TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Jolanta Rudus 206231IABB

Kasutajaliidese automaattestimise raamistike analüüs ja võrdlus ettevõttes MRPeasy OÜ kasutusele võtmiseks

Bakalaureusetöö

Juhendaja: Jekaterina Tšukrejeva
Magistrikraad

Tallinn 2023

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: Jolanta Rudus

17.05.2023

Annotatsioon

Tänapäeval on olemas palju erinevaid automaattestimise raamistikke ja kõigil on omad eelised ja puudused. Nende arvukuse tõttu on keeruline valida endale kõige sobilikum vaid üldises kasutuses oleva informatsiooni põhjal. Vale otsuse korral tuleb raamistikku vahetada, mis on väga aeganõudev ja kulukas ettevõtmine. Käesoleva bakalaureusetöö eesmärgiks oli uurida ja võrrelda TESTRIG andmete järgi kaheksat 2023. aastal enim nõutud veebirakendustele mõeldud testimisraamistikke. Raamistikke võrreldi vastavalt autori ja ettevõtte MRPeasy OÜ koostööna koostatud kriteeriumitele. Kriteeriumid olid jagatud kaheks osaks. Kriteeriumite esimene osa tugines uurimistulemustele ning teine osa autori tehtud praktilisele osale. Saadud võrdluse põhjal analüüsiti ja jõuti järeldusele, milline raamistik automaattestide kirjutamiseks sobib ettevõttesse MRPeasy OÜ kõige rohkem.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 37 leheküljel, 6 peatükki, 10 joonist, 2 tabelit.

Abstract

Analysis and comparison of user interface automation testing frameworks for the implementation in MRPeasy OÜ

Nowadays, there are a lot of different automated testing frameworks available, each with its own advantages and disadvantages. Due to their abundance, it is difficult to choose the most suitable one based on general usage information alone. If the wrong decision is made, the framework will have to be changed, which is a very time-consuming and costly undertaking. The aim of this bachelor's thesis was to investigate and compare eight testing frameworks for web applications that will be in high demand in 2023, according to TESTRIG data. The frameworks were compared according to criteria compiled by the author and the company MRPeasy OÜ. The criteria were divided into two parts. The first part was based on research results, and the second part was based on the author's practical work. Based on the comparison, an analysis was made and a conclusion was reached as to which framework is most suitable for MRPeasy OÜ.

The thesis is in Estonian and contains 37 pages of text, 6 chapters, 10 figures, 2 tables.

Lühendite ja mõistete sõnastik

Tagarakendus	Rakenduse koodi osa, mis vastutab andmete salvestamise ja töötlemise eest.
Eesrakendus	Rakenduse koodi osa, mis vastutab kujunduse ja kasutaja ning rakenduse suhtluse eest.
Agiilne arendus	Tarkvaraarendus, mis toimub korduvate tsüklitena, on kiirelt muudetav ning olude ja vajaduste järgi kohandatav.
Jira tööriist	Jira on vigade jälgimise tööriist, mis võimaldab tarkvaraarendajatel oma tööd kiiremini planeerida, jälgida ja kooskõlastada.
Iteratsioon	Järkjärguline kordamine, kuni jõutakse tulemuseni. Iga järgmise korduse puhul toetutakse eelneva seisu andmetele.
Päring	Informatsiooni leidmine mingis infosüsteemis, järelepärimine.
Operatsioonisüsteem	Tarkvara, mis juhib arvuti, telefoni või muu seadme talitlust, näiteks Windows, Android, Linux.
Dünaamiline veebirakendus	Rakendus, mis uuendab andmeid reaalajas, vastavalt päringule, näiteks klient vajutab nuppu ja hinnad arvutatakse ümber.
Graafiline kasutajaliides	Kasutajaliidese vorm, mis võimaldab kasutajatel suhelda elektrooniliste seadmetega graafiliste ikoonide ja elementide kaudu.
URL	Mingit ressursi aadress internetis.
HTTP-server	Tarkvara programm, mis täidab klient-server mudelis serveri rolli rakendades HTTP võrguprotokolli.
Draiver	Tarkvara riistvaraga liidestatav seadme juhtprogramm.
Headed brauser	Tavaline brauser kasutajaliidesega.
Headless brauser	Brauserit ilma kasutajaliideseta. Mõned brauserid võimaldavad käsitleda seda kui režiimi ja nende vahel ümber lülituda.
Dokumendi objektimudel	Platvormist ja keelest sõltumatu XML-, XHTML-ja HTML-dokumentidega suhtlemise liides.
Rakendusliides	Arvutiprogrammides alamprogrammi määratluste, protokollide ja tööriistade komplekt tarkvaraga töötamiseks.
Placeholder	Dokumendi objektimudeli muutuja, millele määratakse hiljem mingisugused väärtused.
Identifikaator	Mingi üksuse unikaalne kood, et viidata üksusele.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäevarakendused on keerulised. Seetõttu tekib testimismeeskonnal vajadus kontrollida väga palju erinevaid stsenaariume, mis omakorda kulutab väga palju inimressursse ja aega. Ei või jätta tähelepanuta ka seda, et töötajad võivad unustada midagi kontrollida ning viga jõuab ikkagi lõpuks klientideni. Vigane tarkvara võib mõjutada ettevõtte edukust ja kasutajate arvu. Paremaks ja efektiivsemaks inim-ja ettevõtete ressursside kasutamiseks on kasutusele võetud automaattestimine. Automatiseeritud testimine aitab vähendada inimvigade faktorit ja topelt töö mahtu. Samuti soodustab väiksemat kvaliteedikontrolli meeskonda, vähendab ettevõtte tööjõukulu ja võimaldab tarkvaratestimise meeskonnal keskenduda uuematele, suurematele ja tähtsamatele uuendustele.

1.1 Probleemi taust ja kirjeldus

Käesoleva töö autor töötab ettevõttes MRPeasy OÜ tarkvaratestijana. Ettevõte on IT-valdkonnas tegutsev idufirma, mille arendustempo on kiire ning ressursside hulk piiratud.

Firma tarkvaratestimise meeskond tegeleb iga uuenduse testimisega käsitsi. Seetõttu kulub iga muudatuse või paranduse läbivaatamisele väga palju ressursse, sest on vaja kontrollida nii uuenduse kui ka olemasoleva funktsionaalsuse korrektset tööd. Pidev olemasoleva funktsionaalsuse testimine võtab oma mahu tõttu palju aega ning selle rutiinsuse pärast võivad vead märkamata jääda. Eelnevalt kirjeldatud probleemi on võimalik lahendada vaid automatiseeritud testimisega, kuna automaatteste on võimalik käivitada pidevalt ning nendega kaetud funktsionaalsus ei pea olema tihti testitud töötajate enda poolt. Sellepärast otsustati ettevõttes juurutada automaattestimist.

Automatiseeritud testimine on muutumas aina populaarsemaks ning selle jaoks on olemas mitu tööriista. Kuna kõigil neist on omad eelised ja puudused, tekkis vajadus võrrelda ja valida ettevõttesse kõige paremini sobiv tehnoloogia. Selle teostamiseks telliti käesoleva töö autorilt selleteemaline uurimus.

1.2 Eesmärk

Käesoleva töö eesmärgiks on uurida ja võrrelda 2023. aasta TESTRIG uurimuse põhjal kaheksat enim nõutud veebirakendustele mõeldud testimisraamistikku läbivtestide tegemiseks. Võrdlust teostatakse vastavalt ettevõtte MRPeasy OÜ ja üliõpilase määratud nõuetele, mis on jaotatud kaheks osaks. Kriteeriumite esimene osa põhineb uurimistulemustel ning teine osa autori tehtud praktilisel osal. Saadud võrdluse põhjal analüüsitakse ja jõutakse järeldusele, milline raamistik automaattestide kirjutamiseks sobib ettevõttele MRPeasy OÜ kõige rohkem.

1.3 Metoodika

Töö eesmärgini jõudmiseks alustatakse ettevõtte MRPeasy OÜ tutvustusest, et oleks paremini aru saada, mis ettevõttega on tegu ning millised on nende vajadused. Seejärel koostatakse autori ja ettevõtte koostööna kriteeriumid, mille põhjal hakatakse edaspidi raamistikke võrdlema. Autor koostab esialgsed nõuded tuginedes vestlusele ettevõtte testijatega ja nende tööprotsessidele. Seejärel uuritakse manuaalset ja automatiseeritud testimist, et saada nendest rohkem teada. Uuritakse nende eeliseid, puuduseid, peamisi kontseptsioone ning mis olukordades on neid parem kasutada. Sellega soovitakse saada kinnitust, et ettevõte vajab oma probleemi lahendamiseks automaattestimise kasutuselevõtmist.

