudatuste ajalugu ning rakendada neid muudatusi erinevates keskkondades ühtemoodi.“ Andmebaasis on andmed eraldatud moodulite järgi eraldiseisvatesse andmebaasi skeemidesse. [1, p. 25]

2.2 Sõnumivahetuskomponent

Sõnumivahetuskomponendina on missioonijuhtimistarkvaras kasutusel ActiveMQ[1] tarkvara, mis valiti teiste alternatiivide seast, kuna vastas arhitektuurile püstitatud nõuetele. Samuti ka seetõttu, et teised projekti komponendid on loodud Javas. ActiveMQ kasutades on projektis vajaminevate tehnoloogiate hulk väiksem kui mõne teise alternatiivi korral. [1, p. 28]

Sõnumivahetuskomponendi ülesandeks on vahendada ja koordineerida sõnumite liiklust saatvate ja tarbivate osapoolte vahel [1, p. 26]. Kõik satelliidile saadetavad ja satelliidilt vastuvõetavad sõnumid läbivad komponenti ning vastavalt sõnumi sisule jõuab sõnum õige süsteemini.

2.3 Haldusrakendus

Haldusrakendus koosneb kahest komponendist: kasutajaliidesest ja *back-end*'ist. Rakendus on loodud SPA (*single page application*) veebirakendusena. Kasutajaliides on implementeeritud kasutades AngularJS raamistikku. *Back-end*'i loomisel on kasutatud Spring Boot raamistikku, mis on üks enamlevinuid Java raamistikke. [1, p. 29]

Haldusrakendust hakkavad kasutama missioonijuhtimist läbiviivad kasutajad. Selle abil toimub ainult missioonijuhtimistarkvara töö juhtimine. Arhitektuuri kirjeldavas magistritöös [1, pp. 28-29] on kasutajaliidese funktsionaalsust kirjeldatud järgmiselt: „Loodud kasutajaliidese kaudu on võimalik:

- Vaadata viimaseid ja järgmiseid ülelende;
- Vaadata ajas satelliidi asukohta kaardil;

[1] http://activemq.apache.org/

• Vaadata satelliidi telemeetria andmeid;

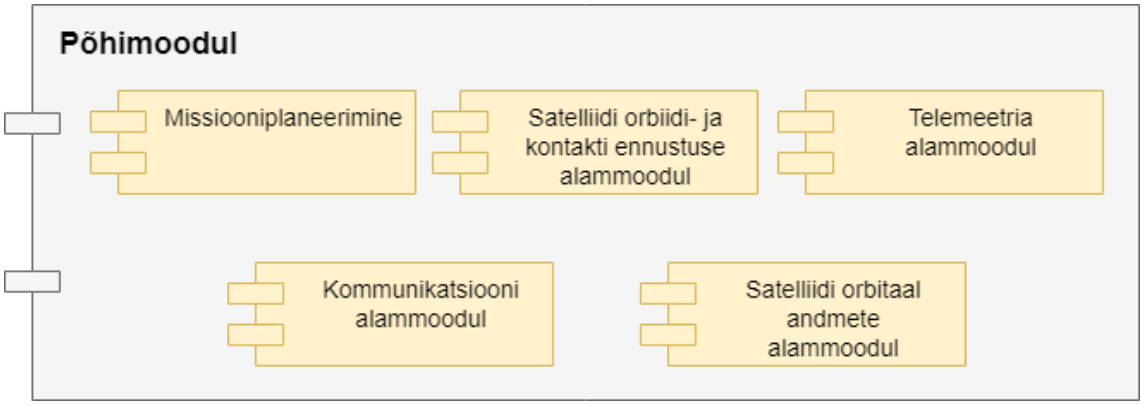
• Planeerida järgimisi missioone ja saata käsklusi satelliidile;

• Vaadata alla laetud andmefaile;

• Hallata rakenduse kasutajaid. „

2.4 Põhimoodul

Missioonijuhtimistarkvara kõige tähtsam osa on põhimoodulil, mis omakorda jaguneb mitmeks erinevaks alammooduliks (Joonis 2). Omavahel suhtlevad alammoodulid peamiselt sõnumivahetuskomponendi vahendusel. [1, p. 31]



Joonis 2. Põhimooduli komponendid [1, p. 31]

2.4.1 Missiooniplaneerimine

Missioonijuhtimistarkvara arhitektuuri magistritöös [1, p. 31] kirjeldatatakse missiooniplaneerimise moodulit järgmiselt: „Missioonijuhtimistarkvara üheks keskseks komponendiks on missiooniplaneerimise moodul. Teisi missioonijuhtimistarkvara komponente ja mooduleid võib vaadata, kui tugiteenuseid, mille abil missioone planeerida.“

Käskluse planeerimiseks satelliidi ülelennu ajaks on vajalik saada andmeid teistelt moodulitelt, mis tegelevad näiteks satelliidi telemeetria ja akude seisukorra andmetega. Saadud andmetest lähtuvalt langetatakse otsus, milline käsk on kõige mõistlikum satelliidile anda. [1, p. 32]

2.4.2 Satelliidi orbiidi- ja kontaktiennustuse alammoodul

Arhitektuuri magistritöös [1, p. 32] kirjeldatakse antud moodulit järgmiselt: „Satelliidi orbiidiennustus- (satellite orbit propagation) mooduli peamiseks ülesandeks on arvutada välja järgmise kontakti täpne aeg ja orbiit. Saadud tulemus salvestatakse andmebaasi ning edastatakse sõnumivahetuse komponenti.“

Mooduli andmeid läheb vaja peamiselt maajaamas. Nende andmete abil tagab maajaam, et antennid oleksid võimelised satelliidilt saabuvaid raadiosignaale vastu võtma ja neid omakorda ka satelliidile edastama. Samuti kasutatakse neid andmeid missiooniplaneerimise moodulis järgmiste sideseansside planeerimiseks. [1, p. 31]

2.4.3 Telemeetria alammoodul

Telemeetria moodulit on kirjeldatud arhitektuuri magistritöös [1, p. 33] järgmiselt: „Telemeetria mooduli eesmärgiks on kirjeldada telemeetria sõnumite sisu ning dekodeerida see vastavalt iga satelliidi alamsüsteemi spetsifikatsioonile. Igal satelliidi alamsüsteemil on kindlaks määratud registrid, mis hoiavad infot konkreetse süsteemi toimimise kohta.“

