Tartu ülikool
Arvutiteaduse instituut
Informaatika õppekava

Priit Pärn

Automaatkontrollide loomise süsteemi kasutajaliides

Bakalaureusetöö (9 EAP)

Juhendajad: Reimo Palm, PhD
Eerik Muuli, MSc

Tartu 2024

Automaatkontrollide loomise süsteemi kasutajaliides

Lühikokkuvõte:
Käesoleva bakalaureusetöö eesmärgiks on luua graafiline kasutajaliides automaatkontrollide koostamiseks automaatkontrolli süsteemile Lahendus, mis võetakse aluseks tegeliku kasutajaliidese loomisel. Töös kirjeldatakse kasutajaliidesele püstitatud nõudeid ja vahendeid, mida kasutati kasutajaliidese arendamiseks. Töö tulemusena valmis kasutajaliidese prototüüp, mida saab kasutada tegeliku kasutajaliidese loomiseks.
Võtmesõnad:
Kasutajaliides, kasutajakogemus, tarkvara testimine, programmeerimise õpetamine
CERCS:
P175 Informaatika, süsteemiteooria
S270 Pedagoogika ja didaktika

The User Interface of the System for Creating Automatic Checks

Abstract:
The goal of this bachelor's thesis is to create a graphical user interface for creating automatic checks for the automatic control system Lahendus, which will be used as a basis for creating the actual user interface. The thesis describes the requirements set for the user interface and the tools that were used to develop the user interface. As a result of this thesis, a prototype of the user interface was completed, which can be used to create a real user interface.
Keywords:
User interface, user experience, software testing, teaching programming
CERCS:
P175 Informatics, systems theory
S270 Pedagogy and didactics

Sisukord

1 Sissejuhatus

Tartu Ülikoolis õpetatakse programmeerimist erinevatel programmeerimise kursustel, kus on kasutusel automaattestid. Mõned sellised kursused on “Programmeerimine” (LTAT.03.001), “Programmeerimise alused” (MTAT.03.236) ja “Programmeerimise alused II” (MTAT.03.256), kus ülesannete lahendusteks on suuresti tudengite poolt koostatud programmid [1-3]. Lahenduste kontrollimiseks kasutatakse tihti Moodle'is automaatteste, mis on varasemalt valmis programmeeritud. Kursuste läbiviijatel on soov ülesandeid kursustel muuta või juurde lisada, et need varieeruksid võrreldes varasemate aastatega, vältimaks olukorda, kus tudengid saaksid kasutada varasemate aastate lahendusi. Seoses selle sooviga tulevad mängu probleemid, mis seostuvad automaattestide loomisega või muutmisega Moodle'i keskkonnas. Moodle'is kasutatakse automaatkontrollide koostamiseks ja jooksutamiseks täiesti eraldi teeki, kus peab iga ülesande puhul automaatkontrolle looma Pythonis neid programmeerides, mis on vägagi aeganõudev ja korduv protsess. Lisaks nõuab see ka programmeerimise teadmisi, mille tõttu igaüks ei saa automaatkontrolle koostada.

Mainitud probleemidele potentsiaalseks lahenduseks oleks kasutajaliides, kus automaatkontrollide loomine käiks läbi kasutajaliidese poolt võimaldatud funktsionaalsuste, milles on ära kirjeldatud, mida esitatud lahendus peab tegema või sisaldama. Nii muutuks kontrollide lisamise ja muutmise protsess kiiremaks ning kasutajasõbralikumaks. Lisaks ei ole sellisel juhul ka vaja programmeerimise teadmisi, sest läbi kasutajaliidese automaatkontrollide loomisel ei pea koodi kirjutama.

Antud bakalaureusetöö eesmärk on luua olemasolevale automaatkontrollisüsteemile kasutajaliidese prototüüp, mille abil saaks kasutaja lihtsasti koostada uusi teste ja olemasolevaid vajadusel muuta. Seda kõike saaks teha nii, et kontrollide loomisel või muutmisel ei pea kasutaja ise mitte ühtegi rida koodi kirjutama.

Bakalaureusetöö esimeses osas kirjeldatakse lähemalt juhiseid, mida silmas pidada kasutajaliidese ja kasutuskogemuse loomisel. Lisaks peatutakse põgusalt tarkvara testimise tähtsusel tänapäeva maailmas. Teises osas tuuakse välja juba olemasolevad lahendused ning nende plussid ja miinused. Kolmandas osas tutvustatakse antud bakalaureusetöö kasutajaliidese prototüübile püstitatud nõudeid ja esialgset prototüüpi. Neljandas osas saab näha valminud kasutajaliidest ja selle arendamise protsessi. Viiendas osas kirjeldatakse

tulemusi ja järeldusi, kus tuuakse välja valminud kasutajaliidese eelised võrreldes Moodle'is automaattestide programmeerimisega ja võimalikud edasiarendused.

2 Mõisted ja terminid

Test: Antud töö raames loodud kasutajaliidese kontekstis on test automaatkontrolli peamine element. Test võib sisaldada ühte või mitut kontrolli.

Kontroll: Antud töö raames loodud kasutajaliidese kontekstis on kontroll kõige väiksem terviklik üksus, mis kontrollib ühte kindlat programmi aspekti.

TSL (ingl *Test Specific Language*) [14]: Testide ja kontrollide kirjapanemise keel.

Kasutajaliides (ingl *user interface*) [18]: Kasutaja ja arvutiprogrammi vaheline ühenduslüli. Kasutajaliides kujutab endast käskude või menüüde komplekti, mille abil kasutaja saab programmiga suhelda.

Aedik/Liivakast (ingl *sandbox*) [18]: Tarkvaraarenduses tootmiskeskkonnaga sarnast funktsionaalsust omav testimiskeskkond, milles katsetatakse uusi koodimuudatusi ilma tootmiskeskkonda ohustamata.

Kasutajakogemus (ingl *user experience*): Kasutajakogemus viitab üldisele kogemusele ja rahulolule, mis inimesel on toote, teenuse või süsteemiga suhtlemisel, eriti seoses sellega, kui lihtne, tõhus, nauditav ja tähendusrikas see suhtlus on.

3 Taust

Järgnevates alapeatükkides tutvustatakse lähemalt kasutajaliidese ja kasutajakogemuse juhised. Lisaks seletatakse lahti tarkvara testimise eesmärke.

3.1 Kasutajaliidesed

Kasutajaliidese loomine, mis näeks lihtne välja, oleks kasutajasõbralik, intuitiivne kasutada ja efektiivne, on probleem, millele on proovitud leida ühtset lahendust sellest ajast saati, kui ilmusid monitorid ja ekraanid. Selle probleemi puhul ei ole olemas kindlat algoritmi, mida jälgida kasutajaliidese loomisel, vaid on olemas juhised, mida tuleks silmas pidada. E. Wong [4] on enda artiklis välja toonud 10 sellist juhist. Järgnevalt on mõned juhised eraldi välja toodud:

1) Esteetiline ja minimalistlik disain, kus kasutajaliideses ei ole näha üleliigset informatsiooni, mis võiks kasutajat segada.
2) Järjepidevus ja standardid ehk kasutajaliidese disain peab olema ühtne. Näiteks nupp, mis on kahes kohas kasutusel, peab tegema mõlemas kohas ühte ja sama asja, mitte ühes kohas ühte ja teises kohas teist.

On olemas mitmeid viise, kuidas luua kasutajaliideseid, millest üks enim kasutatud on lihtsalt visuaalne disainimine, aga selle juures on omad nüansid. E. Wong [5] toob välja Ben Shneidermanni kaheksa kuldset reeglit, mille abil saab luua parema kasutajaliidese:

1) Pürgida järjepidevuse poole ehk kasutada näiteks ikoone ja värviskeeme, mis omavahel sobivad ning on kasutajale tuttavad. Näiteks olukord, kus nupp, mis on tähistatud ikooniga “+”, tähendab millegi lisamist.
2) Lubada kasutajatel kasutada klaviatuuri otseteid tegevuste sooritamiseks.
3) Pakkuda informatiivset tagasisidet kasutaja tegevustele. Tuleks vältida olukorda, kus kasutajale näidatakse sõnumit, mis sisaldab veateadet inimesele mitteloetaval kujul.
4) Kui mingisugune tegevus on sooritatud, siis tuleks sellest kasutajale ka teada anda. Näiteks veebipoes ostu sooritamisel tänada kasutajat ja pakkuda võimalust, kas soovitakse meilile ostutšekki.
5) Pakkuda lihtsat veakäsitlust. Näiteks kui internetis vormi täitmisel unustab kasutaja täita ära vajaliku välja, siis vormi esitamisel tuleks kasutajale teada anda sellest, et üks väli on veel täitmata.

6) Lubada toimingute lihtsat tagasivõtmist, kus kasutaja teab, et ta saab enda tehtud tegevusi tagasi võtta, kui sooritatud toimingul ei olnud soovitud tulemus.
7) Kasutaja käes peaks olema kontroll, kus toiminguid alustab kasutaja ise, mitte süsteem ei alusta neid tema eest.
8) Vähendada lühiajalise mälu koormust. Ei tohiks tekkida olukorda, kus kasutaja peab jätma paljusid asju meelde, sest see on kurnav.

Näiteks H. Jan jt [6] uurisid kasutajaliidest, mis oli mõeldud ravimite jaoks. Seal tuuakse välja head soovitused, mida kasutada sellistes olukordades, mis on seotud just teksti loetavusega. Näiteks ravimid nimekirjas peaksid olema üksteisest selgelt eristatavad, tuleks vältida teatud lühendeid, et inimesed neid segamini ei ajaks (näiteks IV ja IU), ja soovitatakse tehnikat, kus osa ravimi nimest kirjutada suurte tähtedega, et seda selgelt eristada teistest. Kasutajaliideste loomine ei piirdu ainult eelpool nimetatud asjadele tähelepanu juhtimisega, kuid nende rakendamine peaks looma tugeva aluse.

3.2 Kasutajakogemus

Kasutajakogemusele ühtset definitsiooni leida on keeruline, sest kõik eksperdid annavad sellele oma definitsiooni ja ei ole paika pandud kindlat definitsiooni. A. H. Allam jt [7] toob enda töös välja seitse erinevat definitsiooni, mis on teiste poolt öeldud. Üldiselt on kasutajakogemus kasutaja rahulolu mingisuguse kindla tegevusega, mis ei piirdu ainult tarkvara kasutamisega.

P. Morville on teinud diagrammi, mis on näha järgneval joonisel, kus on ära kirjeldatud kasutajakogemuse peamised aspektid:

Joonis 1. Peter Morville'i diagramm kasutajakogemuse aspektidest tõlgitud eesti keelde [8].

1) Kasulikkus: toode peab olema kasulik kasutajale, sest vastasel juhul pole tootest mingisugust kasu. Toode peab aitama kaasa kasutaja eesmärgi teostamisele.
2) Ihaldatavust antakse toote puhul disainis edasi kuvandi, identiteedi, esteetika ja emotsionaalse disaini kaudu.
3) Ligipääsetavus: toode peab olema lihtsasti kättesaadav kõikidele, kaasa arvatud puuetega inimestele.
4) Usutavus: toode peab olema usaldusväärne kasutajale. Näiteks kui veebileht tagastab kasutaja sisendi puhul ilmselgelt vale vastuse, siis ei ole tegemist usutava veebilehega ja kasutajale jääb halb kasutajakogemus selle kohta.
5) Leitavus: toode peab olema lihtsalt leitav, olgu see veebileht internetis või koostisainete loetelu leidmine toidupakendil.
6) Kasutatavus: toode peab olema lihtsasti kasutatav.

7) Väärtuslikkus: toode peab andma väärtust, olgu see näiteks rahaline või ajaline väärtus.