Selleks, et teha valik, milliseid automaattestimisraamistikke võrrelda kasutatakse 2023. aasta TESTRIG uurimust enim nõutud testimisraamistikest. Võrreldakse kõigi raamistike eeliseid ning puuduseid vastavalt eelnevalt paika pandud kriteeriumite esimesele osale ning elimineeritakse ebasobivad töövahendid. Järgnevalt analüüsitakse võrdlusesse jäänud raamistikke vastavalt kriteeriumite teisele poolele, et teha otsus, milline raamistik sobib ettevõttesse kõige enam.

1.4 Töö struktuur

Käesoleva bakalaureusetöö on läbi viidud autori poolt 2022/2023 õppeaastal. Töö koosneb kuuest peatükist. Esimeses peatükis kirjeldatakse probleemi ja tausta, sõnastatakse eesmärk ning pannakse paika eesmärkide saavutamise metoodika. Teises peatükis tutvustatakse ettevõtet MRPeasy OÜ, selle meeskonda, kasutusel olevaid

tööriistu ning määratakse autori ja ettevõtte koostööna loodud nõuded, mille põhjal valitakse töö lõpus ettevõttele sobiv testimisraamistik. Kolmandas peatükis tehakse ülevaade manuaalsest ja automaattestimisest, nende eelistest, puudustest ning kontseptsioonidest. Selle peatükis tulemusena otsustatakse, kas ettevõte vajab ikka automaattestimist või mitte. Neljandas peatükis analüüsitakse raamistike vastavust kriteeriumite esimesele poolele ja saadud tulemuste põhjal elimineeritakse osad raamistikud. Viiendas peatükis analüüsitakse võrdlusesse jäänud raamistike paigaldamise kiirust dokumentatsioonide põhjal ning kirjeldatakse ühte testjuhtumit, mille põhjal luuakse raamistikele ka päris test. Samuti viiakse läbi küsitlus ettevõtte testijatega, et saada teada nende arvamust raamistike koodist ja kasutajaliideste mugavusest. Seejärel tehakse kõigi kriteeriumitega võrdlev tabel ja jõutakse otsusele, milline raamistikest valida. Kuuendas peatükis tehakse kokkuvõte töö tulemustest.

2 MRPeasy OÜ veebirakendus ja meeskond

2.1 MRPeasy OÜ

MRPeasy OÜ on IT-ettevõte, mis asutati 2014. aastal. Ettevõttes töötab 31.03.2023 seisuga 27 töötajat, kellest 6 on arendajad ja 3 testijad. Ettevõte tegeleb pilvepõhise tootmise planeerimise, jälgimise ja laovarude juhtimise tarkvara arendamisega väikestele ja keskmistele tootmisettevõtetele, kus töötab ligikaudu 10 kuni 200 töötajat [1], [2].

Tarkvara võimaldab täita paljusid tootmises nii klientidele kui ka tootjatele olulisi funktsioone. Üheks sellistest funktsioonidest on näiteks kohe peale tellimuse saamist täpse teostusaja ja hinna arvutamine. Tootjale on eriti mugav laoseisu haldamise funktsioon, mis võimaldab jälgida, kui palju varuosasid on laos tootmiseks olemas, kui palju on veel vaja, ning millal ja kui palju osi jõuab tarnijatelt lattu, et neid saaks samuti töösse võtta [1].

Vaatamata sellele, et MRPeasy tarkvara on kasutajasõbralik ja arusaadav, seisneb selle keerukus just erinevates võimalikes funktsioonides ja nende kombinatsioonides. Peale eelnevalt nimetatud funktsioonide, on MRPeasy rakenduses ka väga palju teisi funktsioone: 12 rakenduse kasutamist lihtsustavat funktsiooni; 30 tootmise, materjalide planeerimise ja aruandluse funktsiooni; 22 laojuhtimise funktsiooni; 14 müügi ja tellimuste haldamise funktsiooni; 14 hanke funktsiooni; 20 finantside haldamise ja raamatupidamise funktsiooni; 13 turvalisuse funktsiooni ja 11 integratsiooni teste rakendustega (peamiselt e-kaubanduse platvormid) [3].

2.2 Arendamiseks juba kasutusel olevad tööriistad ja raamistikud

Rakenduse tagarakenduse arendamisel kasutatakse PHP programmeerimiskeelt ja eesrakenduse arendamiseks React raamistikku ning JavaScripti programmeerimiskeelt.

Arendamine toimub Kanban raamistiku metoodika alusel. Kanban on agiilset arendust toetav raamistik, mille eesmärgiks on hallata tööd prioriteetide alusel nii, et ülesanded,

mida võetakse oleksid olemasolevatest ülesannetest kõrgema prioriteediga. Selline lähenemine aitab efektiivselt kasutada olemasolevaid inimressursse ja parandada süsteemi kitsaskohtade käsitlemist. Ettevõttes jälgitakse peamisi Kanban raamistiku põhimõtteid. Tööd visualiseeritakse, et oleks hea jälgitavus ning võimalusel piiratakse pooleliolevat tööd, et mitte tekitada palju pooleliolevaid ülesandeid. Arendajad keskenduvad korraga ühe kindla ülesande täitmisele ja selle klientideni jõudmisele. Meeskonnaliikmed keskenduvad aktiivselt enda ülesannetele vastavalt nende prioriteetidele [4], [5].

Töö haldamine ja visualiseerimine toimub Jira tahvli kaudu, kus kõik töötajad näevad ülesandeid koos prioriteetidega ja kui kaugele on jõudnud nende lõpetatud ülesanded. Näiteks, kas ülesanne läbis koodikontrolli ja testijate tiimi kontrolli. Samuti nähakse, kui kaugel on teiste meeskonnaliikmete ülesanded. Tooteomanik võib tööde prioriteete ümber seada ilma meeskonda häirimata, sest kõik muudatused, mis tehakse väljaspool töös olevaid ülesandeid ei mõjuta meeskonna tööd [4], [6].

2.3 Kriteeriumid testimisraamistiku valimiseks

Ettevõttele MRPeasy OÜ sobiliku testimisraamistiku valimiseks töötas käesoleva töö autor välja peamised kriteeriumid, et valida, milline testimisraamistikest sobib ettevõttele kõige enam. Nõuded loodi tuginedes ettevõtte testijatega suhtlemisele ning nende tööprotsessi jälgimisele. Ettevõttel ega töötajatel polnud raamistike eelistusi. Peale nõuete esialgse versiooni koostamist, kooskõlastas autor need ettevõttega. Seejärel vastavalt nende soovidele tehti kriteeriumites vajalikud muudatused ja täiendused. Peale seda esitati ettevõttele uus nõuete komplekt, mis oli ettevõtte poolt heaks kiidetud.

Esialgsed kriteeriumid, mis sobisid:

- Võimalikult arusaadav testide kirjutamise süntaks ettevõtte arendajatele;
- kiire paigaldamisega;
- kerge kasutuselevõtmisega;
- võimalus integreerida sidusarenduskeskkonnaga;
- peab toetama nii *headed* kui ka *headless* brauserite režiime;
- peab sisaldama võimalust testida alla laetud faile, nagu PDF;

- soovituslikult võiks toetada JavaScripti, kuna kvaliteedikontrolli tegijatel on sellega eelnev kogemus;
- tasuta kasutamise võimalus.

Lisatud kriteeriumid peale juhatusega autori koostatud kriteeriumite üle vaatamist:

- Raamistiku graafiline kasutajaliides peab olema töötajatele mugav;
- peab olema täpse ja põhjaliku dokumentatsiooniga;
- peab olema laialdaselt kasutatav ehk populaarne.

Autor jaotas saadud kriteeriumid kaheks osaks. Esimeses osas on need, mida saab uurimuse tulemusena hinnata. Teises osas on need, mida kontrollitakse töö praktilises osas paigaldades esimeste kriteeriumite järgi sobinud raamistikud projektidesse, tehes teste ning viies läbi ettevõtte töötajatega küsitlus.

Kriteeriumid, mille vastavust peeti võimalikuks leida uurimuse tulemusena:

- Võimalus integreerida sidusarenduskeskkonnaga;
- peab toetama nii *headed* kui ka *headless* brauserite režiime;
- peab sisaldama võimalust testida alla laetud faile, nagu PDF;
- soovituslikult võiks toetada JavaScripti, kuna kvaliteedikontrolli tegijatel on sellega eelnev kogemus;
- tasuta kasutamise võimalus;
- peab olema täpse ja põhjaliku dokumentatsiooniga;
- peab olema laialdaselt kasutatav ehk populaarne.