Satelliidilt sõnumivahetuskomponenti jõudnud telemeetria pakett saadetakse edasi telemeetria moodulisse, mis tuvastab telemeetria liigi paketi päise järgi. Järgnevalt dekodeeritakse see andmebaasis oleva paketi kirjelduse abil ning tulemus salvestatakse andmebaasi. Tulemus saadetakse ka sõnumivahetuskomponendile, et kõikidel telemeetriast huvitatud moodulitel oleks võimalus reaalajas jälgida ja teostada nende andmete abil neile määratud tegevusi. [1, pp. 33-34]

2.4.4 Kommunikatsiooni alammoodul

Kommunikatsiooni alammoodulit on arhitektuuri magistritöös [1, p. 35] kirjeldatud järgmiselt: „Kommunikatsiooni alammoodul omab ühte kõige suuremat tähtsust missioonijuhtimistarkvara põhimoodulis, kuna see realiseerib süsteemi kommunikatsiooniprotokolli dokumendis [30] kirjeldatud L3/L4 taseme protokolli. Kui haldusrakenduses on võimalik kõrgetasemeliselt ära kirjeldada satelliidile saadetavad käsud, siis kommunikatsioonimoodul kodeerib käsud vastavalt protokollile, edastab vastavalt kodeeritud paketid maajaamale ning juhib andmeside seanssi satelliidi ja maajaama vahel. “

2.4.5 Satelliidi orbitaalandmete alammoodul

Missioonijuhtimiseks on missioonijuhimistarkvara komponentidel ja lõppkasutajatel vaja andmeid satelliidi hetke asukoha kohta. Neid andmeid teades on võimalik visualiseerida satelliidi asukohta kaardil ning teha arvutusi järgmise kontakti ennustamiseks. [1, p. 36]

Veebilehti, mis jälgivad orbiidil olevaid satelliite on mitmeid. Antud alammoodulis on selleks valitud Spacke-track.org[1], kust on võimalik TLE (*two-line element set*) formaadis andmeid alla laadida. Kindla ajaperioodi tagant moodul käivitub ning laeb veebilehelt alla andmed satelliidi asukoha kohta. Pärast andmete alla laadimist need dekodeeritakse, salvestatakse andmebaasi ja edastatakse sõnumivahetuskomponendile. [1, p. 37]

2.5 Avalik veebiportaal

Avaliku veebiportaali eesmärgiks on pakkuda kõikidele TTÜ100 satelliidi huvilistele võimalust missiooni jälgimiseks. Teiseks eesmärgiks võimaldada liidestust satelliidiprojekti välistele osapooltele nagu näiteks raadioamatöörid. [1, p. 30]

Avalik veebiportaal koosneb API-st ja kasutajaliidesest. API ülesandeks on võimaldada kolmandate osapoolte liidestumine missioonjuhtimistarkvaraga. Projekti välistel osapooltel on API abil võimalik pääseda ligi missiooni andmetele, mis on mõeldud avalikult kasutamiseks. Raadioamatööridel on võimalus ise saata missioonijuhtimistarkvarale nende poolt kinni püütud raadiosignaalis sisalduvaid satelliidi telemeetria andmeid. Turvalisuse huvides koheldakse sellised andmeid süsteemi poolt kui mitte usaldusväärsest allikast pärit andmeid. Kasutajaliidesest kuvatakse andmeid sama API liidese abil. [1, p. 30]

Avalikule veebiportaalile on lisandunud veel üks nõue. Teaduse arendamiseks ja populariseerimiseks Eesti noorte seas on ette nähtud, et huvitatud koolidel on võimalus ennast registreerida kasutajateks veebilehel https://kosmos100.ee. Õpilastel on võimalik enda tehtud antennide abil vastu võtta satelliidilt tulevaid signaale. Antennide ehitamiseks koostavad Tallinna Tehnikaülikool ja Tartu Ülikool õppematerjalid. Mängulisuse

[1] https://www.space-track.org

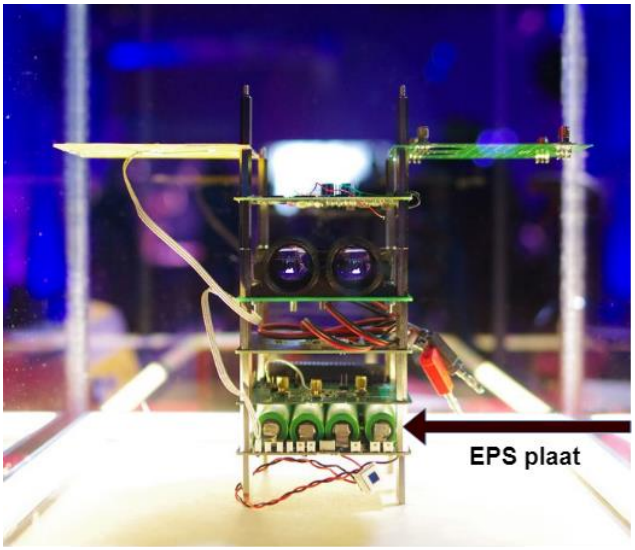
lisamiseks saavad koolid saata ka teistele koolidele signaale läbi satelliidi, mida veebiportaalis kajastatakse ja luuakse edetabeleid.

3 Elektritoitesüsteem

Satelliidil paikneva elektritoitesüsteemi (*EPS*) ülesandeks on elektri kogumine päikesepaneelidelt, salvestamine akudesse ja erinevate satelliidi alamsüsteemide elektriga varustamine. Elektritoitesüsteem koosneb 374 komponendist, mille hulgas on kaitselülitused, telemeetri andurid ning mikrokontroller, mis kõike haldab. [4, p. 18]

Andurite vahendusel kogutud telemeetriat edastatakse maajaamale. Näiteks kui palju voolu on akudes ning teavet akude üldise seisukorra kohta. Saadud andmete abil planeeritakse maa peal missugune käsk on kõige otstarbekam satelliidile saata.

Joonisel nr 3 on näidatud elektritoitesüsteemi asukoht kuupsatelliidil.



Joonis 3. Elektritoitesüsteemi asukoht satelliidil

3.1 Elektritoitesüsteemi kommunikatsiooni protokoll

Satelliidi siinil (*satellite bus)* on elektritoitesüsteem alluvus (*slave*) suhtes ning ta saadab vastuseid ainult juhul, kui tema poole pöördutakse. Info vahetamine toimub HDLC (*High-Level Data Link Control*) pakettidega. Joonisel nr 4 on kujutatud standardse HDLC paketi üldist struktuuri.