Varasemalt loetletud kasutajakogemuse aspekte peetakse põhilisteks, aga vajadusel saab veel rohkem detailidesse laskuda, kus iga aspekt jaotub veel omakorda väiksemateks osadeks.

3.3 Tarkvara testimine

Loodava kasutajaliidese suurem eesmärk on paremini võimaldada tarkvara testimist. Tarkvara testimine on protsess, mis kindlustab, et tarkvara teeb seda, milleks see on loodud, ja ei tee seda, milleks see ei ole loodud [9].

Testimise põhiline eesmärk on vigade leidmine ja nende parandamine. Täpsemalt on eesmärke enamasti neli [10]:

1) Demonstratsioon, kus testidega saab näidata, et tarkvara töötab ja seda saab hakata kasutama.
2) Vigade märkamine, kus testimise tulemusel tulevad välja tarkvara vead ja puudused, mida programmeerides ei osatud ette näha.
3) Ennetamine, kus testimise tulemusel saadud informatsiooniga saab vältida või vähendada vigade arvu ka tulevikus.
4) Kvaliteedi parandamine, kus testide jooksutamisel ja vigade parandamisel muutub ka koodi kvaliteet paremaks.

Hea näide tarkvara testimise tähtsusest on Marsi *Climate Orbiteri* intsident, mis juhtus aastal 1999, kus peale 10-kuust reisi Marsi poole pidi sond jääma Marsi orbiidile, aga hoopis arvatavasti põles ära Marsi atmosfääris. Nimelt tarkvara, mis kontrollis sondi tõukureid, tegi arvutusi imperiaalsete mõõtühikutega, aga teine osa koodist, mis neid andmeid luges, eeldas, et need on meetermõõdustikus, mille tõttu sond kiirendas rohkem kui tarvis ja sisenes Marsi atmosfääri [11].

Tarkvara testimist tehakse enamasti kahte moodi, kas käsitsi või automaatselt. Automaatse testimise puhul saab väga suurel hulgal teste jooksutada, aga on olemas ka testitüüpe, kus programmi peab manuaalselt testima, näiteks *black-box* testimine. Lõputöö raames loodava kasutajaliidese poolt loodud automaattestid on mõeldud automaatseks testimiseks.

4 Olemasolevad lahendused

Järgnevates alapeatükkides tutvustatakse antud ajahetkel Tartu Ülikoolis kasutatavaid lahendusi, ehk kuidas luuakse järgnevates keskkondades automaatteste, millega kontrollitakse tudengite lahendusi. Esimene neist on Moodle'i keskkonnas kasutatav teek VPL. Teine on keskkond Lahendus [12], kus on võimalik automaatteste luua nii neid programmeerides kui ka kasutajaliidese abil.

4.1 Moodle’i keskkonna automaattestid

Lõputöö kirjutamise ajal baseeruvad Tartu Ülikooli Moodle'i keskkonna automaattestid VPL-il. VPL on avatud lähtekoodiga virtuaalne programmeerimise labor, mille saab laiendusena Moodle'isse lisada. VPL võimaldab kirjutada automaatteste, neid käivitada ja redigeerida otse Moodle'is ehk selleks ei ole vaja eraldi arenduskeskkonda alla laadida. Kuna VPL on integreeritud Moodle'iga, siis kirjutatakse VPL automaattestide tulemused otse hinnete tabelisse ehk juhendaja ei pea neid käsitsi sisestama [13].

Suureks puuduseks VPL-i kasutamise puhul on kasutajaliidese puudumine. Automaatteste saab kirjutada Moodle’i keskkonnas tekstiredaktoris, aga pigem eelistatakse selle asemel teisi Pythoni arendusvahendeid, kust hiljem saab need Moodle’isse kopeerida. Siiski toimub nende programmeerimine Pythoni keeles, mis nõuab teatud programmeerimisoskust. Iga automaattest on üks Pythoni programm, mis tuleb iga kontrollitava ülesande jaoks eraldi luua. Seetõttu on nende haldamine ja ümbertegemine keerukas ning töömahukas protsess, eriti mahukamate automaattestide puhul.

Järgnevalt on välja toodud üks väiksema mahuga automaattest Tartu Ülikooli kursuselt “Programmeerimine”, milleks on aastaaegade ülesande kontrollimine. Analüüsime ka peatselt, kuidas see on VPL-i abiga implementeeritud. Joonisel 2 on näha erinevad menüüd ja nende otstarbed.

Joonis 2. Automaattesti vahelehed.

Seoses automaattestimisega huvitavad meid ainult vahelehed “Ülesande kirjeldus” ja “Katsetamine”.

Menüüs “Ülesande kirjeldus” on kirjeldatud automaattesti sätted ja see sisaldab ka kõiki faile, mis on käivitamiseks vajalikud. Joonisel 3 on näha põhiline testimise funktsioon “*check*”, mis võtab sisendiks kuule vastava numbri. Sisendiga tehakse operatsioone ja tagastatakse vastavalt operatsioonide tulemusele teade kasutajale.

```
16  def check(kuu):
17
18      programminimi = "aastaajad.py"
19
20      progolemasolu(programminimi)
21
22      with Plan("Käivitan programmi sisendiga " + str(kuu) + "."):
23
24          # Programmi käivitamine
25
26          sisendid = [kuu] * 20 + [""] * 20
27          oodatav_sisendite_arv = 1
28          veatekst = "Programmil peaks olema ainult üks sisend, kuu number."
29
30          globs, cap = täidaprog(programminimi, sisendid, oodatav_sisendite_arv, veatekst)
31
32          väljund = cap.get_last_stdout()
33
34          oodatav_vastus = õige_vastus(kuu)
35
36          # Väljundis pole midagi
37
38          if väljund.strip() == "":
39              teade = "Programm ei väljastanud midagi, aga pidi väljastama aastaaja nime."
40              lõpeta(teade)
41
42          # Väljundi ridade arv
43
44          väljundiread = []
45          for x in väljund.split('\n'):
46              rida = x.strip()
47              if rida != "":
48                  väljundiread.append(rida)
49
50          if len(väljundiread) > 1:
51              teade = "Programmi väljundis on " + str(len(väljundiread)) + " rida.\n"
52              teade += "Ei suuda otsustada, milline neist on õige.\n"
53              teade += "\nProgrammi väljund oli:" + quote_text_block(väljund)
54              lõpeta(teade)
55
56          # Leidub õige vastus
57
58          if oodatav_vastus not in väljund:
59              teade = "Ei leidnud programmi väljundist õiget vastust '" + str(oodatav_vastus) + "'.\n"
60              teade += "\nProgrammi väljund oli:" + quote_text_block(väljund)
61              lõpeta(teade)
62
63  def test1(): check(1)
64  def test2(): check(2)
65  def test3(): check(3)
66  def test4(): check(4)
67  def test5(): check(5)
```

Joonis 3. Moodle'i keskkonnas VPL laienduse abiga loodud automaattest aastaaegade testimiseks.

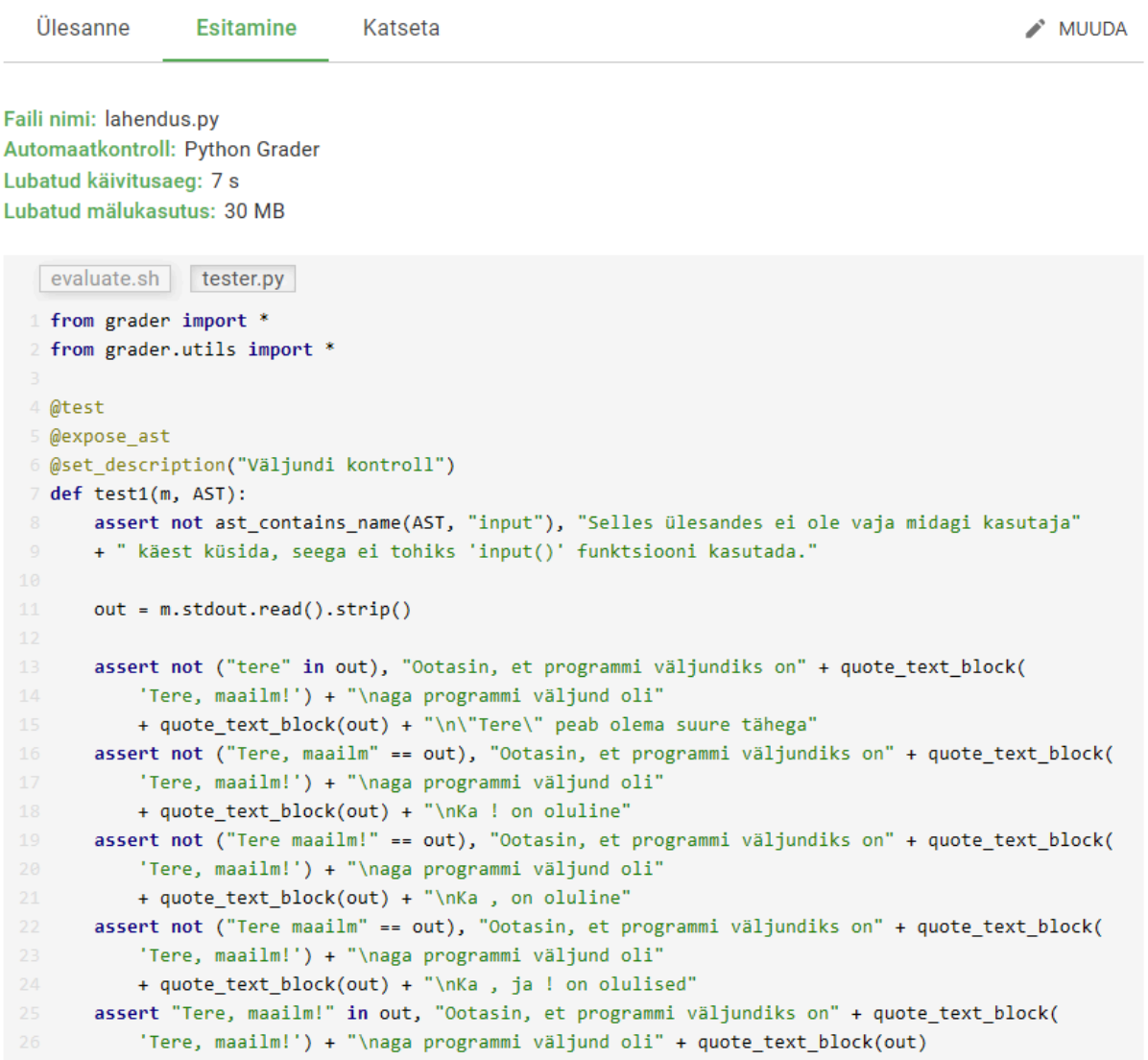
Vahelehel “Esitamine” saab tudeng esitada enda lahenduse ja käivitada automaattestid esitatud lahenduse peal. Joonisel 4 on näha esitatud lahendus ja testimise tulemused.

Joonis 4. Esitatud lahendus ja testimise tulemused keskkonnas Moodle.

Lisaks saab tudeng vahelehel “Redigeerimine” enda lahendust muuta, kui koodi on jäänud mingisugune viga. See on mugav olukordades, kus koodis on pisikene viga sees, sest siis piisab pisivea parandusest kasutajaliideses ning ei pea tervet lahendust uuesti üles laadima.