Kriteeriumid, mida hinnatakse töö praktilises osas:

- Raamistiku graafiline kasutajaliides peab olema töötajatele mugav;
- võimalikult arusaadav testide kirjutamise süntaks ettevõtte arendajatele;
- kiire paigaldamisega;
- kerge kasutuselevõtmisega.

3 Tarkvara testimine ja selle olulisus

Tarkvara testimine on protsess, mille abil hinnatakse ja kontrollitakse, kas tarkvararakendus teeb seda, mida see tegema peaks. Vigadena käsitletakse kõiki töö käigus leitud erinevusi disainist või juhendist. Testimine võimaldab vältida vigade jõudmist klientideni, vähendada arendamise kulusid ja parandada jõudlust. Testimist on kahte tüüpi: käsitsi ja automatiseeritud testimine. Mõlemal tüübil on omad olukorrad, kui neid tuleks kasutada [7], [8].

Vastavalt Østfold University College infotehnoloogia osakonna tudengite läbiviidud uuringule on manuaalne ja autmatiseeritud testimine tegelikult samatähtsad ning mõlemad nõuavad nii tehnilisi oskusi kui ka inimfaktorit [9].

Selles peatükis uuritakse, mis on manuaalne testimine ja automaattestimine. Uuritakse nende erinevusi, eeliseid, puudusi ning nende peamisi kontseptsioone.

3.1 Manuaalne testimine

Manuaalne testimine ehk käsitsi testimine on testimisprotsess, mille käigus tarkvaratestija testib tarkvararakendust vigade tuvastamiseks käsitsi. Manuaalse testimise põhiline eesmärk on tagada, et rakendus oleks veatu ja töötaks vastavalt eelnevalt määratud funktsionaalsetele nõuetele. Selline tarkvara testimine on iga testimisstrateegia jaoks hädavajalik. Seda ei saa mingil juhul täielikult asendada automaattestimisega, kuna see aitab saada parema ülevaate lõppkasutaja vaatenurgast, tarkvara kasutatavusest ja kasutajakogemusest. Samuti aitab leida vigu tarkvaraarenduse elutsükli varasemates etappides. Tarkvara testimine ei ole ainuüksi tehniline, vaid ka inimeste käitumise teadlikkust vajav tegevus. Manuaalset testimist teostatakse ilma automatiseerimisraamistiketa [9], [10].

3.1.1 Manuaalse testimise eelised

Käsitsi testimisel on automatiseeritud testimise ees mitmeid eeliseid. Manuaalne testimine kasutab vigade leidmiseks inimeste intelligentsust ja kõrgemaid kognitiivseid

võimeid, nagu deduktiivne arutluskäik ja analüüs. Selline lähenemine võimaldab leida vigu, mida automaattestid võivad märkamata jätta. See on ka peamiseks põhjuseks, miks on alati vaja automatiseeritud testimisvahenditega tehtud testide ülevaatamiseks inimest, et midagi ei jääks märkamata [11].

Teiseks eeliseks on see, et manuaalne testimine võimaldab testijatel rohkem keskenduda keerukamatele funktsioonidele, nende koostööle ja ebatavalistele stsenaariumitele. Iga üksiku keerulise stsenaariumi jäljendamisega võib automaattestide kasutamine kujuneda väga aeganõudvaks. Manuaalne testimine võimaldab automatiseerimisel kulutada vähem aega piiripealsete stsenaariumite testimiseks ja annab tarkvaratestijatele rohkem aega keerukate funktsioonide ja ebatavaliste olukordade läbimõtlemisele [11].

Käsitsi testimine võimaldab testijatel jälgida toote kvaliteeti kogu selle arendustsükli vältel kulutades rohkem aega selle funktsioonidega ja nende uuendustega tutvumiseks. Selline lähenemine aitab hoida testijate teadmisi projekti kohta heal tasemel, mis kujunevad väga kasulikeks koodi muutmisel ja probleemide ilmnemisel. Seetõttu soovitatakse alati enne automatiseerimise kaalumist stsenaariumit käsitsi testida [11].

Neljandaks eeliseks on see, et testijad katsetavad peale koodi käitumise ka teisi olulisi tegureid, näiteks serveri reageerimisaega. Enamik traditsioonilisi automatiseeritud testimise tööriistu on piiratud sellega, mida nad suudavad toote koodis tuvastada [11].

Käsitsi testimine annab testijatele vabaduse kasutada oma teadmisi kasutajakogemuse jäljendamiseks. Automatiseeritud testimisel tekib oht, et testid ei suuda reaalsetes tingimustes simuleerida erinevat tüüpi kasutajakogemusi, mis sõltuvad paljudest muutujatest ja teguritust, näiteks internetiühenduse kiirus ja olemasolu [11].

Automaattestid ei saa testida süsteemi, mille osa nad on. Käsitsi testimine võimaldab kasutada töötajate teadmisi automaattestide enda vigade tuvastamisel, mis võivad märkamatuna vahele jätta vigu, mis võivad muuta rakenduse testimatuks [11].

3.1.2 Manuaalse testimise puudused

Vaatamata sellele, et käsitsi testimisel on automatiseeritud testimise ees mitmeid eeliseid, on sellel ka puudused, millega tuleks arvestada ning miks tuleks kaaluda lisaks manuaalsele testimisele ka automaattestide kasutuselevõtmist.

Üheks peamistest manuaalse testimine puudustest on selle protsessi pikkus. Käsitsi testimine nõuab palju rohkem aega kui automatiseeritud testimine. Automaattestid on võimelised töötama ka üleöö ilma igasuguse järelevalveta. Manuaalne testimine nõuab töötaja reaalset kohalolu [11].

Teiseks puuduseks on selle mõjutatus inimlikest teguritest. Käsitsi testimine on vastuvõtlik inimlikele vigadele, nagu väsimus, keskendumisvõime puudumine ja tähelepanu hajumisest põhjustatud vead. Selliseid vigu on hiljem raskem tuvastada, kui need jäetakse esmase testimise ajal tähelepanuta. Samuti võtab kõikide rakenduses leitud vigade dokumenteerimine ja nende monitoorimine rohkem aega, mistõttu on muudatuste jälgimine nende hilisemal tegemisel raskem [11].

Manuaalsetel testijatel peab olema tootest väga hea arusaam enne selle tõhusat kasutamist ja testimist. See aitab neil tuvastada vigu, mida automaattestid võivad ebapiisavate teadmiste korral märkamata jääda [11].

Manuaalne testimine on kulukas. Seda seetõttu, et teste teevad automaatiseeritud tööriistade asemel inimesed. Automaattestid on odavamad, kuna nende käitamise hind ei sõltu sellest, mitu korda neid käitatakse, vaid pigem sellest, kui palju aega kulub nende kodeerimisele, hooldamisele ja ülalpidamisele [11].

3.1.3 Manuaalse testimise peamised kontseptsioonid

Tegelikkuses saab kõiki testimise kontseptsioone kasutada nii manuaalses testimises kui ka automatiseeritud testimises, kuid üldiselt neid siiski jaotatakse vastavalt sellele, kus on mõtet neid rohkem kasutada. Manuaalsel testimisel on palju erinevaid kontseptsioone, kuid peamised ja kõige rohkem tuntud on musta kasti testimine, valge kasti testimine, ühiktestimine, süsteemi testimine, integratsiooni testimine ja vastuvõtutestid [10], [12].

Musta kasti testimine ehk *black box testing* on tarkvara testimismeetod, mille käigus testitakse tarkvararakenduste funktsioone, teadmata sisemist koodistruktuuri, paigaldamise üksikasju ja sisemisi teid. Musta kasti testimine keskendub peamiselt tarkvararakenduste sisendile ja väljundile ning põhineb täielikult tarkvaranõuetel ja spetsifikatsioonidel. Selline lähenemine hõlmab testimist välise või lõppkasutaja vaatenurgast ehk testimisel ei saa näha tarkvara sisemist tööd. Seda tuntakse ka kui

käitumistesti. Niinimetatud must kast võib olla mis tahes tarkvarasüsteem või rakendus, mida soovitakse testida [13].