Flag	Address	Control	Information	FCS	Flag
8 bitti	8+ bitti	8 või 16 bitti	n * 8 bitti	16 või 32 bitti	8 bitti

Joonis 4. Standard HDLC paketi struktuur

HDLC paketti ümbritsevad lipud (*flag*), mis annavad märku paketi algusest ja lõpust. Järgneb aadress osa (*address*), kus on märgitud, mis süsteemile on pakett mõeldud. Kontrolli (*control*) välja abil defineeritakse paketi tüüp. Andmeväljal (*information*) on edastatava paketi andmeosa. FCS väljal esitatakse paketi kontrollsumma, mis on arvutatud aadress-, kontroll- ja andmevälja põhjal. Kontrollsumma abil saab vastuvõtja tuvastada vigu, mis võivad tekkida paketi edastamisel. [5, p. 1, 6, p. 12]

Paketi andmete osana (*payload*) on TTÜ100 satelliidi projekti puhul kasutusel MODBUS-RTU[1] protokoll (avatud sideprotokoll (Modicon, 1979) elektroonikaseadmete vaheliseks andmevahetuseks jadasiini kaudu [7]). [5, p. 1]

Joonisel nr 5 on kujutatud elektritoitesüsteemile saadetava HDLC paketi sisu registrite väärtuste lugemise käsu näitel.

HDLC			Modbus style payload					HDLC		
Flag	Eps board address	CTRL byte	Modbus Command	Read register start address (HI)	(LOW)	Nr. of registers to read (HI)	(LOW)	CRC	CRC	FLAG

Joonis 5. Elektritoitesüsteemile saadetav HDLC pakett *read register* käsu näitel

[1] http://www.modbus.org/

4 Nõuded loodavale süsteemile

Järgnevas peatükis kirjeldatakse elektritoitesüsteemile uue tarkvara laadimise mooduli loomiseks vajalikke piiranguid ja nõudeid, mida tuleb realiseerimisel arvesse võtta.

4.1 Elektritoitesüsteemist tulenevad piirangud

Uue tarkvara üleslaadimiseks ootab elektritoitesüsteem uut tarkvara ridade haaval. See tähendab, et tarkvarafail tuleb saata ühe rea kaupa satelliidile. Enne järgmise rea saatmist tuleb oodata eelmisele reale vastust. Tarkvarafailis on read HEX (kuueteistkümnendsüsteem) kujul. Kuna on olemas oht, et üleslaadimise käigus võib minna mingi osa andmetest kaduma nii sisemiste kui ka väliste tegurite tõttu – pilved segavad satelliidile raadio signaali saatmist – tuleb pidada ka järge mitmes rida satelliidile saadeti. Vea tekkimisel tuleb see rida uuesti saata. Samuti tuleb järge pidada, millal kõik tarkvara failis olevad read on elektritoitesüsteemile saadetud, sest pärast neid tuleb edastada veel kogu faili andmete põhjal arvutatud kontrollsumma ja käsk uue tarkvara mällu kirjutamiseks.

Üleslaadimise puhul tuleb ka määrata, mis tüüpi käsuga on tegemist. Käskude tüüpe on viis ja igal ühel neist on oma kindel väärtus. HEX formaadis esinevad need järgnevalt:

- *0x0000 - üleslaadimise tühistamine.*
- *0x0001 - tarkvarafailist pärit andmerida..*
- *0x0002 - kogu faili kontrollsumma.*
- *0x00FF - üleslaadimise protsessi alustamine.*
- *0x0055 - uue tarkvara mällu kirjutamine*

HDLC paketis paiknevad need identifikaatorid tarkvarafaili andmeosa ees, andmeväljal Nende identifikaatorite abil rakendab elektritoitesüsteemi vajalike tegevusi uue tarkvara installeerimiseks.

4.2 Tarkvara failide asukoht

Uue tarkvara laadimiseks kasutatavad failid hakkavad paiknema TTÜ GitLab-i repositooriumis. Valmis missioonijuhtimistarkvara korral on seal olevad failid eelnevalt testitud testsüsteemi poolt, et vähendada võimalust vigase tarkvara laadimiseks elektritoitesüsteemile. Failide testsüsteemi loomine ei ole selle töö skoobis. Failide GitLab-i repositooriumist kättesaamise aga saab juba realiseerida.

Kasutajaliidesest failide kuvamiseks on vajalik faili nimi. Esialgse plaani järgi sisaldub faili nimes tarkvara versiooni identifikaator. Selle järgi on lihtne jälgida, mis versioon elektritoitesüsteemile on viimasena üleslaaditud. Samuti on vajalik saada kätte faili sisu, milles paikneb üleslaaditava tarkvara andmed.

4.3 Funktsionaalsed nõuded

Haldusrakenduses paiknevale uue tarkvara üleslaadimise komponendi kasutajaliidesele on püstitatud järgmised funktsionaalsed nõuded:

1. *Kasutaja näeb eelmise edukalt üleslaetud tarkvaraga seotud andmeid: tarkvara versioon, käsu andmise kuupäev ja kellaaeg, üleslaadimise lõpu kuupäev ja kellaaeg*
2. *Kasutaja saab valida rippmenüüst üleslaetava tarkvara faili.*
3. *Kasutaja ei saa pooleli oleva üleslaadimise protsessi ajal uut üleslaadimist alustada.*
4. *Kasutaja saab katkestada üleslaadimise protsessi.*
5. *Kasutajale visualiseeritakse üleslaadimise protsessi laadimisriba abil.*

Oluline on, et kasutaja teaks, milline tarkvara on viimasena paigaldatud ja millal, et aidata pidada järge üleslaadimiste üle. Samuti on oluline, et ei käivitataks mitut üleslaadimist samaaegselt, mis koormaks nii missioonijuhtimistarkvara kui ka satelliiti üleliigselt.

5 Realisatsioon

Realiseeritav moodul on missioonijuhtimistarkvara alamosa. Seetõttu tuli mooduli loomisel kasutada tehnoloogiad, millega on projekt loodud.

Kasutajaliidese pool on loodud kasutades Javascript-i teeki Angular 4[1] ja TypeScript-i[2]. *Back-end* pool on loodud Javaga. Mooduli tööks vajalikud andmed asuvad PostgreSql andmebaasi põhimooduli skeemis.

Mooduli loomist hõlbustas see, kasutusel olevate tehnoloogiatega on autor varasemalt kokkupuutunud. Ainult Angular 4 raamistiku kasutamist tuli töökäigus õppida.