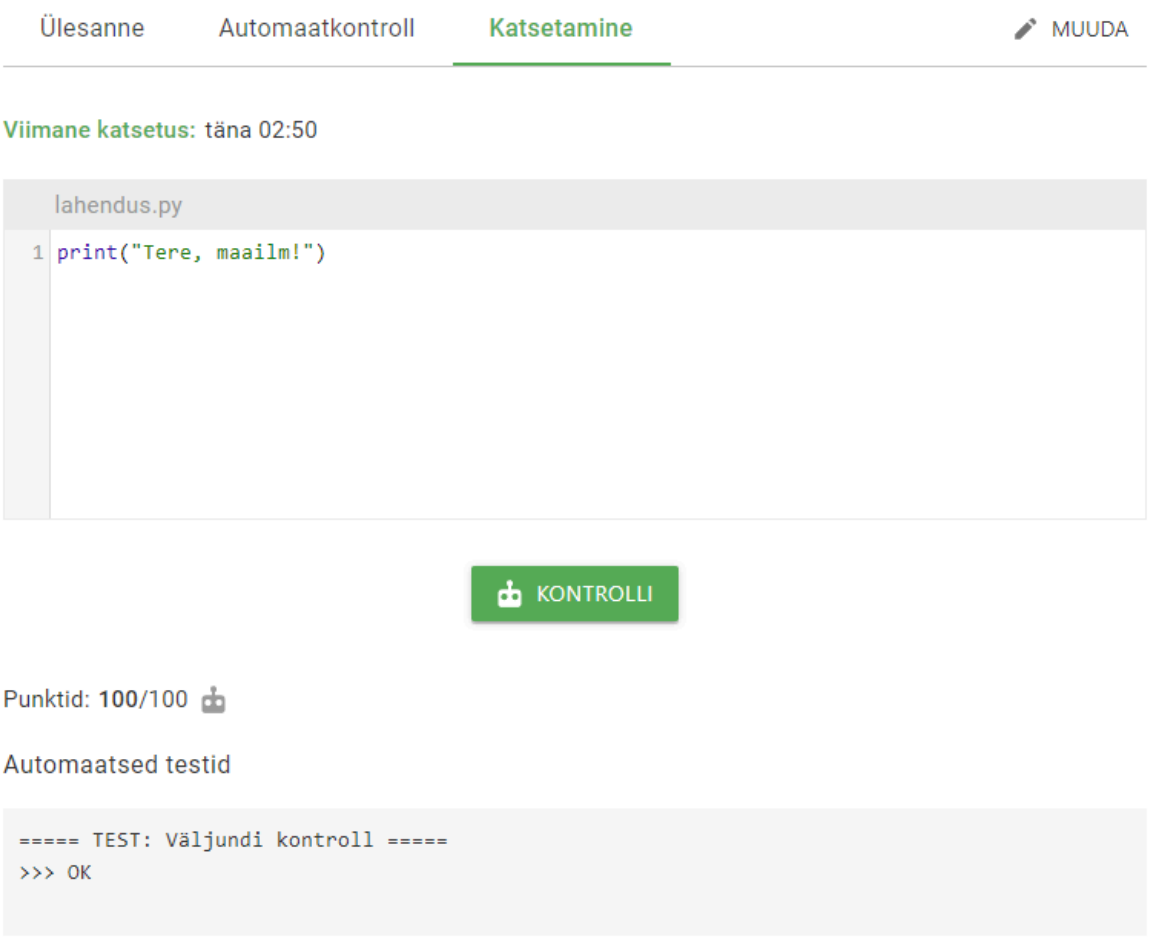
4.2 Lahendus.ut.ee

Lahendus.ut.ee on keskkond, mis on loodud Tartu Ülikooli arvutiteaduse instituudi poolt [12]. Lahendus on alternatiiv Moodle'i keskkonnale ja seal saab sarnaselt luua ja muuta automaatteste, mille abil tudengid saavad testida oma programme. Hetkel käib suuresti testide loomine sarnaselt Moodle'i keskkonnaga, kus tuleb Pythoni keeles valmis kirjutada testid. Joonisel 5 on kujutatud funktsioon “test1”, mis kontrollib, kas programm väljastab ekraanile teksti “Tere, maailm!”.



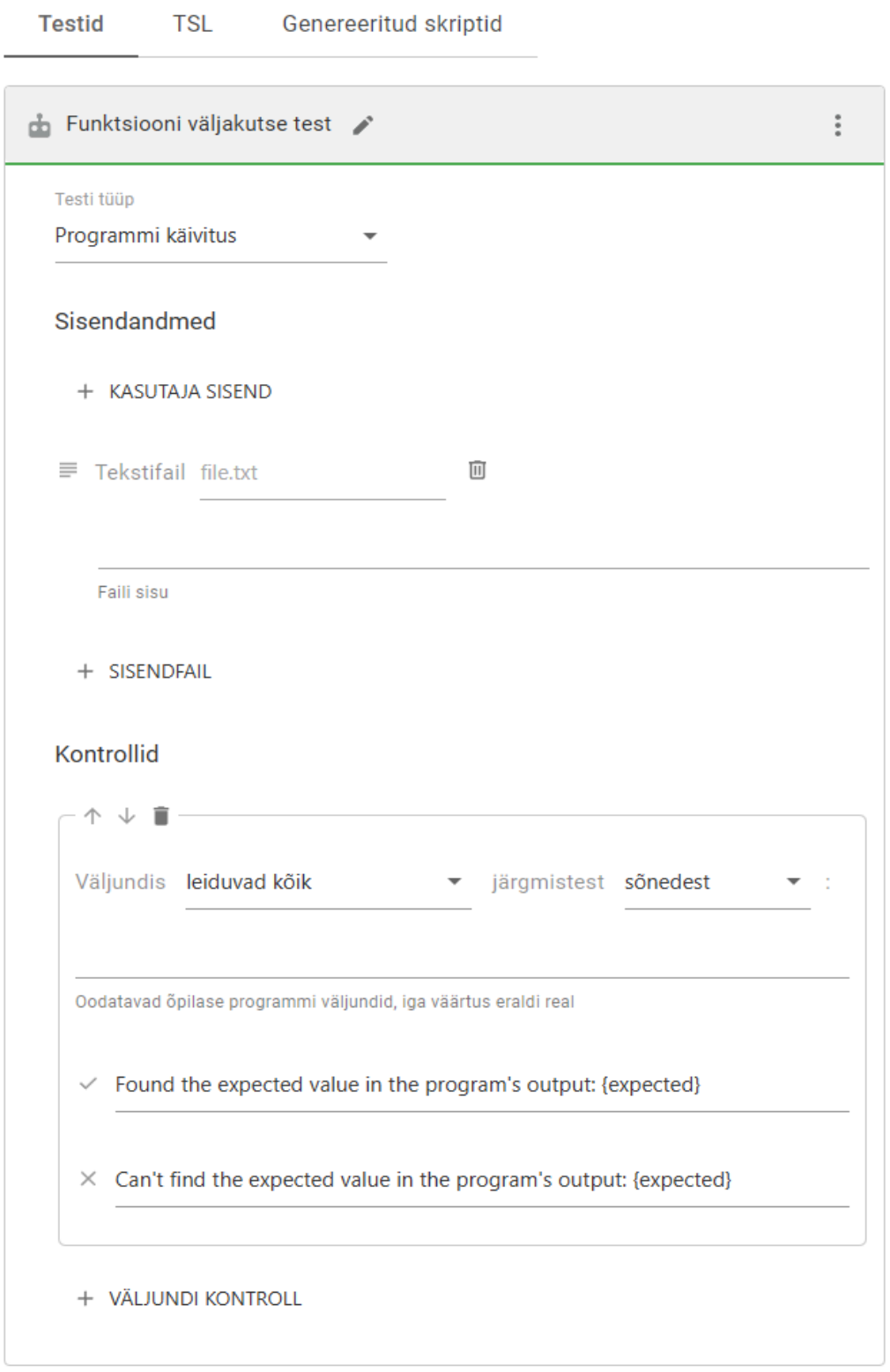
Joonis 5. Automaattestid Lahendus.ut.ee keskkonnas, mis testib, et programm väljastaks ekraanile teksti “Tere, maailm!”.

Oma lahenduse sisestamine toimub vahelehel “Katsetamine”, kus tuleb sisestada kogu programmi kood ning seejärel vajutada nupule “KONTROLLI”. Pärast nupule vajutamist käivitatakse automaattestid ning kasutajale kuvatakse vastav info, nagu on näidatud joonisel 6.



Joonis 6. Kasutaja sisestatud programmi kontrollimine ja testide tulemused.

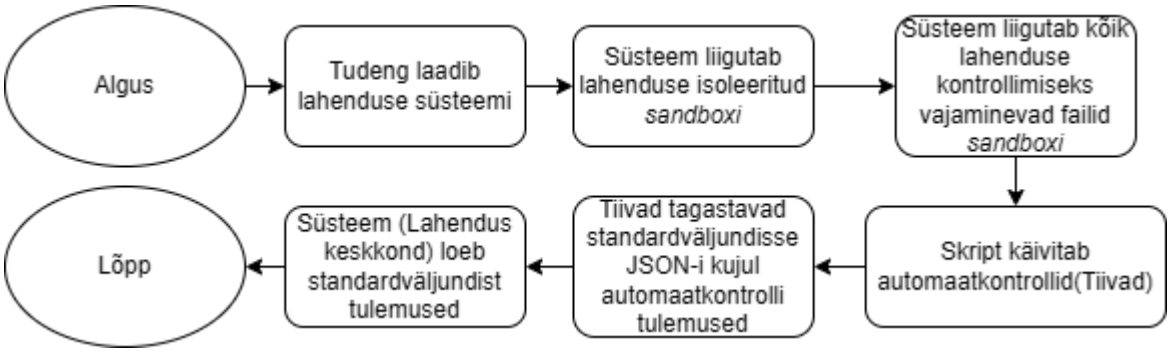
Lahenduse keskkonnas on olemas poolik kasutajaliides, mis sai loodud ajal, kui antud töö autor viibis Kaitseväes, sest kasutajaliidese järele soov Lahenduse lehel oli suur. Joonisel 7 on näha Lahenduse keskkonnas asuv kasutajaliides.



Joonis 7. Lahendus.ut.ee lehel asuv kasutajaliides automaattestide loomiseks.