Valge kasti testimine ehk *white box testing* on testimistehnika, mille käigus testitakse tarkvara sisemist struktuuri, disaini ja kodeerimist, et kontrollida sisend-väljundvoogu ning parandada disaini, kasutatavust ja turvalisust. Testimisel keskendutakse rakenduse sisemisel tööl ja kõik keerleb sisemise testimise ümber. Valge kasti testimisel on kood testijatele nähtav, seega nimetatakse seda ka *clear box testing, open box testing, transparent box testing, code-based testing* ja *glass box testing* [14].

Ühiktestimine ehk *unit testing* on tarkvara testimise valge kasti testimistehnika, mille käigus testitakse tarkvara üksikuid üksusi või komponente. Eesmärk on kinnitada, et iga tarkvarakoodi üksus toimib ootuspäraselt. Ühiktestimine toimub rakenduse arendamise ajal. Tavaliselt seda teevad arendajad ise peale koodi kirjutamist või koodi kirjutamise ajal. Ühiktestid eraldavad koodi osa ja kontrollivad selle õigsust. Testidega kaetav osa võib olla individuaalne funktsioon, meetod, protseduur, moodul või objekt, mis just valmis sai, või mis otsustati ühiktestidega katta. Ühiktestimine on arendatava uuenduse esimene testimise tase, mis tehakse enne integratsiooni testimist [15].

Süsteemi testimine ehk *system testing* on testimise tase, mis kinnitab tervikliku ja täielikult integreeritud tarkvaratoote. Süsteemi testi eesmärk on hinnata süsteemi täielikke spetsifikatsioone. Tavaliselt on tarkvara vaid üks element suuremas arvutipõhises süsteemis. Lõppkokkuvõttes on tarkvara liidestatud teiste tarkvara-ja riistvarasüsteemidega. Süsteemi testimine on defineeritud kui erinevate testide seeria, mille ainus eesmärk on arvutipõhise süsteemi täielik kasutamine [16].

Integratsiooni testimine ehk *integration testing* on testimise tüüp, kus tarkvaramoodulid integreeritakse olemasolevasse süsteemi ja testitakse kogu süsteemi, et näha, kas need töötavad omavahel õigesti ja mõjutavad teineteise tööd nii nagu peab. Tüüpiline tarkvaraprojekt koosneb mitmest tarkvaramoodulist, mille on kodeerinud erinevad programmeerijad. Selle taseme testimise eesmärk on leida defektid nende tarkvaramoodulite vahelises koostoimes, kui need on integreeritud [17].

Vastuvõtutestid ehk *user acceptance testing* on testimisviis, mille viib läbi lõppkasutaja või klient tarkvarasüsteemi kontrollimiseks ja aktsepteerimiseks enne, kui uus tarkvararakenduse osa viiakse tootmiskeskkonda. Sellist testimist tehakse tavaliselt

suuremate uuenduste puhul testimise viimases faasis klientidega, kes küsisid uuendust või sattusid mingite kriteeriumite järgi valimisse. Seda tehakse pärast funktsionaalsuse, integratsiooni ja süsteemi testimist [18], [19], [20].

3.2 Automaattestimine

Automattestimine on käsitsi testimise protsessi automatiseerimine, mille tegemiseks kasutatakse testitavast eraldiseisvat rakendust. Töö protsessis asendatakse käsitsi testimine automatiseeritud testimistööriistade kogumiga, mis aitab tarkvaratestijatel kontrollida tarkvara kvaliteeti ja otsida tekkinud vigu. Testid simuleerivad korduvalt testitava rakenduse koodi elutsükleid ja võrdlevad seejärel saadud tulemusi eeldatud tulemustega. Teisisõnu on automaattestimine protsess, mille käigus tööriist täidab automaatselt teatud koodina kirja pandud mustri järgi ülesandeid. See vähendab käsitsi testijate pinget ja võimaldab neil keskenduda suurema väärtusega ülesannetele, nagu testitulemuste ülevaatamine ja keerukate stsenaariumite läbimõtlemine ning testimine [21], [22], [23].

Käesoleva töö autor peab oluliseks välja tuua asjaolu, et automaattestimine ja manuaaltestimine on mõlemad väga olulised ning üks ei saa täielikult asendada teist. Vaatamata sellele on automaattestimisel manuaaltestimise ees palju eeliseid, mida järgmises alapeatükis kirjeldatakse [9].

3.2.1 Automaattestimise eelised

Üheks kõige mugavamatest funktsioonidest on see, et automaattestidel saab süsteemis määrata, millal ja kui tihti teste käivitada. Kuna testimisrakendus suudab kogu testimis- ja võrdlemisprotsessi iseseisvalt läbi viia, pole igas iteratsioonis vaja korduvat välist inimese juhtimissisendit, mis säästab nii aega kui ka raha, kuna süsteemil puudub vajadus füüsiliselt puhata, nagu inimeste puhul. Inimeste kohalolu on vaja vaid testide kirjutamisel ja hiljem vigade tekkimisel tulemuste läbivaatamisel [22], [23].

Automatiseeritud testimine on oluline, et saavutada parem testide katvus lühema aja jooksul ning tulemuste suurem täpsus. Ettevõtte arendatav veebirakendus vajab ja saab pidevaid uuendusi, et olla tasemel kõigi uute muudatustega. Samuti, et tulla toime tarbijate vajadustega, uute funktsioonide kasutuselevõtuga ja uute ilmnenud vigade kõrvaldamisega. Iga kord, kui koodis parandatakse leitud vead, tuleb veenduda, et tehtud

muudatused ei tekitanud uusi vigu. Seetõttu tuleb testida rakendust ka kõigi vanade funktsioonidega, mille testimine iga kord nullist, kui uus värskendus välja tuleb, on ilmselgelt range ja ebameeldiv protsess. Sellepärast on testijal väga mugav testida käsitsi uuendust ning käivitada testid, mis vaataksid üle vanade funktsioonide toimimise [22].

Tekib võimalus testida rakendust stsenaariumide puhul, mida on käsitsi väga raske reprodutseerida. Näiteks, et teada saada, kuidas veebirakendus käitub, kui seda kasutab korraga palju kasutajaid, vajab testimisprotsess käsitsi proovimiseks palju inimressursse ja täpset ajastust, mis jätab tohutult väikese võimaliku veamarginaali. Automatiseeritud testimine võimaldab korrata eelnevalt kirjeldatud juhtumit lihtsalt ja kiiresti saates süsteemi hulgaliselt päringuid samal ajal [22].

Samuti saab eelisena välja tuua kiire tagasiside mehhanismi ja töö protsessi muutumise järjepidevamaks ja jätkusuutlikumaks. Automatiseerimine kiirendab testitsükleid ja eemaldab korduvad monotoonsed testjuhtumid. Samuti võimaldab luua paremaid testjuhtumite stsenaariume, parandades ja kiirendades projekteerijate, arendajate ja klientide vahelist suhtlust. Tõrgete kõrvaldamine muutub kergemaks, seega tagab automatiseerimine lõpptoote parema kvaliteedi ja skaleeritavuse [23].

Pikas perspektiivis vähenevad tegevuskulud ja tõuseb tööjõu kasutamise efektiivsus. Peale selle tagatakse tootearendusse tehtud investeeringute parem tasuvus. Kuigi automatiseerimise testimisel võivad olla suured esialgsed kulud ja muud lisakulud, saavad ettevõtted parandada pikaajalist kasumlikkust läbi tööjõu vähendamise. Automatiseeritud testskriptid vajavad minimaalset sekkumist ja nõuavad harva testjuhtumi täitmist ja skriptivigade tõrkeotsingut. Seega muutub paremaks tööjõu kasutamine, sest seda hakatakse rakendama olulisemates äriprotsessides, mitte korduvates ülesannetest. Korduv käsitsi testimine tarkvaratoote turule toomiseks põhjustab toote viivitusi ja mõjutab kvaliteeti. Automatiseerimise testimine vähendab tootearenduse elutsükli kulusid, pakub võimalikult veavaba toodet ja lühendab toote turule toomise tähtaegu [23].

Automaattestimine kiirendab tarkvara testimist erinevates brauserites ja teeb võimalikuks testida tarkvara erinevatel operatsioonisüsteemidel ja masinatel. Kuna kliendid kasutavad laias valikus veebilehitsejaid, on käsitsi brauseriülene testimine keeruline väljakutse. Automatiseeritud brauserite vahelise testimise tööriistad, teostavad samu testjuhtumeid

mitu korda paralleelselt mitmes brauseris, kõrvaldades sellega kirjeldatud probleemi. Samuti pole testjuhtumite jaotamine mitme operatsioonisüsteemi, brauseri ja masinate vahel käsitsi testimise abil võimalik, kuna testijad saavad ühel platvormil või seadmes korraga teha ainult ühe testi. Automaatsed testimistööriistad jaotavad testi teostamise nii, et see töötab korraga mitmel seadmel või platvormil parandades seeläbi testimise tõhusust ja kiirust [23].