5.1 Tarkvara failidele ligipääsemine

GitLab-i repositooriumis olevatele tarkvara faildele ligipääsemiseks on kasutusel GitLab4J-API[3]. Tegemist on Java-t kasutatavatele rakendustele mõeldud API-ga ning ligipääs privaatsetele repositooriumitele on lihtsam, kui läbi GitLab-i enda repositooriumi API. API kasutamiseks tuleb ette anda GitLabi serveri aadress ja privaatne tõend (*token*), mille saab lasta genereerida GitLab-i kasutajaliideses seadete alt. Joonisel nr 6 on näidatud loodud meetodit API-ga ühendumiseks.

```
private GitLabApi connectToGitLab() {
      return new GitLabApi("https://gitlab.cs.ttu.ee", "token");
}
```

Joonis 6. Meetod GitLab4J-API-ga ühenduse loomiseks

GitLab4J-API koosneb oma korda mitmest väiksemast API-st erinevate funktsionaalsustega. Failide kättesaamiseks kasutatakse kahte alam API-t: Repository API ja RepositoryFile API. Peale ühenduse loomist, tuleb kasutada Repository API-t. API *getTree* meetodile tuleb sisendiks anda numbrina *(integer)* soovitava projekti id, kus

[1] https://angular.io/

[2] https://www.typescriptlang.org/

[3] http://www.messners.com/#gitlab4j-api/gitlab4j-api.html

failid paiknevad. Meetod tagastab loendi (*list*) projektis olevatest failidest. Itereerides üle tagastatud väärtuse, võetakse loendis oleva objekti nimi ja lisatakse see koos teiste nimedega uude loendisse, mille meetod tagastab. Kogu kirjeldatud tegevus toimub loodud meetodi *getSoftwareFiles* sees (Joonis 7), mida kasutatakse kasutajale rippmenüüst tarkvara failide nimede kuvamiseks.

```
private List<String> getSoftwareFiles() throws GitLabApiException
{
        GitLabApi gitlabApi = connectToGitLab();
        List<String> files = new ArrayList<>();
        List<TreeItem> allFiles = gitlabApi
             .getRepositoryApi()
             .getTree(projekti number);
        for (TreeItem item : allFiles) {
            files.add(item.getName());
        }
        return files;
}
```

Joonis 7. Funktsioon tarkvara failide nimede kättesaamiseks

Kui kasutaja on rippmenüüst valinud faili nime on vajalik GitLabi repositooriumist kätte saada faili sisu. RepositoryFile API *getFile* meetodile tuleb sisendiks anda soovitava faili nimi, projekti id arvuna ja sõnena (*String*) parameeter *master*. Faili sisu on Base64 kodeeringus (“MIME kodeerimissüsteem, mis teisendab suvalise baidijada ASCII kirjamärkideks 64-märgilises alamhulgas (inglise suur- ja väiketähed, kümnendnumbrid, plussmärk, kaldkriips”) [8]) ning see tuleb dekodeerida tagasi esialgsele kujule. Kuna elektritoitesüsteemil paiknev komponent ootab tarkvara faili sisu reakaupa, siis salvestatakse sisu sõne tüüpi massiivi, mille kasutamine hõlbustab rea numbri jälgimist. Kogu kirjeldatud tegevus toimub loodud meetodi *getFileContent* sees (Joonis 8).

```
private String[] getFileContent(String fileName) throws
GitLabApiException
{
	GitLabApi gitlabApi = connectToGitLab();
	RepositoryFile file =
	gitlabApi.getRepositoryFileApi().getFile(fileName, projekti
	number, "master");
	byte[] decodedBytes =
	Base64.getDecoder().decode(file.getContent());
	String decodedString = new String(decodedBytes);
	return decodedString.split("\n");
}
```

Joonis 8. Funktsioon valitud tarkvara faili sisu saamiseks

5.2 Kasutajaliides

Missioonijuhtimistarkvara veebirakendus on loodud kasutades JHipster[1] generaatorit, mis kiirelt genereerib projekt, mille *back-end* põhineb Javal ja kasutab Spring Boot-i Kuna elektritoitesüsteemile uue tarkvara laadimise moodul paikneb missioonijuhtimistarkvaras, siis on selle realiseerimisel järgitud projekti kasutajaliidese disaini ja juba kasutatusel olevaid elemente.

Missioonijuhtimistarkvara kasutajaliides on nii inglis- kui eestikeelne. Lisas nr 1 on näidatud, kuidas käib liidese keele vahetamine. Mooduli realiseerimisel tuli liidese kakskeelsusega arvestada. Keele vahetamiseks tuleb deklareerida muutujad koos vastavate väärtustega JSON-i objektis. Keele seadistamiseks vajalikud JSON objektid paiknevad mitmes erinevas failis, kuid keelte põhjal on failid liigitatud eraldi projekti pakettidesse (*package*) vastavalt *et* (eestikeelne) ja *en* (ingliskeelne). Muutujad on mõlemas failis inglise keeles, kuid nendele omistav väärtus tuleb deklareerida vastavalt paketi keelele. Tabelis nr 2 on väljatoodud, kuidas on kakskeelselt kirjeldatud mooduli kasutajaliidese komponendid.

[1] https://www.jhipster.tech/

Tabel 1. Mooduli kasutajaliidese sisu

Mooduli kasutajaliidese ingliskeelne sisu *en/commanding.json*	Mooduli kasutajaliidese eestikeelne sisu *et/commanding.json*

"upload.software": {
"title": "Upload new software",
"start.uploading": "Start uploading",
"cancel": "Cancel",
"version": "Version",
"select": "Select software file to upload",
"last.info": "Last software information",
"start.date": "Started at",
"finished.date": "Finished at"
"uploading": "Uploading",
"cancelled": "Uploading cancelled",
"updated": "Uploading completed successfully"
}
	"upload.software": {
	"title": "Uue tarkvara laadimine",
	"start.uploading": "Alusta üleslaadimist",
	"cancel": "Peata",
	"version": "Versioon",
	"select": "Vali üleslaetav tarkvara fail",
	"last.info": "Viimase tarkvara informatsioon",
	"start.date": "Alustamise kuupäev",
	"finished.date": "Lõpetamise kuupäev"
	"uploading": "Üleslaadimine: ",
	"cancelled": "Üleslaadimine tühistatud",
	"updated": "Üleslaadimine edukalt lõpetatud"
	}

Kasutades JHipsteri jhiTranslate tõlkimise võimalust, tuleb liidese HTML failis ainult deklareerida soovitav muutuja ning vastavalt sellele, mis on liidesest valitud keel, muutub tekst kasutajaliideses. Joonisel nr 10 on näidatud, kuidas kuvatakse mooduli pealkirja kakskeelselt kasutajaliideses jhiTranslate'i abil.

```
<h2>
      <span jhiTranslate="commanding.upload.software.title">
      </span>
</h2>
```

Joonis 9. Mooduli pealkirja kuvamine kakskeelselt, kasutades jhiTranslate'i

Uue tarkvara laadimise mooduli saab kasutaja avada navigatsiooni ribast, Käsumooduli punkti alt, kust avaneb rippmenüü (Lisa 2). Mooduli avamisel avaneb kasutajale vaade (Lisa 3), kus ekraani vasakul servas asub valikkast (*select-box*), mille peale vajutades avaneb rippmenüü, GitLab-i repositooriumis asuvatest tarkvara failidest. Ekraani paremal pool, asub tabelis info viimasest edukalt üleslaetud tarkvarast.