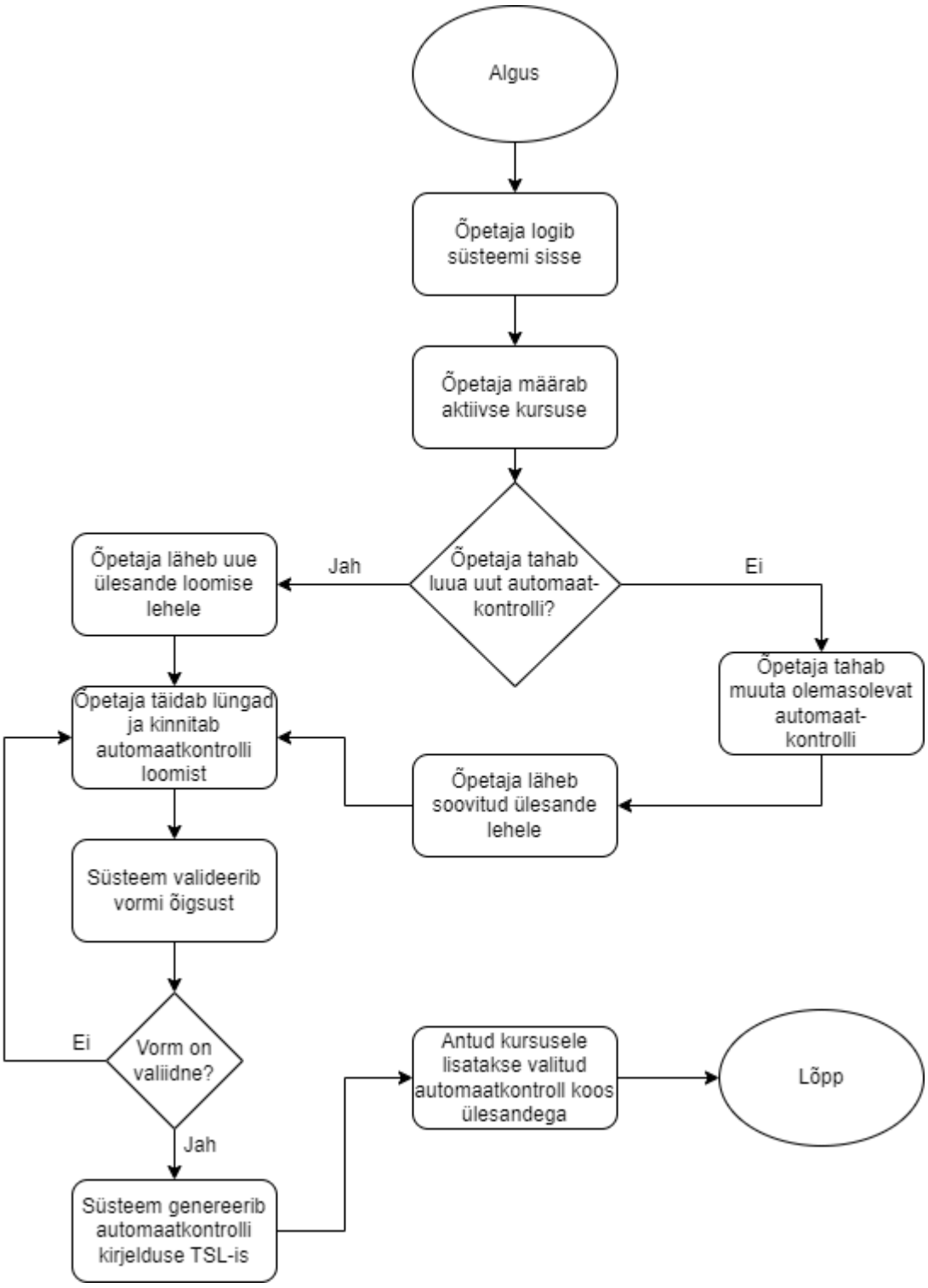
Lahenduse kasutajaliideses saab valida programmi käivituse ja funktsiooni väljakutse testide vahel. Kuid antud hetkel on sealt puudu mõningad kontrollid, näiteks funktsioon kasutab ainult lokaalseid muutujaid, programm viskab erindi, väljundfaili kirjutamise kontroll ja funktsiooni puhul argumendi väärtuse muutmise kontroll ning mitmeid teisi.

Järgmisena tutvustatakse protsessi, mis läbitakse, kui tudeng laadib Lahenduse keskkonda oma programmi kontrollimiseks. Joonisel 8 on kujutatud terve protsess. Esiteks laadib või kopeerib tudeng oma lahenduse süsteemi. Seejärel liigutab süsteem lahenduse isoleeritud liivakasti, et vältida olukorda, kus tudengi programm teeb midagi pahatahtlikku. Sellisel juhul on pahatahtlik programm suletud liivakasti ja sealt välja ei saa. Pärast seda lisab süsteem kõik vajaminevad failid liivakasti, et saaks edukalt tudengi programmi testida. Järgmisena käivitab skript automaatkontrollid tudengi lahenduse peal, kus kasutatakse Tiibade teegi erinevaid funktsioone. Tiivad on eraldi teek, mis sisaldab endas erinevaid funktsioone, mis kujutavad automaatkontrolle. Kui kontrollid on läbi tehtud, tagastatakse standardväljundisse JSON-i kujul automaatkontrollide tulemused. Viimase sammuna loeb Lahenduse keskkond standardväljundist tulemused ja kuvab need kasutajale.



Joonis 8. Süsteemi vaade ehk mis toimub taustal, kui tudeng esitab oma lahenduse.

Teisena tutvustakse protsessi, kus õpetaja tahab luua uut või muuta olemasolevat automaatkontrolli. See on kujutatud joonisel 9. Sammu “Süsteem valideerib vormi õigsust” all on mõeldud seda, et kasutajaliideses oleksid kõik kohustuslikud väljad täidetud. Näitena võib välja tuua sellise olukorra, kus kasutajaliideses märgitakse, et programm võtab sisendiks faili, aga faili nime ei ole antud. Sellises olukorras ei ole vorm valiidne ja sellest antakse kasutajale teada. Pärast vormi valiidsuse kontrollimist genereerib süsteem automaatkontrolli kirjelduse TSL-is.



Joonis 9. Õpetaja vaade, kus kuvatakse sammud automaatkontrolli loomiseks.

Joonisel 9 nimetatud TSL-i tutvustatakse põhjalikumalt järgmises peatükis, aga lühidalt öeldes on tegemist failiga, kust saadakse kätte vajalik info automaattestide loomiseks.

4.3 TSL

TSL on testide ja kontrollide kirjapanemise keel, mis põhineb YAML-il/JSON-il ja mille abil saab defineerida automaatkontrolle [14]. Joonisel 10 on toodud üks näide TSL-failist, kus teostatakse programmifaili kontroll. Alguses on nimetatud programmeerimiskeel ja TSL-i versioon, seejärel tuleb põhiline kontrollitüüp, milleks on programmi faili kontroll, mis omakorda sisaldab väiksemaid teste programmi faili kohta. Esimene neist on programmi olemasolu, kus on ära kirjeldatud sõnumid, mis edastatakse kasutajale, punktid ja erinevad parameetrid. Peale seda on näha veel kahte tüüpi teste, programmi fail ei ole tühi ja programmi fail on Pythoni fail, kus kontrollitakse, kas programmi sisu on Pythoni kood.

```
language: python3
tslVersion: 1.0.0

fileCheck:
  fileName: solution.py
  tests:
    - type: file_exists
      id: 5
      beforeMessage: Enne testi sõnum
      failedMessage: Faili nimega {file_name} ei eksisteeri
      passedMessage: Well done!
      points: 5.1
      name: Faili olemasolu kontroll
      inputs: "[Sisend 1, Sisend 2] - NB - ei ole tegelik sisend!"
      visibleToUser: False
    - type: file_not_empty
    - type: file_is_python
```

Joonis 10. Näide TSL-st, kus on kirjeldatud programmi faili kontroll.

TSL-i kasutatakse Lahenduses, sest kasutajaliideses tehtud valikute põhjal luuakse TSL-fail. TSL-failist genereeritakse automaatkontrolli kood, mis käivitatakse liivakastis, kus tagastatakse kasutajale tulemus. TSL on samal ajal ka kasutajaliidese aluseks ehk

kasutajaliidesest saab luua TSL-i ja vastupidi, sest sellises olukorras peab salvestama andmebaasi ainult TSL-i ja ei ole vajadust kasutajaliidese kirjeldust salvestada eraldi.

Enne kui TSL-failist genereeritakse automaatkontrolli kood, rakendub parsija, mis kontrollib TSL-faili valiidsust automaattestide loomisel kasutajaliideses. Kui midagi on paigast ära, siis kuvatakse vastav info kasutajale ja ei lubata automaatkontrolli salvestada. Kui TSL-fail on valiidne, siis saadetakse see kompilaatorisse, kus sellest genereeritakse vastavad automaatkontrollid Pythoni keeles, mida hiljem liivakastis käivitatakse. Pärast Tiibade funktsioonide käivitamist tagastatakse tulemused standardväljundisse ja süsteem loeb standardväljundist tulemused ning kuvab need kasutajale.

5 Kasutajaliidese loomine

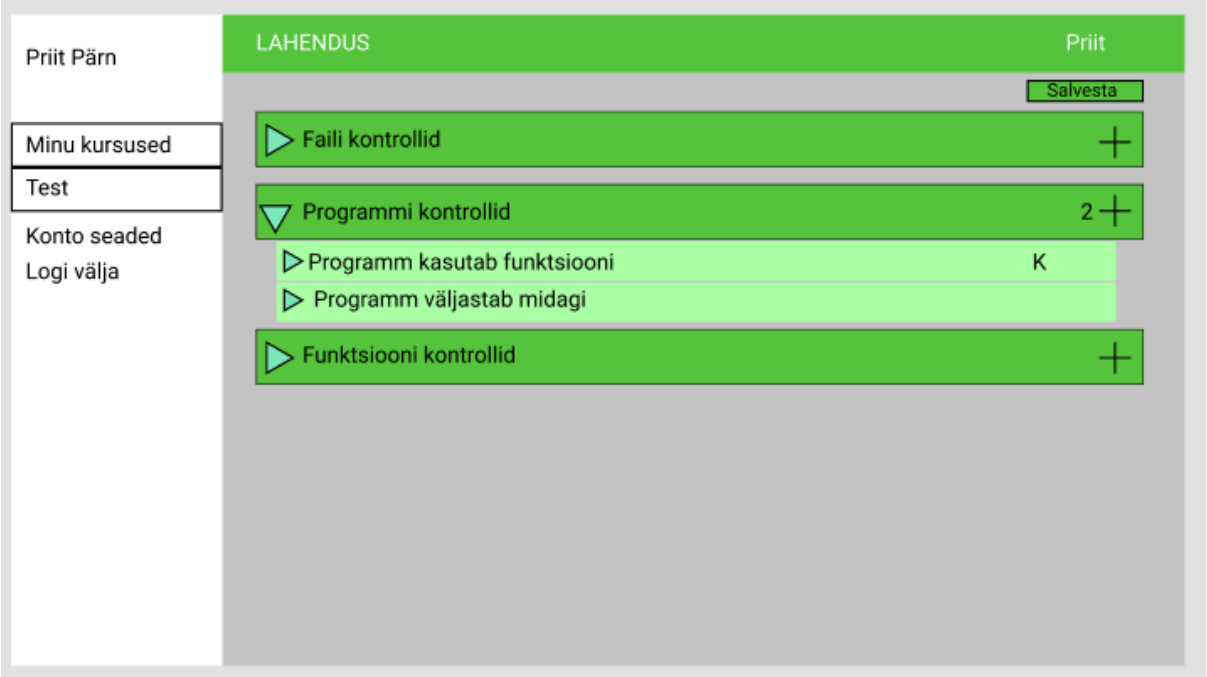
Järgnev peatükk on jagatud väiksemateks osadeks, kus esmalt kirjeldatakse nõudeid kasutajaliidesele ehk milliseid teste see peab sisaldama ja milline võiks olla nende sisu. Seejärel tutvustatakse prototüüpi, mis sai loodud veebilehel Figma [15]. Pärast prototüüpi näidatakse valminud lõplikku kasutajaliidest ja tuuakse näiteid erinevatest testidest, mida kasutajaliidese abil saab luua.