Tekib võimalus viia läbi keerulisi ja mitme sammuga testjuhtumite stsenaariume väga lihtsalt ja vähese ajaga. Keeruliste testjuhtumite puhul, võivad käsitsi testimisel tekkida vead väga kergelt, sest testijateks on inimesed. Mõned näited faktoritest, miks võivad tekkida vead on tähelepanu kõrvalejuhtimine ja väsimus. Automatiseeritud testimise abil saab koostada juhendi käskudest ning tarkvara täidab need just selles järjekorras, nagu peab [23].

3.2.2 Automaattestimise puudused

Kõigel on omad piirangud, alustades kavandatud ärimudelitest ja lõpetades valmiva tarkvaraga, mille tegemisel läbitakse tarkvaraarendus ja kvaliteedi tagamine. Kõige tähtsam on neid piiranguid mõista, et meeskond oleks sellest teadlik ja neil oleks võimalus nendega toime tulla [24].

Pole kahtlust, et automatiseeritud testimine on kiireim ja kulutõhusam kvaliteedikontrolli tegemise meetod, kuid see ei saa teha kõike. Automatiseeritud testimisel on käsitsi testimise ees märkimisväärseid puudusi, millest mõned on süsteemi sisse ehitatud ja neid tuleb tasakaalustada käsitsi testimisega. Osad on tingitud ebatäpsest eelprogrammeerimisest. Samuti hõlmavad piiranguid ka keerulise kujundusega dünaamilisi rakendusi. Lõppkokkuvõttes on ettevõtte eesmärgid suurimaks määravaks teguriks, kas ettevõte peaks kasutama automaatset või käsitsi testimist. Sõltuvalt sellest, mida rohkem vajatakse, võib kasutada automaatset testimist, käsitsi testimist või nende kombinatsiooni [24], [25].

Üks automatiseeritud testimise suurimaid miinuseid on automatiseerimise alguses vajalike tööriistadega seotud kulud ja aeg. Lisaks võib tööriista kasutamise õppimine, vajalike testide väljatöötamine ja käitamine olla aeganõudev protsess, mille läbiviimiseks on vaja kedagi, kes oskab automatiseerimistestide skripte kirjutada ning läbi selle vältida väikeste vigade juhuslikku sisestamist. Vaatamata sellele tuleb sageli esialgne

investeering tuluna väga kiiresti tagasi, kuna arendaja tootlikkus paraneb ja tulemused on usaldusväärsemad [25], [26].

Teiseks puuduseks on keerukus. Automaattestide arendamine võib võtta kauem aega kui käsitsi tehtud testid. Eriti juhul, kui need on halvasti planeeritud, ja neid võib olla arenduse elutsüklisse keerulisem integreerida. Olukorras, kui testid on halvasti kirjutatud või raskesti hallatavad, võib kvaliteedikontrolli efektiivsus langeda. See omakorda võib negatiivselt mõjutada rakenduse testimise pidevust ja testidega kaetavust tarkvara eluea jooksul [26].

Kolmandaks miinuseks on pidevate uuenduste vajadus ja olemasolevate testide haldamine vastavalt testitava tarkvara uuendustele. Iga rakenduse uuenduse jaoks tuleb kõik seotud olemasolevad testid üle vaadata ja muuta vastavalt uuendustele, või kirjutada uued. Olemasolevaid teste tuleb alati värskendada ja hoida ajakohasena, et nende tulemused oleksid usaldusväärsed [26].

Automatiseeritud testid võivad anda valepositiivseid ja valenegatiivseid tulemusi. Isegi kui tõelist probleemi pole, võivad automatiseeritud testid ebaõnnestuda. See võib juhtuda, kui testi skript sisaldab viga või test pole ei ole piisavalt hästi kavandatud, et hõlmata kõiki selle kasutusjuhtumeid. Samamoodi võivad testid anda ka valenegatiivseid tulemusi. Näiteks, kui nende eesmärk on kontrollida millegi olemasolu, mitte selle toimimist [26].

Viiendaks puuduseks on töökindla kui hästi hallatava testimiskeskkonna raske väljatöötamine. Juba ainuüksi põhjalikku ja hallatavat automaattestide komplekti, mis kataksid suurema osa rakendusest, pole kerge luua. Peale selle peab testide komplekt olema piisavalt usaldusväärne, et seda saaks korduvalt ja regulaarselt jooksutada ilma valepositiivseid või valenegatiivseid tulemusi saamata. Testiskriptid peavad olema piisavalt kohandatavad, et arvestada rakenduse võimalike muudatustega, näiteks rakenduse uuendamise käigus muudetakse sisestusväljade nimesiltide nimetused. Sellepärast nõuavad testid täpset kavandamist ja head hallatavust ning arendaja kõrget kvalifikatsiooni [26].

Veel üheks peamistest puudustest on see, et graafilise kasutajaliidese komponentide testimine pole võimalik. Kuigi automaatteste võidakse kasutada suurema osa rakenduse funktsionaalsuse kontrollimiseks, ei sobi need graafika ega helifailide testimiseks. Seda

seetõttu, et arvutiuuringud nõuavad tulemuse kinnitamiseks sageli tekstipõhiseid kirjeldusi, mille tegemine antud juhtudel on peaaegu võimatu [26].

3.2.3 Automaattestimise peamised kontseptsioonid

Automaattestimise peamised ja kõige rohkem tuntud kontseptsioonid on funktsionaalne testimine, ühiktestimine, integratsiooni testimine, *smoke* testimine, mittefunktsionaalne testimine, jõudluse testimine, regressioonitestimine, märksõnapõhine testimine ja andmepõhine testimine. Ühiktestimise ja integratsiooni testimise kontseptsioonide tutvustused on leitavad käesoleva töö manuaalse testimise peatükist 3.1.3, kuna kõiki testimise kontseptsioone saab kasutada nii manuaalsete kui ka automaattestide tegemisel [27], [19].

Funktsionaalne testimine ehk *functional testing* on protsess, mille kaudu kvaliteedikontrollijad määravad kindlaks, kas tarkvara toimib vastavalt eelnevalt kindlaksmääratud nõuetele. See kasutab musta kasti testimistehnikaid, mille puhul testijal puuduvad teadmised süsteemisisesest loogikast. Funktsionaalne testimine on seotud ainult selle kinnitamisega, kas süsteem töötab ettenähtud viisil. Funktsionaalne testimine keskendub eelkõige süsteemi põhifunktsioonide, selle põhilise kasutatavuse, kasutajatele ligipääsetavuse ja sarnaste asjade testimisele [27], [19], [28].

Smoke testimine on tarkvara testimistehnika, mis põhineb kontrollil, kas kõige olulisemad/peamised funktsioonid töötavad ootuspäraselt ja annab kvaliteedikontrolli tegijatele võimaluse oma testimisvoorudega edasi liikuda uuenduse spetsiifilisema testimise juurde. Seda tuntakse ka kui usalduse testimist. See on minimaalne testide komplekt, mis koosneb kindlatest testidest põhilise funktsionaalsuse testimiseks. Komplekti käivitatakse iga tarkvaraarenduse elutsükli jooksul, et veenduda, et uuendus ei riku peamisi rakenduse funktsioone ja, et hoida kokku kogu meeskonna aega ja ressursse edasisel uuenduse testimisel, kui see peaks olema vigane. *Smoke* testimist tehakse siis, kui väljatöötatud tarkvarafunktsioonid on tarkvara ehitusega integreeritud. Kui testimisel selles etapis ebaõnnestub, saadetakse rakendus üldiselt arendusmeeskonnale muutmiseks tagasi [27], [19], [29].

Mittefunktsionaalne testimine ehk *non-functional testing* on üks testimisviisidest, mida kasutatakse tarkvararakenduse jõudluse, kasutatavuse, töökindluse ja muude mittefunktsionaalsete omaduste hindamiseks ehk see ei keskendu sellele, mida toode teeb,

vaid sellele, kui hästi see seda teeb. Selle eesmärk on testida süsteemi valmisolekut mittefunktsionaalsete kriteeriumite alusel. Mittefunktsionaalsed kriteeriumid on need, mida funktsionaalne testimine kunagi ei arvesta. Selline testimine on oluline tarkvara töökindluse ja funktsionaalsuse kinnitamiseks. Selle tarkvara testimismeetodi aluseks on tarkvaranõuete spetsifikatsioon, mis võimaldab töötajatel kontrollida, kas süsteem vastab kasutaja nõuetele. See aitab vähendada toote mittefunktsionaalsete komponentidega seotud tootmisriski. Tavaliselt järgneb mittefunktsionaalne testimine funktsionaalsele testimisele, kuna on loogiline teada, et toode teeb seda, mida ta peaks tegema, enne kui uuritakse, kui hästi see seda teeb [19], [20], [30].