Valikkasti all asub nupp üleslaadimise alustamiseks, kuid see muutub aktiivseks alles siis, kui on valitud üleslaadimiseks soovitud faili nimi. Nupule vajutades alustatakse üleslaadimise protsessi, samal ajal muutub nupp jälle mitteaktiivseks, et vältida uue üleslaadimise protsessi käivitamist. Lehe ülesse ilmub laadimisriba (Lisa 4), mis kuvab kasutajale, protsendiliselt, kui kaugel üleslaadimise protsess on. Alustamise nupu kõrvale ilmub üleslaadimise peatamise nupp (Lisa 4), mida vajutades protsess katkestatakse ning laadimisriba kaob. Protsessi lõpule jõudmisel teavitatakse kasutajat edukast üleslaadimisest, laadimisriba kaob, ning info tabelis uueneb.

Tarkvara failidega seotud andmete kätte saamiseks, ning protsesside ja info kuvamiseks, kastutatakse erinevaid REST päringuid, mis tegelevad andmete küsimisega mooduli *back-end* poolelt.

5.3 REST API

Tabelis nr 2 on toodud välja kõik REST API päringud, mida kasutavad kasutajaliides ja *back-end* rakendus omavaheliseks suhtluseks.

Tabel 2 REST API päringud

Operatisoon	Meetod	Ressurss
Gitlab-i Repositooriumis olevate failide kuvamine	GET	/software-files
Viimase edukalt üleslaetud tarkvara faili info kuvamine	GET	/software-files/lastVersion
Uue tarkvara üleslaadimise alustamine	POST	/software-files/upload
Tarkvara üleslaadimise peatamine	GET	/software-files/cancel
Üleslaadimise progressi kuvamine	GET	/software-files/progress
Üleslaadimise progressi edukalt lõpetamine	GET	/software-files/completed

5.4 Mooduli *back-end* pool

Back-end pool on loodava mooduli kõige tähtsam osa. See tegeleb REST-i päringutega, andmebaasi andmete salvestamise ja küsimisega, ning suhtleb sõnumivahetuskomponendiga. *Back-end* poolel koostatakse vajalik HDLC paket ja jälgitakse üleslaadimisprotessi kulgu.

5.4.1 Üleslaadimise alustamine

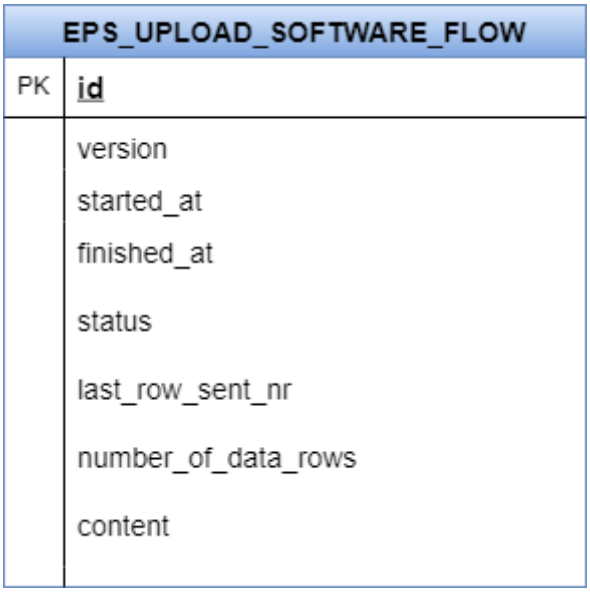
Andes kasutajaliidesest üleslaadimise alustamise käsu, kutsutakse välja back-end'is paiknev RESTI teenus. Teenuse tulemusena koostatakse HDLC pakett. HDLC pakettide koostamiseks oli juba missioonijuhtimistarkvaras olemas *BusCommand* klass. Üleslaadimise protsessi korral on aga iga üles saadetav sõnum erineva sisuga ning selleks implementeeriti *BusCommand* klassi kasutav alamklass *EPSSoftwareUploadCommand*.

Alamklassi konstruktorile antakse sisendiks tarkvarafaili rida ja üleslaadimise paketi tüüp, milleks on 00FF. Väljundina saadakse HDLC pakett ilma lippudeta. Koostatud pakett saadetakse sõnumivahetuskomponendile, kus lisatakse paketi alguse ja lõpu lipud. Joonisel nr 10 on näidatud HDLC paketti, lihtsustatud struktuuriga, mis saadetakse elektritoitesüsteemile üleslaadimise protsessi alustamiseks.

Flag	BA	Cntrl	Code	Addr	Type	Data	CRC	Flag
7E	64	3	10	03E5	00FF		17 E5	7E

Joonis 10. HDLC pakett üleslaadimise alustamiseks

Lisaks luuakse ka andmebaasis olevas tabelis *EPS_UPLOAD_SOFTWARE_FLOW* uus kirje. Tabelis olev rida iseloomustab ühte üleslaadimise protsessi. Andmebaasi salvestatakse üleslaadimise protsessi unikaalne id, tarkvarafaili versiooni nimi, algus ja lõpp kuupäev koos kellaajaga, üleslaadimise staatus, viimasena saadetud rea number, tarkvarafailis olevate ridade arv ja tarkvara faili sisu (Joonis 11).



Joonis 11. Andmebaasis oleva tabeli struktuur

5.4.2 Üleslaadimise protsess

Sõnumivahetuskomponendist saadetakse üleslaadimise käsk satelliidile. Satelliidil jõuab käsk elektritoitesüsteemini ja saadab maa peale sõnumi, et on valmis üleslaadimise protsessiks. Maapeal jõuab see sõnum jälle sõnumivahetuskomponenti. Komponendist

edastatakse sõnum *ResponseProcessor* klassile, mis töötleb vastuvõetavaid sõnumeid. Antud klass oli juba realiseeritud, kuid sinna tuli lisada funktsionaalsus, mis tuvastaks elektritoitesüsteemilt tulevad üleslaadimisega seotud vastused teistest sissetulevatest sõnumitest. Selleks lisati funktsionaalsus, mis kontrollib vastusena tuleva sõnumi sisu ja kui selgub, et vastus on seotud üleslaadimisega teavitatakse sellest klassi *EPSSoftwareUploader*. Klass loodi koos funktsionaalsusega, mis tegeleb tarkvarafaili üleslaadimisega.