5.1 Nõuded kasutajaliidesele

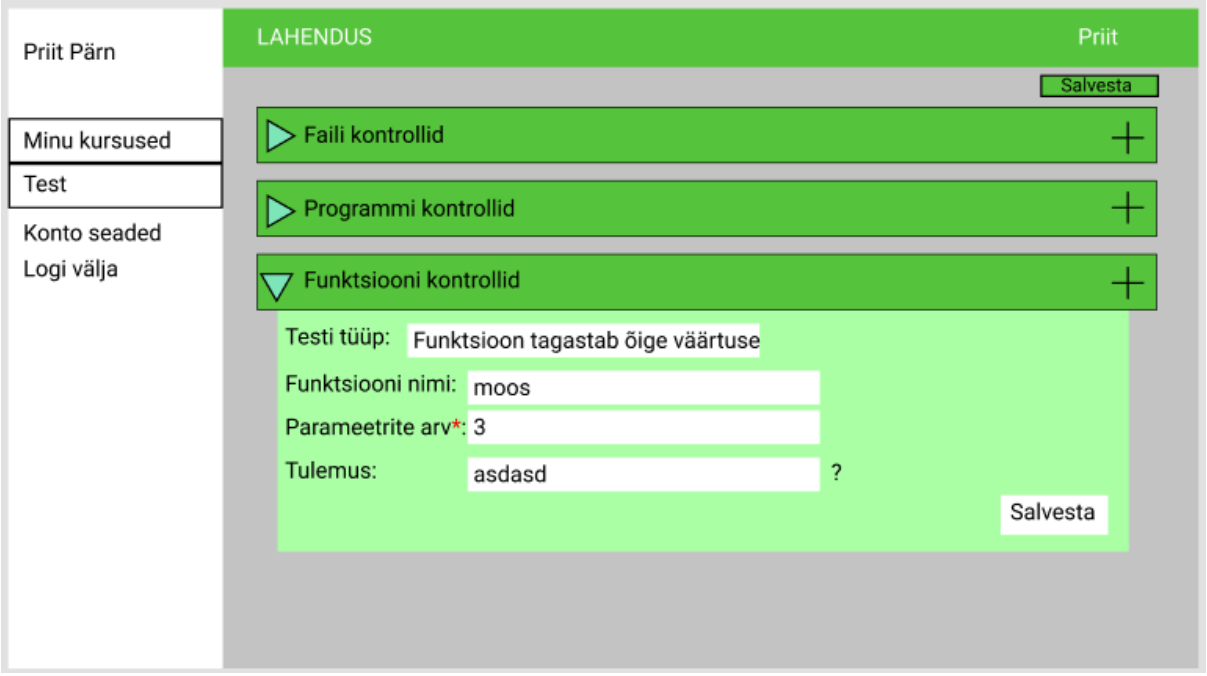
Kasutajaliideses peaks saama luua automaatteste, millega saaks testida aine "Programmeerimine" õppesisu ehk need oleksid tingimuslaused, tsüklid, funktsioonid, sõnetöötlus, erindid ja rekursioon. Lisaks kuna "Programmeerimine" aine toimub Pythoni keeles, siis on kasutajaliideses loodavad testid mõeldud Pythoni programmidele. Näiteks võiks saada testida, kas funktsioon sisaldab tsüklit või programm kutsub välja funktsiooni.

5.2 Figma prototüüp

Esialgne prototüüp kasutajaliidesest sai tehtud Figmas, et paika panna üldine stiil ja testide loomise loogika. Testide loomine ja asetus sai valitud silmas pidades kasutajaliidese häid tavasid ja olemasoleva Lahenduse lehe stiili. Sellepärast oli ka põhiliseks värviks valitud roheline, sest ülejäänud Lahenduse keskkond kasutab ka põhiliselt rohelist ja selle erinevaid toone. Kasutajaliides koosneb kolmest põhilisest kontrollide plokist, nagu on näha joonisel 11. Plokkidesse saab lisada vastavaid automaatteste. Esialgu sai valitud vastavad kolm põhilist kontrollide plokki, sest otsustati, et need oleksid põhilised plokid, mille alla lisatakse erinevaid teisi kontrolle, mis teema poolest kuuluksid vastavale plokile. Nii oleksid kontrollid eraldatud vastavate plokkide järgi. Joonisel 12 on näha, kuidas toimuks funktsiooni testi lisamine Figma kasutajaliidesega.



Joonis 11. Figmas loodud esialgne kasutajaliidese prototüüp, kus on näha põhilised kontrollid.



Joonis 12. Figma kasutajaliidese prototüübis testi lisamine.

Algne Figma prototüüp oli mõeldud selleks, et panna paika visioon, milline umbkaudu võiks reaalne kasutajaliides välja näha. Reaalne kasutajaliides erineb prototüübist mingil määral elementide paigutuse ja üldise väljanägemise poolest, nagu tutvustatakse järgmises peatükis.

6 Valminud kasutajaliides

Järgnevas peatükis tutvustatakse valminud kasutajaliideses kasutatud tehnoloogiaid ja nende valiku põhjuseid. Seejärel on põhjalikum kasutajaliidese tutvustus, kus tuuakse välja kõik moodulid kasutajaliideses. Lõpetuseks tutvustatakse arendusprotsessi ennast lähemalt ja tuuakse välja esinenud probleemid, mis esinesid arendamise ja testimise käigus.

6.1 Kasutatud tehnoloogiad kasutajaliideses

Nimetatud tehnoloogiad said valitud sellepärast, et juba olemasolev lahendus.ut.ee lehekülg kasutab raamistikku Materialize [16] ja sellisel juhul jääks loodava kasutajaliidese stiil samasuguseks juba olemasolevaga.

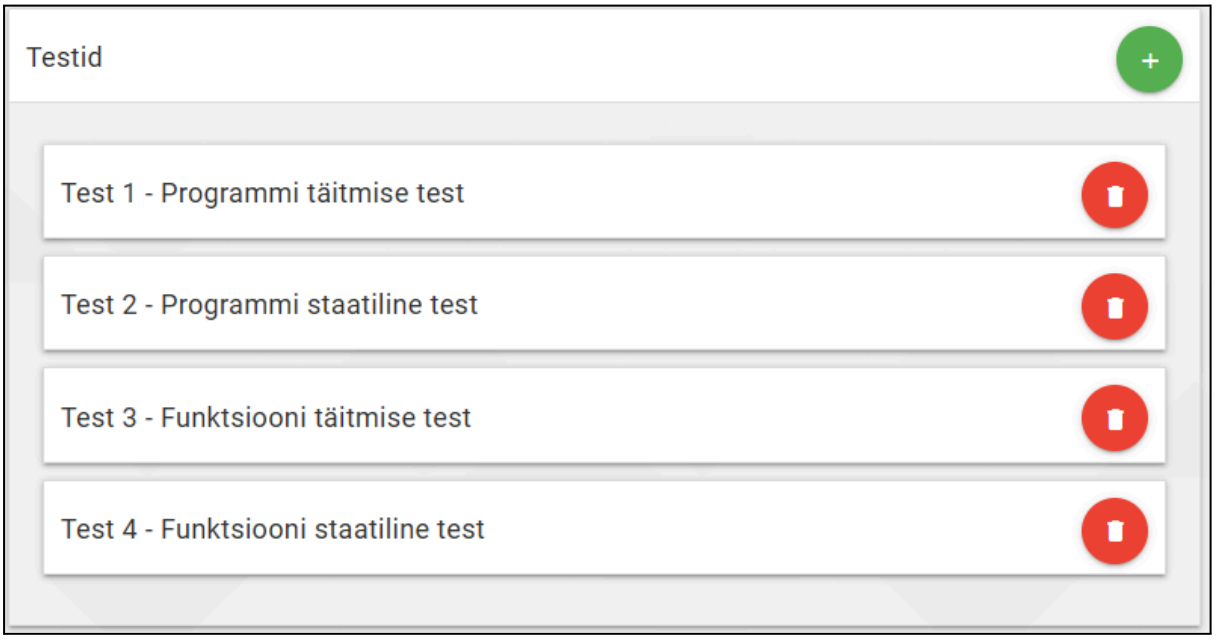
Materialize on eessüsteemi raamistik, mis põhineb materjalidisainil [16]. Materjalidisain on Google'i poolt aastal 2014 loodud avatud lähtekoodiga disainimise süsteem. Materjalidisainis võetakse eeskuju päriselust, kus näiteks teatud materjalid ja pinnad peegeldavad valgust eri moodi ning on erineva tekstuuriga, ja selle omapära on seotud graafilise disaini parimate tavadega [17]. Materialize kasutab materjalidisaini põhitavasid ja selle abil on loodud kasutajaliidese jaoks vajaminevad põhielemendid, mida kasutatakse ka antud lõputöö kasutajaliideses. Alustades tavalisest nupust ja lõpetades modaalidega, on need Materialize'is olemas.

Kasutajaliidese üldine struktuur sai tehtud HTML-is, mille abil pandi paika põhiline loogika ehk raamistik, mille sees testid hakkavad olema. CSS-i kasutati vähe, sest suurem osa kujundusest tuli Materialize raamistikust. Põhiline CSS-i kasutus oli nuppude ja teksti paigutus ning nende väljanägemine. Eriloogika loomisel kasutati Javascripti. Näiteks, kui valida rippmenüüst test, mis tohib esineda ainult ühe korra, siis see test läheb rippmenüü valikus halliks ja seda ei saa enam uuesti valida. Valitud testi kustutamisel tehakse test rippmenüüs jälle valitavaks.

6.2 Kasutajaliidese tutvustus

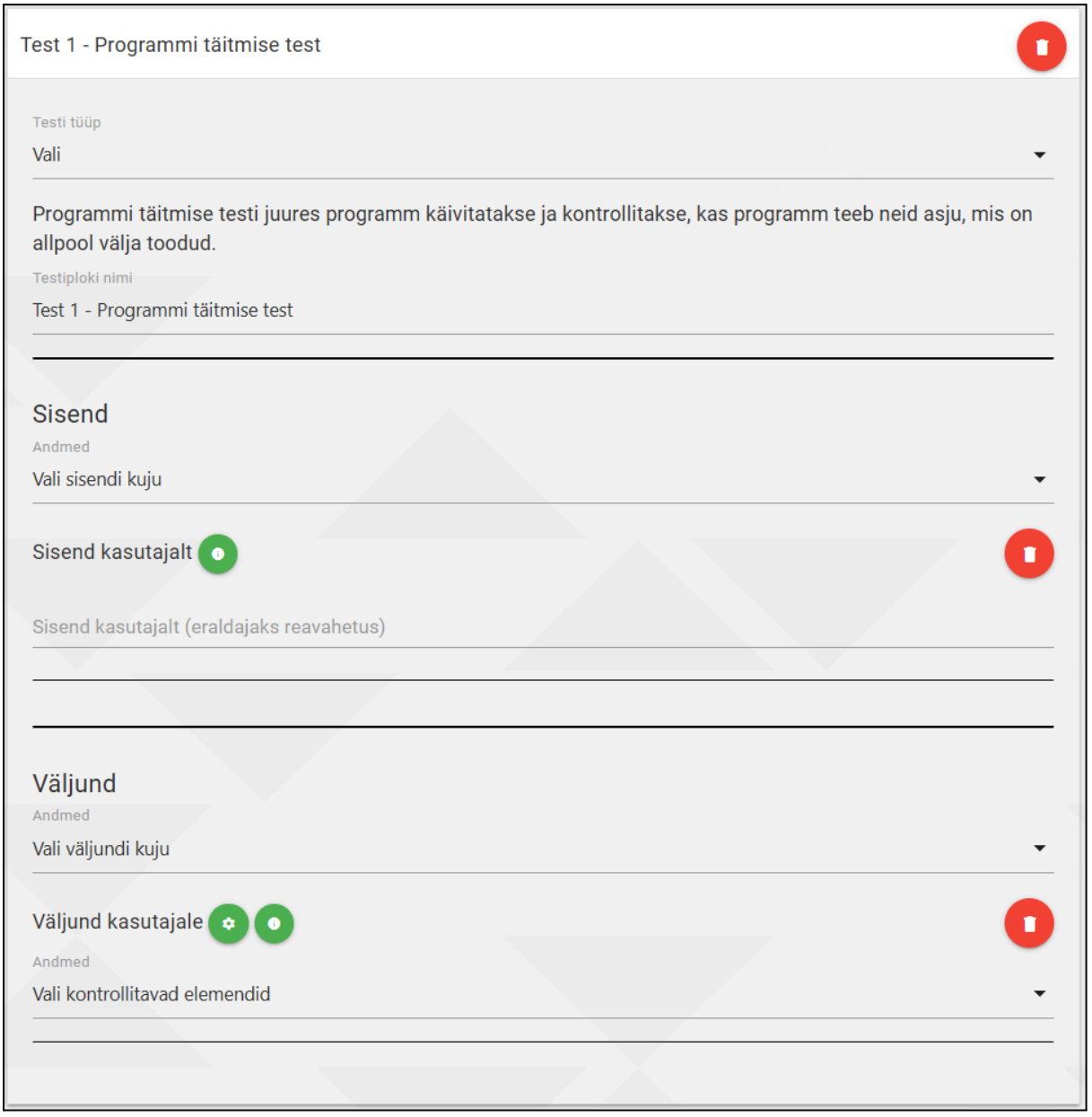
Järgnevalt tutvustatakse kasutajaliideses olevaid võimalusi ja funktsionaalsusi.