Jõudluse testimine ehk *performance testing* on tarkvara testimise tüüp, mille eesmärk on hinnata, kuidas teatud tarkvara eritingimustes toimib ja optimeerida tarkvara võimekust. Jõudlus viitab antud juhul mitmele muutujale: stabiilsus, mastaapsus, kiirus ja reageerimisvõime. Seda kõike testitakse mitmel tasemel liikluse ja koormuse korral. Jõudlustestimine tagab, et arendatav tarkvara vastab jõudlusnõuetele. Üks selline jõudlustestimise näide on kontrollida, kas rakendus suudab toime tulla tuhandete kasutajatega, kes logivad korraga sisse, või tuhandete kasutajatega, kes teevad rakenduses samal ajal samu või erinevaid toiminguid. See aitab tuvastada rakenduses esinevaid kitsaskohti. Rakenduse testimine tagab, et teie rakendus peab vastu, kui sellele pääseb korraga juurde palju kasutajaid [27], [20], [31].

Regressioonitestimine ehk *regression testing* on tarkvara testimine, mille eesmärgiks on tagada, et koodimuudatus ei ole olemasolevaid funktsioone negatiivselt mõjutanud. Teisisõnu võib öelda, et regressioonitestimise käigus käivitatakse juba olemasolevad testikomplektid uuesti, et tagada olemasolevate funktsioonide hea toimimine. Samuti tagab, et olemasolev toode töötab korralikult uute funktsioonidega enne nende väljalaskmist. Tuleb meeles pidada, et regressioonitestimine pole kordustestimine. Kordustestimist tehakse siis, kui rakenduses tuvastatud probleem lahendatakse ja lahendust on vaja uuesti testida, et kontrollida, kas probleem lahenes ja kõik toimib [27], [19], [32].

Märksõnapõhine testimine ehk *keyword-driven testing* on testimismeetod, mis kasutab testitava rakendusega seotud märksõnu sisaldavaid andmefaile. Antud meetod on väga kasulik inimeste jaoks, kellel pole tehnilist tausta, kuna see tuvastab kirja pandud märksõna tabelis juba valimis programmeeritud tegevustega. Need märksõnad

kirjeldavad konkreetseid samme vajalike toimingute sooritamiseks. Märksõnapõhine test koosneb kõrge ja madala tasemega märksõnadest, sealhulgas märksõnaargumentidest, mis on koostatud testjuhtumi toimingu kirjeldamiseks erinevatest märksõnadest. Seda nimetatakse ka tabelipõhiseks testimiseks või tegevussõnapõhiseks testimiseks. Märksõnapõhine testimine on lühike, paindlik, korduvkasutatav ja hõlpsasti hooldatav, mistõttu on see paljude ettevõtete jaoks populaarne valik. See ei nõua programmeerimisteadmisi, võimaldab funktsionaalsetel testijatel planeerida testimist juba enne rakenduse väljatöötamist [19], [27], [33].

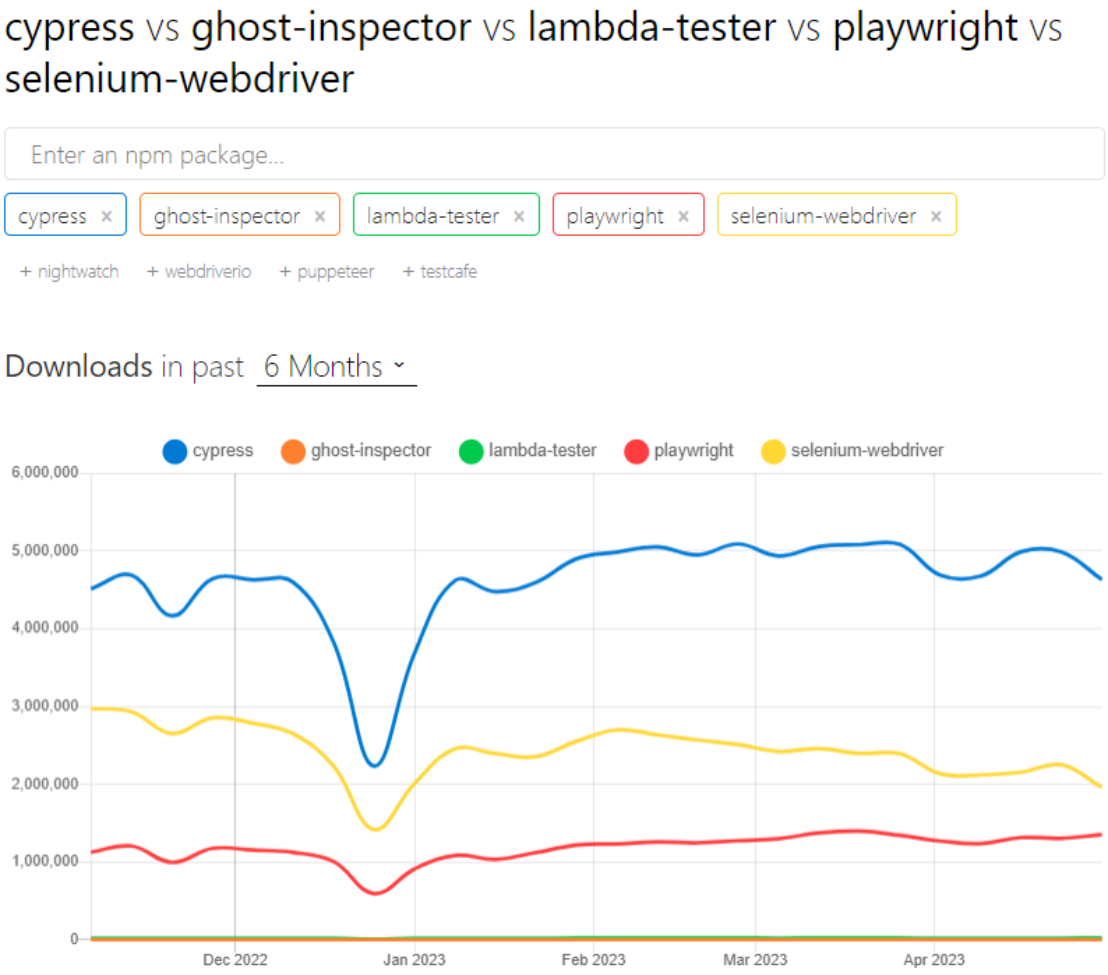
Andmepõhine testimine ehk *data-driven testing* on tarkvara testimismeetod, mille puhul testandmed salvestatakse tabeli-või arvutustabelivormingus. Andmepõhine testimine võimaldab testijatel sisestada ühe testskripti, mis suudab testida kõiki tabelis olevaid testandmeid ja oodata testiväljundit samas tabelis. Andmepõhine testimine aitab hoida andmeid testskriptidest lahus ja samu testskripte saab käivitada erinevate sisendandmete kombinatsioonide jaoks [27], [34].

Kogu kolmandast peatükist võib järeldada, et ettevõtte MRPeasy OÜ jaoks on kõige parem variant kasutada manuaalse ja automatiseeritud testimise kombinatsiooni. Kuna käesoleva töö probleemi üheks osaks oli pidev vana funktsionaalsuse testimise vajadus, peab autor automaattestimise kasutuselevõtmist väga oluliseks vaatamata selle puudustele. Isegi, kui automaattestimisel on omad piirangud ja selle esialgne paigaldamine võtab aega ning raha, ületavad selle positiivsed küljed kõik peatükis 3.2.2 üles loetletud puudused. Lõpptulemusena testimismeeskonna töö efektiivsus ja töö kvaliteet kasvavad. Samuti muutub ka testijate töö keskkond paremaks pinge vähenemise tõttu, kuna töötajatel kaob vajadus muretseda sellepärast, kas nad vaatasid vana funktsionaalsuse piisavalt hästi läbi või ei.

4 Valitud testimisraamistike vastavus kriteeriumitele

Käesoleva töö jaoks võttis autor oma raamistike võrdlusesse vastavalt TESTRIG 2023. aasta uuringule kaheksa enim nõutud veebirakendustele mõeldud testimisraamistikku. Nendeks osutusid Cypress, Playwright, Katalon Studio, Tosca, Ghost Inspector, Ranorex Studio, Lambda Test ja Selenium [35].