EPSSoftwareUploader klassis asuva peamise funktsiooni tööpõhimõte seisneb selles, et andmebaasi tabelist küsitakse viimati lisatud kirje üleslaadimise kohta. Joonisel 12 on näidatud, mida andmebaasis olev tabel sisaldab pärast üleslaadimise käsu andmist.

	id [PK] uuid	version text	started_at timestamp without time zone	finished_at timestamp without time zone	status text	last_row_sent_nr integer	number_of_data_rows integer	content bytea
1	102e583f-7816-4f29-b51a-58414f7071d2	Software_V1	2018-05-10 23:45:40.954		Started	-1	1886	<binary data>
*								

Joonis 12. Tabelis olev kirje peale üleslaadimise käsu andmist

Funktsioonile vajalikud andmed on id, viimasena saadetud rea number, staatus ja faili andmed. Faili andmed on salvestatud sõnede massiivina. Massiivide indeksid algavad 0-ga ja seetõttu on viimasena saadetud numbri veerus väärtus -1. Võttes viimasena saadetud rea numbri, suurendades seda ühe võrra, saadakse number 0 ja selle abil küsitakse massiivist esimene tarkvarafaili rida.

Järgmisena tuleb saadetava rea andmed panna HDLC paketi sisse. *EPSUploadSoftwareCommand* alamklassi konstruktori sisenditeks on massiivis oleva rea sisu ja paketi tüübi identifikaator, 0001, mis näitab, et tegemist on üleslaadimise andmetega. Saadud HDLC pakett saadetakse sõnumivahetuskomponendile, mis edastab selle satelliidile. Id abil uuendatakse andmebaasis olevat viimasena saadetud rea numbrit ja staatust. Joonisel nr 13 on näidatud andmebaasi tabeli sisu pärast esimese tarkvarafaili rea saatmist.

	id [PK] uuid	version text	started_at timestamp without time zone	finished_at timestamp without time zone	status text	last_row_sent_nr integer	number_of_data_rows integer	content bytea
1	102e583f-7816-4f29-b51a-58414f7071d2	Software_V1	2018-05-10 23:45:40.954		Updating	0	1886	<binary data>
*								

Joonis 13. Tabeli sisu pärast esimese tarkvarafaili rea saatmist

Satelliidilt tuleva korrektse vastuse korral kordab funktsioon eespool kirjeldatud tööd kuni jõutakse faili lõpuni.

5.4.3 Üleslaadimise lõpetamine

Jõudes tarkvara faili viimase reani, kutsub peafunktsioon välja kontrollsumma arvutamise ja saatmisega tegeleva funktsiooni. Üleslaadimise edukuse kinnitamiseks, tuleb arvutada kontrollsumma kõikide tarkvarafailis olevate andmete üle ja saata see eraldi HDLC paketina elektritoitesüsteemile. Järjekordselt kasutatakse *EPSSoftwareUploadCommand* konstruktorit, kus seekord andmeosana antakse arvutatud kontrollsumma ja reatüübi identifikatoori väärtus, 0002, mis näitab, et tegemist on paketiga, millega saadetakse kontrollsumma. Loodud pakett saadetakse elektritoitesüsteemile jälle läbi sõnumivahetuskomponendi. Vastuse korral, mis näitab, et kontrollsumma oli õige tuleb veel viimasena saata pakett, mis ütleb, et tarkvara uuenduse võib elektritoitesüsteem endale lõplikult peale installeerida. Selleks kasutatakse järjekordselt sama HDLC paketi loomise viisi, lihtsalt andmeosa väljale ei lisata midagi ning rea identifikaatoriks märgitakse väärtus, 0055, mis näitab, et tegemist on käsuga, mis lubab tarkvara lõplikult uuendada. Andmebaasis märgitakse üleslaadimise protsess lõppenuks. Joonisel nr 14 on näidatud andmebaasi tabeli sisu, pärast üleslaadimise lõpetamist.

	id [PK] uuid	version text	started_at timestamp without time zone	finished_at timestamp without time zone	status text	last_row_sent_nr integer	number_of_data_rows integer	content bytea
1	102e583f-7816-4f29-b51	Software_V1	2018-05-10 23:45:40.95	2018-05-10 23:55:43.17	Completed	1886	1886	<binary data>
*								

Joonis 14. Tabeli sisu pärast üleslaadimise lõpetamist

5.4.4 Ebasobiv rida

Olukorras kui elektritoitesüsteemilt saabuv vastus annab teada, et saadetud pakett mingil põhjusel ei ole korrektne, tuleb viimasena saadetud rida uuesti edastada. Selleks kutsutakse välja peafunktsioon sisendiga *false*. Selle sisendi väärtuse korral ei suurendata andmebaasist saadud viimasena saadetud rea numbrit. Pakett koostatakse uuesti ning saadetakse sõnumivahetuskomponendile.

6 Testimine

Testimise võimalusi oli kaks: elektritoitesüsteemilt tulevate vastuste imiteerimine ja testimine elektritoitesüsteemi prototüübiga.

6.1 Elektritoitesüsteemilt tulevate sõnumite imiteerimine

Kõige lihtsam viis lahenduse testimiseks oli elektritoitesüsteemilt tulevate vastuste jäljendamine. Kuna missioonijuhtimistarkvara projekt on arendusfaasis on hetkel võimalik kirjutada sõnumeid otse ActiveMQ-sse ilma, et need peaksid läbima mingeid süsteemi komponente. Tuleb avada veebilehitseja, kuhu on sisestatud sõnumikomponendi paiknemise pordi aadress ja avanevasse rakendusse sisselogida. Rakendusest tuleb valida vastav teema (*topic*) kuhu sõnumeid saata.

Lahenduse testimiseks käivitati mooduli kasutajaliidesest üleslaadimise protsess. Süsteem edastab küll kõik sõnumid sõnumivahetuskomponendile, kuid sealt neid enam edasi ei saadeta. Teades millise sõnumiga elektritoitesüsteem vastab, kirjutatakse see läbi veebilehitsejast avanenud rakenduse otse ActiveMQ sisse tulevate sõnumite teemase. Sealt korjab selle vastuse jälle välja *ResponseProcessor* klass, mis edastab selle *EPSUploadSoftware* klassi meetodile.