Joonisel 13 on näha kasutajaliidese üldine vaade, kus on eraldi näha kõiki nelja erinevat kontrolli, mis on võimalik koostada. Algselt Figmas loodud prototüübil oli määratud kolm kontrollitüüpi nagu on näha joonisel 11. Nüüd kasutajaliidese implementeerimisel sai see ümber tehtud neljaks põhiliseks testitüübiks, sest nõuete ülevaatamisel sai otsustatud, et nii saab paremini neid kategoriseerida. Teste saab lisada läbi "+" märgiga nupu, millele vajutades lisatakse test testide konteinerisse. Alles peale lisamist saab testi sees määrata, mida testitakse. Kustutamine toimub läbi prügikasti ikooniga nupu.



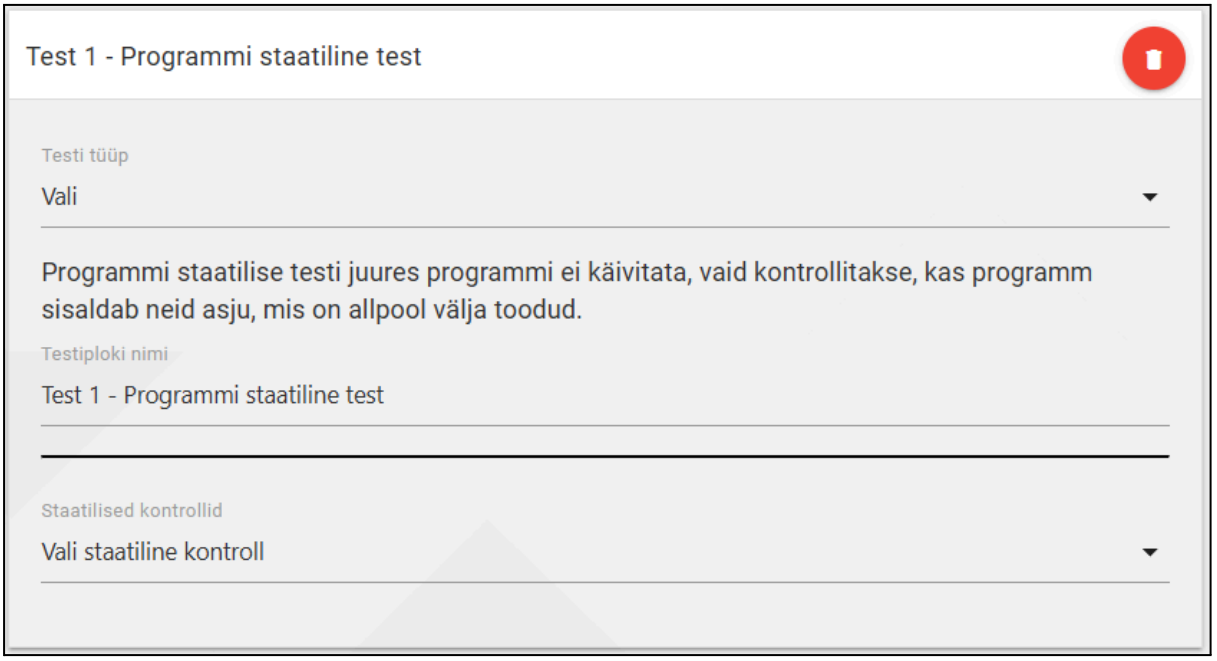
Joonis 13. Üldine kasutajaliidese vaade koos kõigi võimalike testidega.

Joonisel 14 on näha programmi täitmise test. Rippmenüü “Testi tüüp” laseb valida eelneval lehel välja toodud nelja erineva testi vahel. Sisendi ja väljundi valimine toimub läbi rippmenüü, mille kohta joonised tulevad hiljem. Testi sisu sai disainitud selliselt, sest nii on kõik vajaminev informatsioon valitud testi puhul kõik ühes kohas olemas.



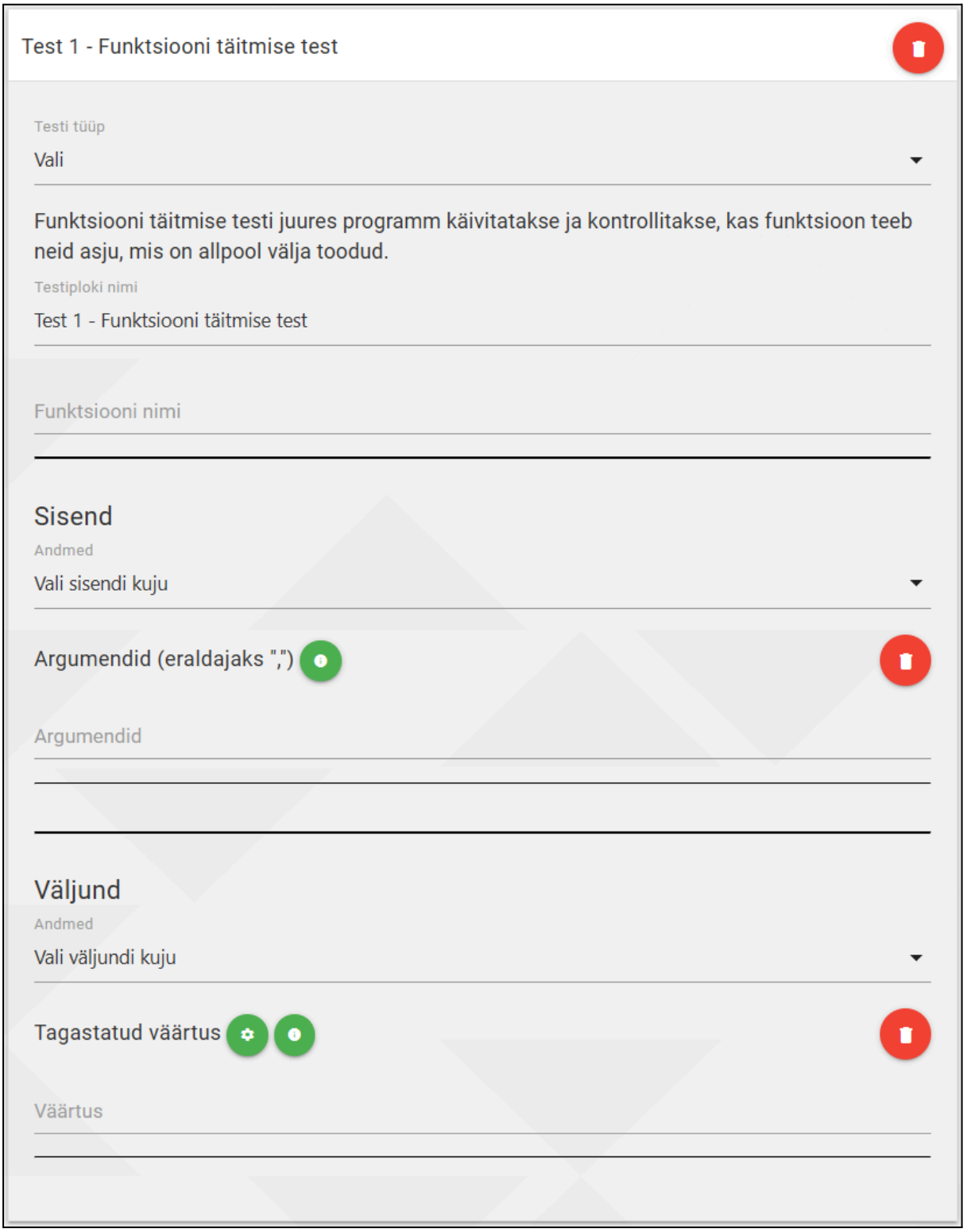
Joonis 14. Programmi täitmise test.

Joonisel 15 on näha programmi staatiline test. Rippmenüü “Vali staatiline kontroll” alt saab valida erinevaid staatilisi kontrolle, näiteks programm sisaldab tsüklit, sisaldab võtmesõna või defineerib funktsiooni.



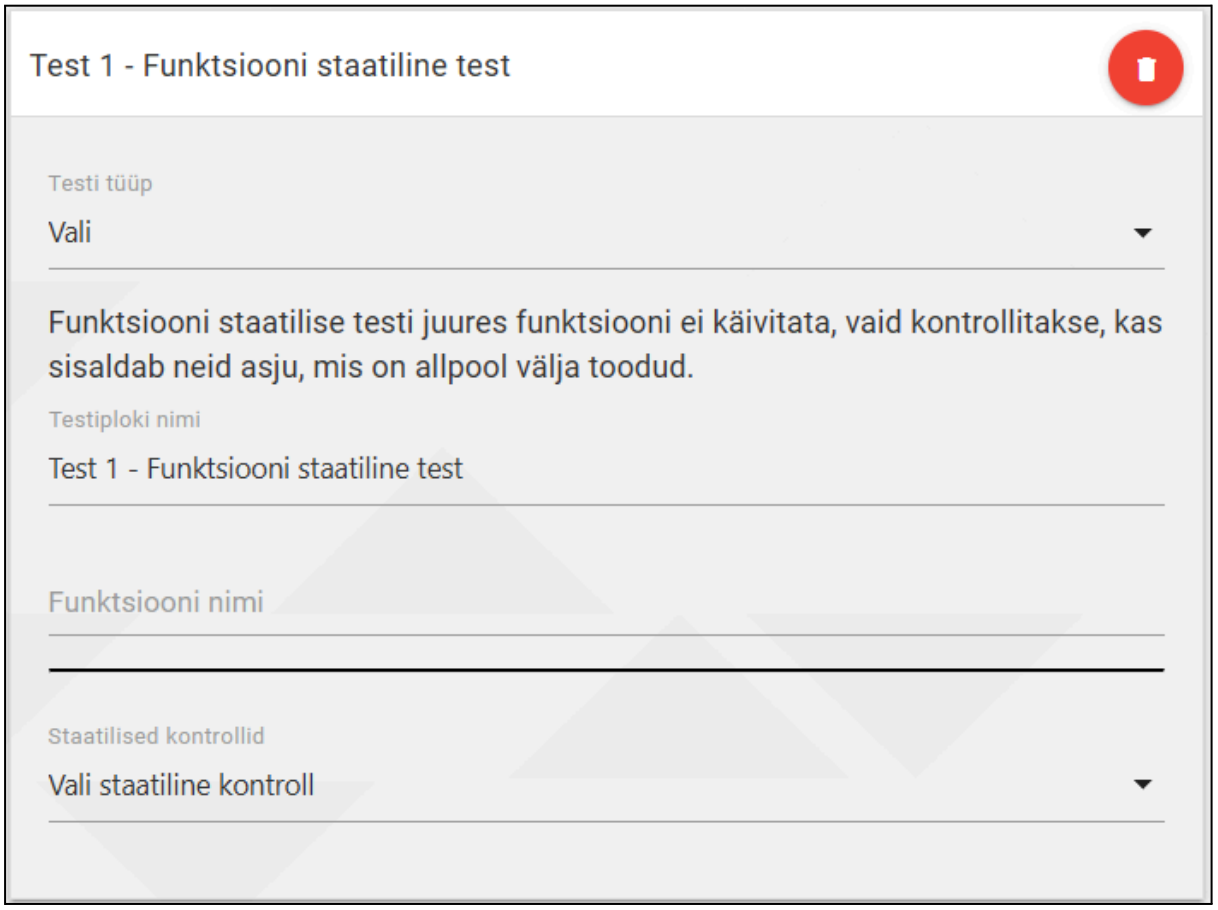
Joonis 15. Programmi staatiline test.