Jooniselt 1 on näha kaheksast kõigi raamistike, peale Katalon Studio, Tosca ja Ranorex Studio, kuna need pole *npm* paketiga alla laetavad, allalaadimiste arvu võrdlust viimase kuue kuu jooksul perioodil 2022. aasta novembrist kuni 2023. aasta aprillini kaasa arvatud [19], [36].



Joonis 1. Cypress, Playwright, Ghost Inspector, Lambda Test ja Selenium allalaadimiste võrdlus [37].

Raamistiku allalaadimiste arvu võib võrrelda populaarsusega ja kuna üheks paika pandud kriteeriumitest oli raamistiku populaarsus, otsustas autor selle põhjal uuringust Ghost Inspectori ja Lambda Testi välja jätta.

4.1 Selenium WebDriver

Selenium WebDriver on avatud lähtekoodiga tööriist, mis on üles ehitatud WebDriveri protokollile ja tagab paljude brauserite tuge erinevates programmeerimiskeeltes. Kuna see on eksisteerinud juba väga kaua, on sellel suur kasutajate kogukond, kes on valmis üksteist aitama [38].

Testide täitmine toimub lühidalt kokku võttes läbi kolme peamise etapi:

1. Testimiseks kirja pandud käsud tehakse URL-iks;
2. HTTP-serveri abil saadetakse URL-id brauseri draiveritele;
3. URL-id edastatakse päringuna tegelikele brauseritele ja kõik testide skriptides olevad käsud täidetakse [38].

Selenium WebDriveri eelised:

- Toetab integreerumist sidusarenduskeskkondadega;
- tasuta kasutamise võimalus;
- võimaldab testida alla laetud failide sisu;
- toetab *headed* ja *headless* brauserite režiime;
- toetab mitut programmeerimiskeelt: C#, JavaScript, Java, Ruby, Python ja PHP [38], [39].

Selenium WebDriveri puudused:

- Raamistikul on põhjalik dokumentatsioon, kuid paljud näited on vaid Java arenduskeeles;
- testjuhtumite loomine on aeganõudev [38], [39].

4.2 Cypress

Cypress on JavaScriptil põhinev testimisraamistik, mis on üles ehitatud uuele arhitektuurile ja töötab testitava rakendusega samas käitamistsüklis. See on

arendajasõbralik tööriist, mis kasutab ainulaadset dokumendi objektimudeli manipuleerimistehnikat ja töötab otse brauseris koos testitava rakendusega [40].

Cypressi eelised:

- Kõik Cypressi testide skriptid käivitatakse brauseri sees;
- tasuta kasutamise võimalus;
- toetab JavaScripti;
- toetab *headed* ja *headless* brauserite režiime;
- võimaldab testida alla laetud failide sisu;
- toetab integreerumist sidusarenduskeskkondadega;
- põhjalik dokumentatsioon [38].

Cypressi puudused:

- Ei saa kasutada kahe brauseri samaaegseks testimiseks;
- ei toeta mitme akna testimist ehk kui testi ajal mingi nupu vajutamisel avaneb uus aken, siis seda testida ei saa. Kõik toimub ühes aknas;
- ei toeta Safari brauserit [38].

4.3 Playwright

Playwright on avatud lähtekoodiga Node.js teek veebibrauserite automatiseerimisraamistik läbivtestide kirjutamiseks. Playwright töötab otse WebSocketiga, mis tähendab, et pärast testi käivitamist teisendatakse kood JSON-vormingusse ja saadetakse serverisse WebSocketi protokolli kasutades. Kui ühendus on loodud, saadetakse testis olevad käsud Playwrighti serverisse. Kliendi ja serveri ühendused jäävad aktiivseks, kuni üks või mõlemad pooled need katkestavad [41].

Playwrighti eelised:

- Toetab mitut programmeerimiskeelt: JavaScript, TypeScript, Python, Java ja C#;
- võimaldab testida alla laetud failide sisu;
- toetab *headed* ja *headless* brauserite režiime;
- toetab integreerumist sidusarenduskeskkondadega;
- Codegen funktsioon aitab salvestada kasutusjuhte ja genereerida neile kood keeltes, mida Playwright toetab [42].

Playwrighti puudused:

- Põhjalik, kuid segane dokumentatsioon. Otsitava teema leidmine võtab aega ja näiteid on vähe;
- kogukond on raamistiku uudsuse tõttu veel liiga väike;
- toetab ainult mobiilseadmete emulaatorite testimist ehk ei toeta päris seadmete testimist [43].

4.4 Katalon Studio

Katlon Studio on testimise automatiseerimise raamistik, mis toetab veebi-, mobiili- ja rakendusliidese testimist. Katlon Studio toetab visuaalset testiredaktorit, mis võimaldab mittetehnilistel kasutajatel teste luua ja käivitada. Katlon Studio toetab ka märksõnapõhist testimist, võimaldades kasutajatel kirjutada teste loomuliku keele süntaksi abil. Lisaks pakub Katlon Studio automaatset testiaruannet, mis teeb testitulemuste jälgimise ja probleemide tuvastamise lihtsaks [44].

Katalon Studio eelised:

- Võimaldab testida alla laetud failide sisu;
- toetab *headed* ja *headless* brauserite režiime;
- hea dokumentatsioon;
- toetab integreerumist sidusarenduskeskkondadega;
- on olemas tasuta versioon [44], [45].

Katalon Studio puudused:

- Arenduskeeleks on Groovy, mis on üles ehitatud Java programmeerimiskeelele;
- väike kasutajate kogukond;
- pole avatud lähtekoodiga;
- ei toeta testide paralleelset jooksutamist [44], [45].

4.5 Tosca

Tosca on testimise automatiseerimise tööriist, mis toetab veebirakenduste täielikku testimist. Tosca pakub testimiseks välja mudelipõhise lähenemisviisi, mis võimaldab kasutajatel luua testjuhtumeid ilma kodeerimisoskusteta. Tosca pakub ka

automatiseeritud riskianalüüsi, et tuvastada rakenduse kõige kriitilisemad valdkonnad [46].

Tosca eelised:

- Toetab paralleelset testide käivitamist, võimaldades kasutajatel testida korraga mitmes keskkonnas;
- võimaldab testida alla laetud failide sisu;
- toetab integreerumist sidusarenduskeskkondadega;
- genereerib koodi testimiseks ise vastavalt kasutaja tegevustele;
- abivalmis kogukond;
- toetab *headed* ja *headless* brauserite režiime;
- põhjalik dokumentatsioon [46].

Tosca puudused:

- Toetab Visual Basic, Java ja C# arenduskeeli;
- Töötab ainult Windows operatsioonisüsteemiga;
- Pole tasuta versiooni [45].

4.6 Ranorex Studio

Ranorex Studio on testimise automatiseerimise tööriist, mis toetab veebirakenduste täielikku testimist. Ranorex Studio pakub objektide hoidlat, mis võimaldab kasutajatel objekte erinevates testides uuesti kasutada. Ranorex Studio pakub ka automaatset ootamist, oodates, kuni rakendus lõpetab lehe laadimise enne järgmise käsu täitmist. Lisaks toetab Ranorex Studio integreerimist teiste testimistööriistadega, sealhulgas Jenkinsi ja Jiraga [47].

Ranorex Studio eelised:

- Võimaldab testida alla laetud failide sisu;
- toetab integreerumist sidusarenduskeskkondadega;
- võimaldab testida veebirakendusi, töölauarakendusi ja mobiilirakendusi;
- ei vaja täiendavaid programmeerimise teadmisi, kuna võib kasutada märksõnapõhist testimist;
- hea dokumentatsioon;

- genereerib automaatselt aruandeid [47].

Ranorex Studio puudused:

- Tasuline litsents;
- väike kogukond;
- vajab rakendusliidese testimiseks lisa integratsioone;
- toetab C# ja VB.NET programmeerimiskeeli [48].

4.7 Raamistike võrdlus kriteeriumite põhjal

Tabelis 1 on välja toodud koondtabel raamistike vastavuse kohta esimesele poolele kriteeriumitest. Kui raamistik vastab kriteeriumile, on kasti märgitud „+“, kui ei vasta, siis „-“, ning, kui raamistik vastab osaliselt kriteeriumile on kirja pandud, mis oli puudu.

Tabel 1. Raamistike võrdluse tabel kriteeriumite põhjal.