Antud testimise abil saab jälgida, kas andmebaasi muudatusi teostatakse nii nagu oodati, ning kuidas käitub kasutajaliides. Samuti saab kontrollida, kuidas lahendus käitub, kui jäljendada vastust, et saadetud rida ei sobinud. Testimisele kulub vähe aega, kuna ei pea ootama kuni sõnumid jõuavad elektritoitesüsteemile ja sealt tagasi. Miinusteks sellisel testimisel oli, et sõnumid ei jõua kunagi sihtpunkti ja ei selgu, kuidas päris olukorras loodud lahendus töötab.

6.2 Sõnumite saatmine prototüübile

Teiseks testimise viisiks on sõnumite saatmine prototüübile. Missioonijuhtimistarkvara meeskonnal on enda prototüüp, mis töötab nii nagu ka satelliidile minev elektritoitesüsteemi. Prototüübi abil saab testida loodud tarkvaraliste lahenduste sobivust.

Prototüübiga suhtlemiseks tuleb ühendada arvuti USB pesasse väike raadio antenn. Sarnane antenn asub ka prototüübi küljes. Antennide vahendusel toimub HDLC pakettide saatmine süsteemi ja elektritoitesüsteemi vahel. Eelmises peatükis kirjeldatud testimise korral saadetud paketid sõnumivahetuskomponendist ei lahkunud. Nüüd aga edastatakse sõnumid USB pesas paiknevale antennile. Arvuti küljes olev antenn saadab sõnumi elektritoitesüsteemi prototüübile, mille küljes olev antenn need vastu võtab ja sõnumeid töötlevale süsteemile edastab. Vastupidises järjekorras jõuab vastus tagasi missioonijuhtimistarkvara süsteemile.

6.2.1 Üleslaadimisel tekkinud probleemid

Lahenduse testimise käigus selgus, et testitava tarkvarafaili 1886st reast õnnestub saata ainult kuus rida. Kuna kuue rea saatmine oli siiski edukas, tekkis kahtlus, et viga peitub pigem tarkvarafaili andmetes, kui loodud lahenduses. Süvenedes põhjalikult tarkvarafaili andmetesse, avastati, et seal esinevad ka väärtused kujul 7E, millega muidu identifitseeritakse paketi algust ja lõppu. Kuna teistsuguste käskude saatmine prototüübile töötab, oli üsna keerukas jõuda probleemile jälile. Meeskonnale oli teada, et 7E tüüpi väärtusi võib küll esineda ning sellisel puhul tuleb rakendada muudatust, kus bittide kujul olevale väärtusele lisatakse viiele 1-le lõppu juurde üks 0. Sellist tegevust nimetatakse *bit-stuffinguks*.

Pakettidele *bit-stuffingu* tegemise ülesanne on missioonijuhtimistarkvaras komponendil, mis tegeleb raadiosignaalidega, ning seda ei ole veel realiseeritud. Sellest tulenevalt on hetkel USB pesasse ühendataval antennil üks riistvaraline komponent, mis selle teostab enne antennist välja saatmist. Seda teades, tekkis veel rohkem küsimusi, miks ei õnnestu rohkem ridu saata. Kuid kuna probleem tekkis alles rea puhul, kus paketi andmeosas oli väärtus 7E, uuriti seda edasi. Konsulteerides probleemi üle elektritoitesüsteemi loonud meeskonnaga selgus, et HDLC pakettidele tuleb rakendada ka *byte-stuffing*.

Byte-stuffing'u korral asendatakse paketi andmeosas paiknev 7E sümbol 7D 5E sümbolitega ja 7D sümboli korral 7D 5D sümbolitega. Sümboli 7D korral on tegemist *escape* sümboliga, mis näitab vastuvõtjale, et õige sümbol on asendatud. Sümbolite 5E ja 5D näol on tegemist vastavalt sümbolitega 7E ja 7D, mille viies bit on inverteeritud. [9, p. 3]

*Byte-stuffing'*ut missioonijuhtimistarkvaras ei olnud realiseeritud. Paradoksaalsemaks teeb asjaolu see, et meeskonna arutelu käigus selgus, et tegelikult *byte-stuffing*'ut ei olegi vaja realiseerida kuna seda läheb vaja ainult satelliidi peal olevate komponentide omavahelise suhtluse korral. Satelliidile maapealt saadetavad HDLC paketid, pannakse omakorda kõrgema taseme raadioside pakettide sisse. Satelliidile jõudes, pakitakse need lahti HDLC pakettideks ja edastatakse mööda siini õigele süsteemile.

Loodud mooduli testimiseks prototüübil läks seda siiski vaja. Kuna HDLC pakettide sisu on sõne tüüpi, loodi lihtne for-tsükkel, mis itereerib üle paketi ja asendab 7E ja 7D sümbolid vastavalt 7D 5E või 7D 5D sümbolitega. Järgmisel testimisel õnnestus saata 1886 reast ainult 18 rida.

Kuna probleem tekkis jälle reaga, kus oli sümbol 7E, siis uuriti loodud *byte*-stuffing'u meetodit. Selgus, et selle implementatsioon ei olnud õige. Meetod asendas liigselt 7E ja 7D sümboleid. Vasakult paremale lugedes, tuli vaadata paketis olevaid andmeid kahekaupa ja asendada sümbolid, kus 7E või 7D on paaris. Eelnevalt loodud meetod asendas ka sümbolid, kus 7E ja 7D esinesid näiteks kujul 57 E0. Loodud meetodit täiendati nii, et üle paketi itereeritakse paari kaupa ja *flag* või *escape* sümboli korral see asendatakse (Joonis 15).

```
private String byteStuffing(String data) {
        StringBuilder stuffed = new StringBuilder();
        for (int i = 0; i < data.length(); i++) {
            if (i % TWO == 0) {
                String hexByte = data.substring(i, i + TWO);
                if (hexByte.equals(FLAG)) {
                    stuffed.append(ESCAPED_FLAG);
                } else if (hexByte.equals(ESCAPE)) {
                    stuffed.append(ESCAPED_ESCAPE);
                } else {
                    stuffed.append(hexByte);
                }
            }
        }
        return stuffed.toString();
}
```

Joonis 15. Täiustatud byte-stuffing meetod

Uuesti testimisel töötas saatmine edukalt. Kõik tarkvarafaili read jõudsid kohale ning uus tarkvara sai prototüübile paigaldatud.

7 Edasine töö

Järgnevalt tuleb otsustada, kuidas toimub tarkvara failide testimine ja nende üles panemine GitLabi repositooriumisse. Samuti tuleb hoolt kanda ka selle eest, et toimuks liidestumine teiste veel mitte valmis olevate missioonijuhtimistarkvara osadega, mis tegelevad kommunikatsiooniga maa ja satelliidi vahel. Kuna kogu missioonijuhtimistarkvara on arendus faasis, peab arvestama ka sellega, et süsteemis võib tekkida muudatusi ja võib tekkida olukord, kus bakalaureuse töö jaoks loodud lahendust tuleb muuta.