Joonisel 16 on näha funktsiooni täitmise test. Võrreldes programmi täitmise testiga on selle testi puhul võimalik anda ette argumente ja kontrollida lisaks funktsiooni tagastatud väärtust.



Joonis 16. Funktsiooni täitmise test.

Joonisel 17 on näha funktsiooni staatiline test. Analoogiliselt programmi staatilise testiga joonisel 15 saab ka siin valida erinevaid staatilisi kontrolle.



Joonis 17. Funktsiooni staatiline test.

Modaalid on hüpikaknad, mida saab esile kutsuda näiteks nupule vajutamisega. Enamjaolt kasutatakse modaale selleks, et küsida lisainformatsiooni või nõusolekut kasutajalt. Antud kasutajaliideses kasutatakse modaale sisendite või väljundite juures, et kuvada kasutajale informatsiooni või küsida informatsiooni. Modaalide nupud on sarnased lisamise nupule, aga infomodaali sees on täht “i” ja sätete modaali sees hammasratas, nagu on näha joonisel 18. Neid vajutades avaneb hüpikaken vastava sisuga.

Joonise 18. Sätete ja info modaalide nupud.

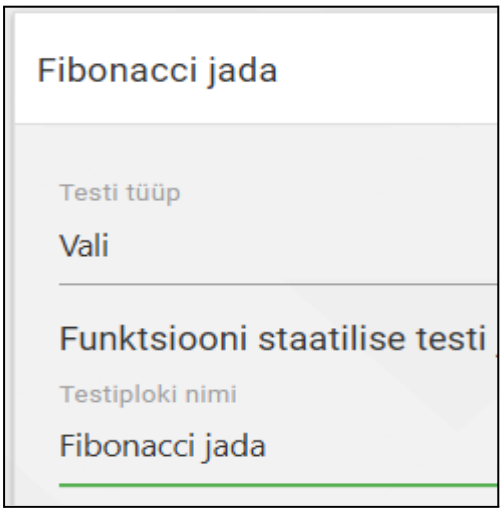
Joonisel 19 on toodud modaal, mis kuvab lisainfot kontrolli “Väljund kasutajale” puhul.

Joonis 19. Näide lisainfo modaalist kontrolli “Väljund kasutajale” puhul.

Joonisel 20 on näha sätete modaali kontrolli “Väljund kasutajale” puhul, kus saab määrata nime, teateid, kasutajale nähtavust ja punkte.

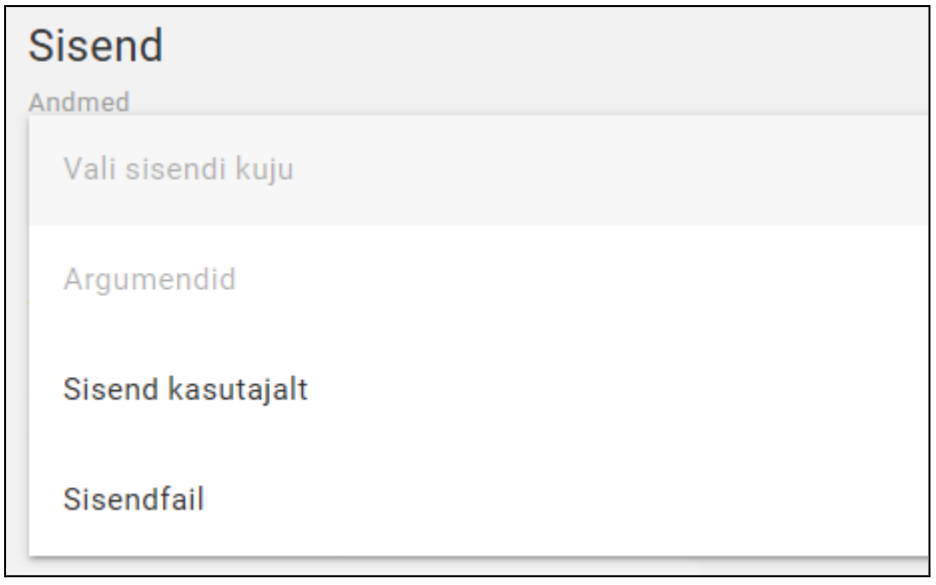
Joonis 20. Näide sätete modaalist kontrolli “Väljund kasutajale” puhul.

Lisatavate kontrollide nimed on muudetavad. Joonisel 21 on näha tekstilahter “Tekstiploki nimi”, kuhu kirjutades enda testi nime, muudetakse testi nime.



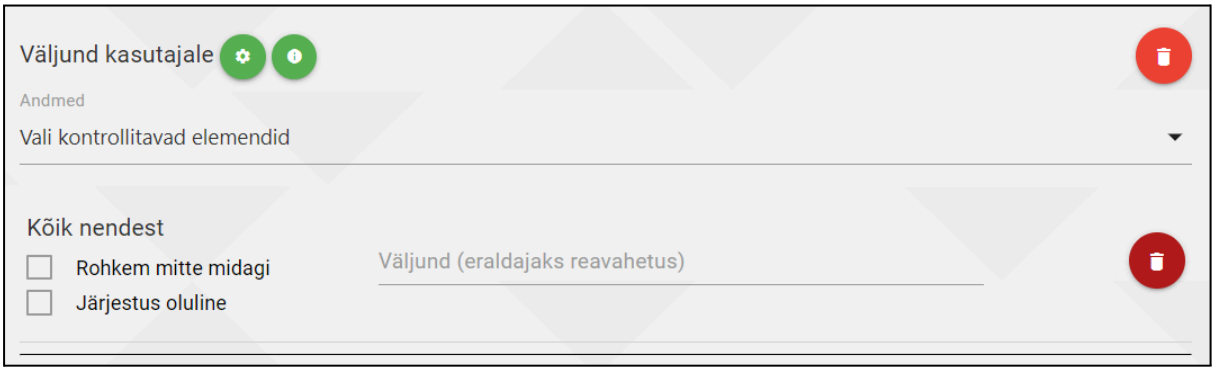
Joonis 21. Muudetud nimega test.

Sisendi või väljundi kuju valimine toimub läbi rippmenüü, mille sisu on vastavalt valitud kontrollile erinev. Joonisel 22 on toodud näide funktsiooni täitmise testi sisendi rippmenüüst, kus saab funktsioonile informatsiooni kaasa anda läbi argumentide, kasutajasisendi või sisendfaili. Teatud sisendeid ja väljundeid saab ka valida korduvalt, näiteks sisendi puhul sisendfaili saab valida korduvalt, kui vaja testida programmi, mis nõuab mitu sisendfaili. Sarnaselt saab ka väljundi puhul valida mitu väljundfaili.



Joonis 22. Sisendi rippmenüü.

Väljundi sisestamisel on mitu erinevat võimalust: “Kõik nendest”, “Mõni nendest”, “Ükskõik millise”, “Mitte ühtegi nendest”, “Puudub vähemalt üks nendest”, “Mitte ühtegi”. Alloleval joonisel on välja toodud variant “Kõik nendest”. Lisaks on märgata kahte erinevat tooni kustutamise nuppu. Heledama taustaga nupule vajutades kustutatakse ära terve väljundi plokk, mis hõlmab endas ka “Kõik nendest” valikut. Tumedama taustaga nupule vajutades kustutatakse ära ainult “Kõik nendest” valik.



Joonis 23. Jaotises "Väljund kasutajale" valiti "Kõik nendest".

Edasises kirjeldatakse arendamise protsessi ja selle käigus ilmnenud probleeme ning nende lahendusi.

6.3 Kasutajaliidese arendamise protsess

Kogu arendus algas nõuetest, mis testid peavad olemas olema ja mida need sisaldavad. Nõuded olid kirjas dokumendis, mille algselt juhendajad lõid ja mida sai täiendatud vastavalt muudatustele. Algne visioon kasutajaliidesest sai paika pandud Figmas. Visiooni elluviimine oli pidev arendamise ja testimise protsess, kus pärast täienduste tegemist testisid seda autor ja juhendajad. Tihti tuli testimise tagasisidest vajadus teha muudatusi, kas midagi lisada/muuta/eemaldada. Autor testis lisaks ka kasutajaliidese ilmele rohkem funktsionaalset poolt, et tagada kasutajaliidese töökindlus. Lisaks sellele toimusid töö autori ja juhendajate vahel kas iga nädal või vahel üle nädala teisipäeviti koosolekud, kus arutati tehtud tööd ja pandi paika edasised sammud.

6.4 Esinenud probleemid

Arenduse ja testimise käigus tuli välja ka teatavaid probleeme. Kõige problemaatilisem oli kiirus, sest ühest hetkest alates võttis lehekülje laadimine liiga kaua aega. Lahendus sellele oli staatiliste andmete kohandamine, kus need muudeti rohkem dünaamilisemaks, et veebilehe laadimisel oleks väiksem hulk staatilisi andmeid. Lisaks eemaldati üleliigseid protsesse.

Samuti esinesid disaini küsimused ehk kuidas peaksid olema raadionupud/märkeruudud asetatud. Lisaks mängis suurt rolli sõnastus, sest märkeruudu tekst peab olema lühike, konkreetne ja kõigile üheselt mõistetav. Joonisel 24 on näha, kuidas said eelnevalt mainitud disaini küsimused lahendatud.

Joonis 24. Näide märkeruutudest ja nende sõnastustest.

Kui teste on lisatud palju, võis tekkida olukord, kus oli raske vahet teha väljundis olevate testide ja üldiselt sisendi ning väljundi vahel. Selle lahenduseks said implementeeritud eraldusjooned. Sisendi ja väljundi vahele lisati paksem eraldusjoon ning väljundis/sisendis olevate elementide vahele peenemad eraldusjooned, et olukord liiga kirjuks ei läheks. Joonisel 25 on näha eelnevalt mainitud eraldusjooned.

Joonis 25. Näide eraldusjoontest sisendi ja väljundi ning sisendis olevate väärtuste vahel.

Eraldi tähelepanu tuleks pöörata punktide süsteemile ehk kuidas määrata, kui palju punkte on mingisugune test väärt. Algse idee järgi määrab koostaja ise igale testile mingisuguse punktisumma vahemikus ühest sajani. Sellises olukorras peaks koostaja ise meeles pidama testide punktiarvestust, sest lõpuks peab testide punktisumma kokku olema 100. See idee aga ei sobinud, sest olukorras, kus on teste kümneid või rohkem, läheb punktide üle järje pidamine koostajal kiiresti käest ära. Selle lahenduseks tuli kasutusele punktide juures kaalude süsteem, et ei peaks arvet pidama selle üle, et lõpuks saaks kokku täpselt 100 punkti. Kaalude süsteemi puhul näiteks olukorras, kus on 2 testi, millest esimene annab 1 punkti ja teine test 3 punkti, on nende testide väärtused vastavalt 25% ja 75%.

Arendamise käigus ilmus ka erinevaid probleeme seoses tõrgetega. Kõige huvitavam neist oli selline, kus teste kustutades kustutati ära konteiner, mis sisaldas kõiki lisatud teste. Konteineri taastamiseks pidi lehekülge värskendama. Probleem tulenes testi kustutamise loogikast. Kui vajutada kustutamise nupu ääre peale, siis ei tuvastatud, et vajutati kustutamise nupule, vaid vajutati selle ülemelemendile, mille tõttu kustutati rohkem kui vaja. Lahendus probleemile oli lisada kontroll vahele, mis enne kustutamist kontrollib, kas vajutati korrektsele elemendile.