Missioonijuhtimistarkvara meeskonnal on veel palju tööd teha, ning kui mooduliga rohkem tegeleda pole vaja, saab panustada muude komponentide loomisse.

8 Kokkuvõte

Bakalaureuse töö eesmärgiks oli luua moodul TTÜ100 tudengisatelliidi missioonijuhtimistarkvarasse, mida kasutades saab satelliidi elektritoitesüsteemile laadida uut tarkvara. Mooduli loomisel tuli arvestada missioonijuhtimistarkvaras kasutusel olevate tehnoloogiatega.

Töö raames kirjeldati missioonijuhtimistarkvara arhitektuuri, mis koosneb viiest suurest komponendist: andmebaasist, sõnumivahetuskomponendist, haldusrakendusest, põhimoodulist ja avalikust veebiportaalist. Räägiti elektritoitesüsteemist ja selgitati millised peavad olema HDLC paketid, mille vahendusel käib suhtlus elektritoitesüsteemiga. Järgmisena selgitati nõudeid, mida tuli mooduli loomisel arvesse võtta. Nendest olulisemad olid elektritoitesüsteemi eripärast tulenevad piirangud, näiteks üleslaetava faili rea haaval saatmine. Järgmisena kirjeldati mooduli realisatsiooni. Selgitati, kuidas pääsetakse ligi tarkvara failidele ja kuidas toimib kasutajaliides. Kirjeldati REST teenuse päringuid ja mooduli *back-end* poolt. Eelviimases peatükis räägiti kahest viisist, kuidas testiti loodud lahendust. Esimeseks viisiks oli vastuste imiteerimine. Teise viisina testiti lahendust prototüübil ja testimise käigus esinesid probleemid, mis tuli lahendada. Viimases peatükis räägiti mooduliga seotud edasisest tööst.

Lõpp tulemusena valmis moodul missioonijuhtimistarkvaras, mida kasutades on võimalik elektritoitesüsteemile edukalt uus tarkvara üles laadida. Lahenduse testimiseks kasutati prototüüpi.

Kasutatud kirjandus

[1] S. Romanov, „Kuupsateliidi missioonijuhtimistarkvara arhitektuur,“ 2017.

[2] „ANDMEKAITSE JA INFOTURBE LEKSIKON,“ [Võrgumaterjal]. Available: https://akit.cyber.ee/term/13166-hdlc. [Kasutatud 28 04 2018].

[3] „TTU100 Satellite,“ [Võrgumaterjal]. Available: https://ttu.ee/projektid/mektory-est/satelliidiprogramm-4/satelliidiprogramm/ttu-mektory-satelliidiprogramm/. [Kasutatud 19 05 2018].

[4] V. Pooler, „Elektritoitesüsteemi disain kuupsatelliidile,“ Tallinn, 2016.

[5] V. Sinivee, *EPS communication protocol and register map,* Tallinn, 2017.

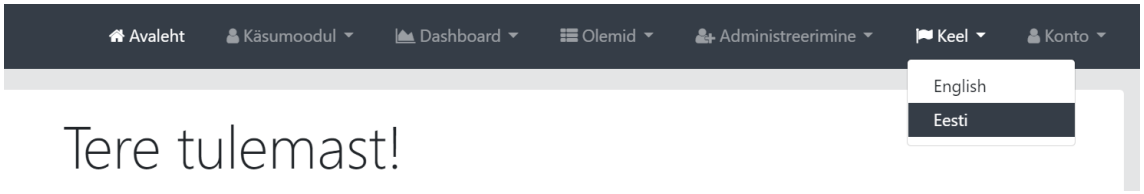
[6] R. Adalbert, *Tut-Mektory Nanosatellite Satellite TCTM Protocol Description,* Tallinn, 2016.

[7] „ANDMEKAITSE JA INFOTURBE LEKSIKON,“ [Võrgumaterjal]. Available: https://akit.cyber.ee/term/12860-modbus. [Kasutatud 01 05 2018].

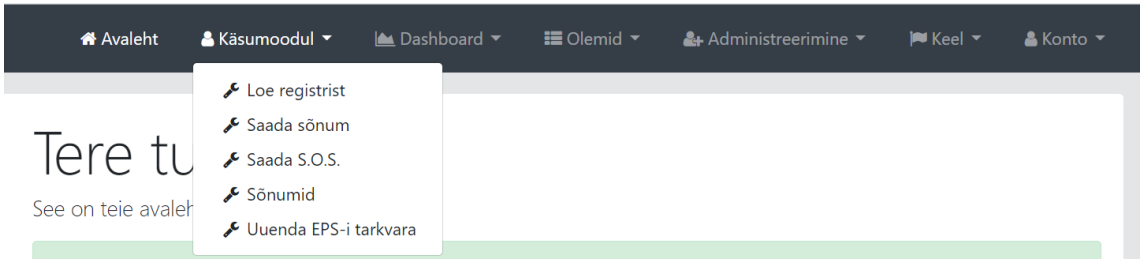
[8] „ANDMEKAITSE JA INFOTURBE LEKSIKON,“ [Võrgumaterjal]. Available: https://akit.cyber.ee/term/8826-base64. [Kasutatud 28 04 2018].

[9] E. Priidel, „HDLC_AX25_KISS,“ Tallinn, 2016.

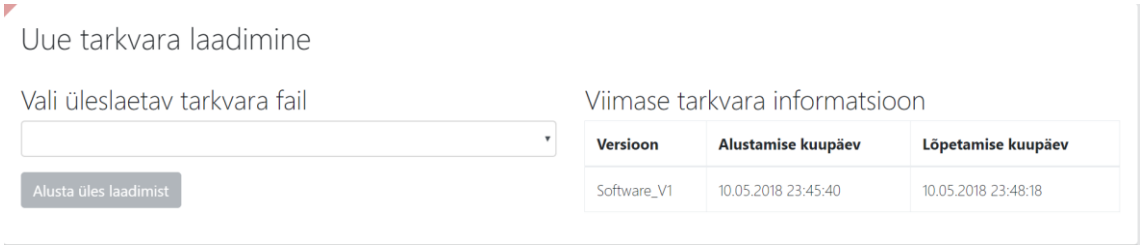
Lisa 1 – Kasutajaliidesest keele muutmine



Lisa 2 – Mooduli avamine navigatsiooni ribal olevast rippmenüüst



Lisa 3 – Mooduli kasutajaliides



Lisa 4 – Üleslaadimise progressi kuvamine