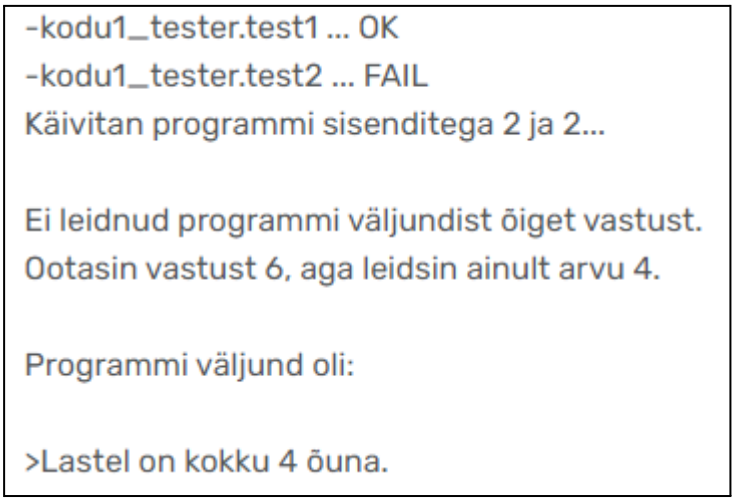
7 Tulemused ja järeldused

Antud peatükis tuuakse välja, millised eelised on loodud kasutajaliidese kasutamisel võrreldes Moodle'is olemasoleva VPL-teegiga. Lisaks loetletakse võimalikud edasiarendused.

7.1 Kasutajaliidese eelised võrreldes Moodle'iga

Põhilised põhjused, miks tekkis soov kasutajaliidese järele, olid Moodle'is automaattestide loomise ning muutmise aeglus ja haldamise keerulisus. Moodle'is peab automaatteste looma neid programmeerides, aga kasutajaliidese abil ei pea koodi kirjutama ja saab lihtsalt rippmenüüst valides ning välju täites automaatteste luua. Järgnevalt on välja toodud kasutajaliidese eelised Moodle ees:

1) Moodle'is koosneb iga automaattest Pythoni failidest, mis kopeeritakse iga ülesande juurde eraldi, seetõttu on automaattestide haldamine keeruline ja ajakulukas. Näiteks kui avastatakse viga ühes failis, mida kasutatakse paljudes automaattestides, siis peab käsitsi kõik need testid läbi käima ja vea ära parandama. Kasutajaliidese abil loodud automaattesti saab kasutada paljudes erinevates ülesannetes ja automaattestis vea avastamise korral tuleb see ainult ühes kohas ära parandada ning tehtud parandus rakendub kõikjal, kus antud automaattesti kasutatakse.
2) Kasutajale tagasiside andmine nii positiivselt kui ka negatiivselt läbitud testide puhul. Moodle'is toimub täpsem tagasiside andmine ainult sellisel juhul, kui automaattesti ei läbitud edukalt, nagu on näha joonisel 26. Korrektse lahenduse korral kuvatakse kasutajale lihtsalt “OK”.



Joonis 26. Moodle’i automaattesti tagasiside.

3) Kasutajaliideses saab igale testile määrata punktisumma, mis näitab kui palju vastava automaattesti eest korrektse lahenduse korral punkte saab. Moodle'is hetkel selline võimalus puudub.

Eelnevalt toodud eeliste loetelu ei ole lõplik. Nimetatud loetelu võib pikeneda, kui võtta arvesse võimalikke edasiarendusi, mida tutvustatakse järgmises alapeatükis.

7.2 Võimalikud edasiarendused

Järgnevalt on toodud nimekiri võimalikest edasiarendustest, mis kas esialgu ei olnud töö skoobis või osutusid mahuliselt liiga aeganõudvaks:

1) Objektorienteeritud programmeerimisega seotud kontrollid, näiteks võiks saada kontrollida klasside olemasolu ja nendega tehtavaid operatsioone.
2) Kasutajaliideses testide järjestamine, mille korral saaks kasutaja ise määrata testide järjekorda. Hetkel määratakse järjekord testide lisamise alusel, mis tähendab, et järjekorda on siiski võimalik muuta, kuid see ei ole intuitiivne ning kasutajasõbralik.
3) TSL-i loomine kasutajaliidese abil ja kasutajaliidese loomine TSL-i alusel.
4) Java programmeerimiskeele tugi. Hetkel on fookus olnud ainult Pythoni programmeerimiskeelel, sest kasutajaliides sai loodud pidades silmas Tartu Ülikooli kursusi “Programmeerimine”, “Programmeerimise alused I” ning “Programmeerimise alused II”, kus programmeerimine toimub Pythonis. Tulevikus oleks soov ka tugi lisada Javale, et kasutajaliidest saaks kasutada ka teistel kursustel, mis ei kasuta vaid Pythonit põhilise programmeerimiskeelena.

Eelnevalt mainitud loetelus kõige tähtsama olulisusega punktid oleksid esimene ja viimane punkt. Nende täitmisel avaneks võimalus kasutada kasutajaliidest rohkematel kursustel. Peale esimest ja viimast punkti võiks edasi vaadata ülejäänud punkte loetelus.

8 Kokkuvõte

Antud bakalaureusetöö eesmärk oli luua kasutajaliidese prototüüp, mille abil saaks koostada automaatteste ja mis oleks aluseks tegelikule kasutajaliidesele. Lisaks oleks see kasutajaliides tulevikus asendus hetkel kasutusel olevale Moodle'is VPL-teegile baseeruval automaatkontrollide süsteemile.

Kasutajaliidese arendamiseks olid algselt paika pandud nõuded, mida loodav kasutajaliidese prototüüp peab rahuldama. Esialgu tehti Figmas algne prototüüp, et paika panna struktuur ja loogika, mille alusel automaatteste lisatakse ning millised väljad võiksid neil olla. Seejärel alustati kasutajaliidese arendamisega, kus kasutati HTML-i, CSS-i ja JavaScripti põhiliste tööriistadena koos Materialize raamistikuga.

Tulevikus on võimalik kasutajaliidest edasi arendada, täiendades seda erinevate funktsionaalsustega nagu Java programmeerimiskeele tugi, objektorienteeritud kontseptsioonide testid, TSL-i loomine kasutajaliidesest ning erinevad kasutajaliidese täiendused, mis hõlmavad endas visuaalset disaini ja kasutajaliidesele elementide lisamist.

Viidatud kirjandus

[1] Tartu Ülikool õppeinfosüsteem ÕIS. Programmeerimise alused (3 EAP). https://ois2.ut.ee/#/courses/MTAT.03.236/version/03fed275-4c6c-930b-8019-ab03dfb78130/details (vaadatud 05.05.2024).

[2] Tartu Ülikool õppeinfosüsteem ÕIS. Programmeerimise alused II (3 EAP). https://ois2.ut.ee/#/courses/MTAT.03.256/version/67a8d9d1-c977-7eed-d9c9-c73bc8c61e10/details (vaadatud 05.05.2024).

[3] Tartu Ülikool õppeinfosüsteem ÕIS. Programmeerimine (6 EAP). https://ois2.ut.ee/#/courses/LTAT.03.001/version/05371a85-939c-bb06-dfa6-d7f7397ee8a4/details (vaadatud 05.05.2024).

[4] Wong E. User Interface Design Guidelines: 10 Rules of Thumb. *Interaction Design Foundation - IxDF*, 2024. URL: https://www.interaction-design.org/literature/article/user-interface-design-guidelines-10-rules-of-thumb (vaadatud 10.05.2024).

[5] Wong E. Shneiderman's Eight Golden Rules Will Help You Design Better Interfaces. *Interaction Design Foundation - IxDF*, 2024. URL: https://www.interaction-design.org/literature/article/shneiderman-s-eight-golden-rules-will-help-you-design-better-interfaces (vaadatud 10.05.2024).

[6] Horsky J., Schiff G. D., Johnston D., Mercincavage L., Bell D., Middleton B. Interface design principles for usable decision support: A targeted review of best practices for clinical prescribing interventions. *Journal of Biomedical Informatics*, 45(6), 2012, 1202-1216. DOI: 10.1016/j.jbi.2012.09.002.

[7] Allam A. H., Hussin A. R. C., Dahlan H. M. User Experience: Challenges and Opportunities. *Journal of Information Systems Research and Innovation*, 3(1), 2013, 28-36. URL: https://seminar.utmspace.edu.my/jisri/download/F1_FinalPublished/Pub4_UserExperienceChallenges.pdf (vaadatud 01.05.2024).

[8] Morville P. User Experience Design. 2004. URL: https://semanticstudios.com/user_experience_design/ (vaadatud 01.05.2024).

[9] Singh S. K., Singh A. Software Testing. Lucknow: Vandana Publications. 2012. URL: https://books.google.ee/books?hl=en&lr=&id=HdKwDwAAQBAJ&oi=fnd&pg=PT3&dq=Singh+S.+K.,+Singh+A..+Software+Testing&ots=7adHp-lCc0&sig=ZqgfsuklTfEUG_hzKTIGCespBYQ&redir_esc=y#v=onepage&q&f=false (vaadatud 01.05.2024).

[10] Tuteja M., Dubey G. A Research Study on importance of Testing and Quality Assurance in Software Development Life Cycle (SDLC) Models. *International Journal of Soft Computing and Engineering*, 2(3), 2012, 251-257. URL: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=605c413a3bd7eb537338480113b33ea911f3987b (vaadatud 01.05.2024).

[11] Harish A. When NASA Lost a Spacecraft Due to a Metric Math Mistake. 2020. URL: https://www.simscale.com/blog/nasa-mars-climate-orbiter-metric/#:~:text=In%20September%20of%201999%2C%20after,%2C%20i.e.%2C%20the%20metric%20units (vaadatud 01.05.2024).

[12] Lahendus. URL: https://lahendus.ut.ee (vaadatud 26.02.2024).

[13] Rodriguez-del-Pino J. C., Rubio-Royo E., Hernández-Figueroa Z. J. A Virtual Programming Lab for Moodle with automatic assessment and anti-plagiarism features. 2012. URL: https://www.researchgate.net/publication/275652921_A_Virtual_Programming_Lab_for_Moodle_with_automatic_assessment_and_anti-plagiarism_features (vaadatud 04.05.2024).

[14] Muuli E., Palm R., Lepp M. Simplifying the creation and maintenance of automated assessments of programming tasks via Test Specific Language. *Proceedings of the 2022 6th International Conference on Education and E-Learning*, 2022. DOI: https://doi.org/10.1145/3578837.3578840

[15] Figma: The Collaborative Interface Design Tool. URL: https://www.figma.com/ (vaadatud 05.05.2024).

[16] Materialize: A modern responsive front-end framework based on Material Design. URL: https://materializecss.com/about.html (vaadatud 10.05.2024).

[17] Material Design. URL: https://m2.material.io/design/introduction (vaadatud 19.02.2024).

[18] Vallaste H. E-teatmik. URL: http://vallaste.ee/.

Lisad

I. Valminud kasutajaliidese Githubi repositoorium

Antud lõputöö käigus valminud kasutajaliidese peafail on kättesaadav järgnevalt veebiaadressilt.
https://github.com/Kaelkirjak/TSL-kasutajaliides/tree/master/wui/static/tsl-ui.html

Valminud kasutajaliidese koodirepositoorium asub järgneval veebiaadressil, mis sisaldab ka faile, mis ei ole loodud töö autori poolt, kuid on vajalikud kasutajaliidese toimimiseks.
https://github.com/Kaelkirjak/TSL-kasutajaliides/tree/master

II. Figma prototüüp

Figma prototüüp asub järgneval veebiaadressil.

https://www.figma.com/design/PbCG0zQ3T4PeyiZQQCOJhh/Kasutajaliides?node-id=0%3A1&t=1CH07pwRGb5KJfgt-1

III. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Priit Pärn,

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose **Automaatkontrollide loomise süsteemi kasutajaliides**, mille juhendajad on Reimo Palm ja Eerik Muuli, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Priit Pärn
14.05.2024