	Selenium	**Cypress**	**Katalon Studio**	**Ranorex Studio**	**Tosca**	**Playwright**
Integreeritav sidusarenduskeskkonnaga	+	+	+	+	+	+
Toetab *headed* ja *headless* brauserite režiime	+	+	+	-	+	+
Toetab alla laetud failide testimist, näiteks PDF	+	+	+	+	+	+
Toetab JavaScripti	+	+	-	-	-	+
Täpne ja põhjalik dokumentatsioon	Osad näited ainult Java keeles.	+	+	+	+	+
Laialdaselt kasutatav ehk populaarne	+	+	+	+	+	+
Tasuta kasutamise võimalus	+	+	+	-	-	+

Esines mitu kriteeriumit, millele vastasid kõik raamistikud. Nende hulgas olid integreerimine sidusarenduskeskkonnaga, alla laetud failide testimine ja populaarsus. Dokumentatsiooni põhjalikkuse ja täpsuse kriteeriumile vastasid muidu samuti kõik valikus olevad raamistikud, kuid Seleniumi dokumentatsiooni raamistiku ja testide struktuuri puudutavad näited ja raskemad testinäited olid ainult Java programmeerimiskeeles. Testimist brauserite režiimides *headed* ja *headless* toetavad kõik peale Ranorex Studio raamistiku. See kriteerium on väga oluline tulevikus regressioonitestide käivitamiseks, kuna testid läbitakse kiiremini ning tegevus toimub taustal. Soovitud programmeerimiskeelt toetavad vaid Selenium, Cypress ja Playwright. Tasuta kasutamise võimalus on kõigil raamistikel peale Ranorex Studio ja Tosca. Nendes raamistikes on erineva maksumusega paketid. Neil on tasuta prooviperiood, kuid pikaajalist tasuta kasutamise võimalust pole.

Autor jõudis järeldusele, et vastavalt tabelis 1 toodud võrdlusele, sobivad MRPeasy OÜ ettevõttesse kasutusele võtmiseks kõige rohkem kaks raamistikku: Cypress ja Playwright. Kriteeriumite kohaselt oleks Selenium ka sobinud, kuid kuna neil pole dokumentatsioonis mõnedes peatükkides näiteid JavaScripti keele jaoks, otsustas autor selle välja jätta.

5 Cypress ja Playwright raamistike võrdlus

Vastavalt autori eelnevale uurimusele neljandas peatükis jäi võrdlusesse kaks raamistikku: Cypress ja Playwright.

Käesolevas peatükis üritab autor leida vastust, milline kahest raamistikust vastab paremini kriteeriumitele, mida ei saanud hinnata eelmises peatükis. Nendeks kriteeriumiteks on kerge, kiire paigaldamine ning kasutuselevõtmine ja võimalikult arusaadav testide kirjutamise süntaks ettevõtte arendajatele. Selleks, et testida paigaldamist, teeb autor Cypress ja Playwright raamistike paigaldamise katse raamistike dokumentatsioonides olevate juhendite järgi ning võrdleb, kumba on kergem ja kiirem projekti paigaldada. Peale selle, et testida, kumba on kergem ja kiirem kasutusele võtta, kirjutab autor ühe testi mõlemale raamistikule. Samuti, et saada vastust, millise raamistiku süntaks on ettevõtte testijatele arusaadavam, näitab autor oma eelnevalt tehtud teste küsitluses MRPeasy OÜ kvaliteedikontrolli tegijatele, et saada nende tagasiside koodi ja kasutajaliidese kohta. Paigaldamise ja küsitluse tulemusena tehakse järeldused, milline raamistikest sobib kõige enam MRPeasy OÜ ettevõttesse kasutusele võtmiseks.

5.1 Raamistike paigaldamise võrdlus

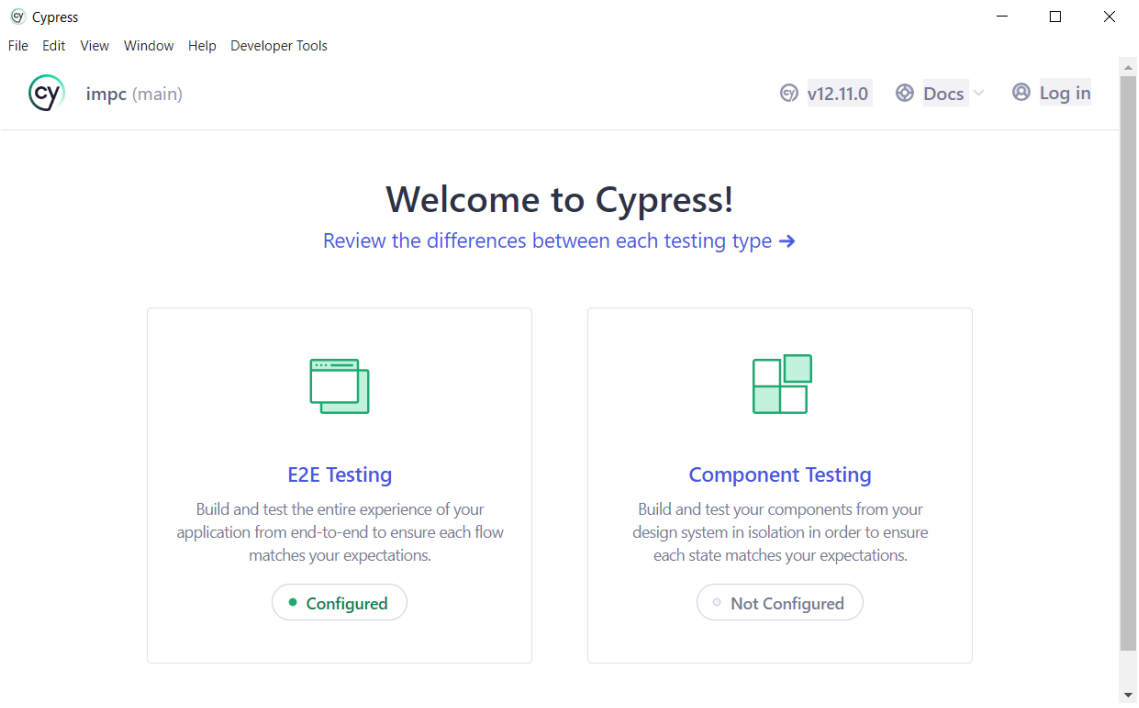
Üheks raamistike valimise kriteeriumiks oli kerge, kiire paigaldamine ja kasutuselevõtmine. Selleks, et mõõta, kui palju aega võtab testimisraamistike Cypress ja Playwright projekti paigaldamine ja avamine ehk kasutama asumine, tegi autor GitLab keskkonda kaks eraldiseisvat projekti. Projekti, millesse paigaldati Cypress raamistikku nimetati „ImpC“ ja projekti, kuhu paigutati Playwright „ImpP“. Seejärel klooniti mõlemad projektid autori arvutisse ning avati Visual Studio Code programmi abil. Visual Studio Code on programm, mida kasutatakse koodi kirjutamiseks ja redigeerimiseks. Aja mõõtmiseks kasutati iPhone telefoni sisseehitatud stopperit. Stopper pandi käima, kui autor vajutas Google otsingus klahvile „Enter“, peale juhendit otsiva teksti sisestamist. Stopper pandi kinni kohe peale seda, kui avanes raamistiku visuaalne paneel. Tuleb ära mainida, et autor polnud varem kummagi testimisraamistiku installeerimise juhendit näinud.

Tegevused Cypress raamistiku installeerimiseks enne stopperi käivitamist:

1. Google otsingusse sisestati „Cypress *installation documentation*“.

Tegevused Cypress raamistiku installeerimiseks peale stopperi käivitamist:

1. Otsiti vajalik link;
2. Avati vajalik link;
3. Autor luges installeerimise juhendi läbi;
4. Avati juba lahtiolev projekt „ImpC“;
5. Avati uus terminal;
6. Sisestati esimene käsk „npm install cypress --save-dev“, mis installeeris raamistiku projekti;
7. Sisestati teine käsk „npx cypress open“, et avada raamistiku testide visuaalne liides, mida on näha joonisel 2.



Joonis 2. Cypress raamistiku visuaalne liides.

Cypress raamistiku installeerimine ja liidese avamine võttis kokku 2 minutit ja 5 sekundit. Google otsingumootoris oli sisse antud otsinguga vajalik link kohe esimesel kohal. Samuti oli installeerimise juhend lühike ja täpne ning jaotatud erinevate paketihaldurite kaupa, nagu *npm* ja *yarn*. Esimene käsk oli vajalik, et teha informatsiooni faili versioonide

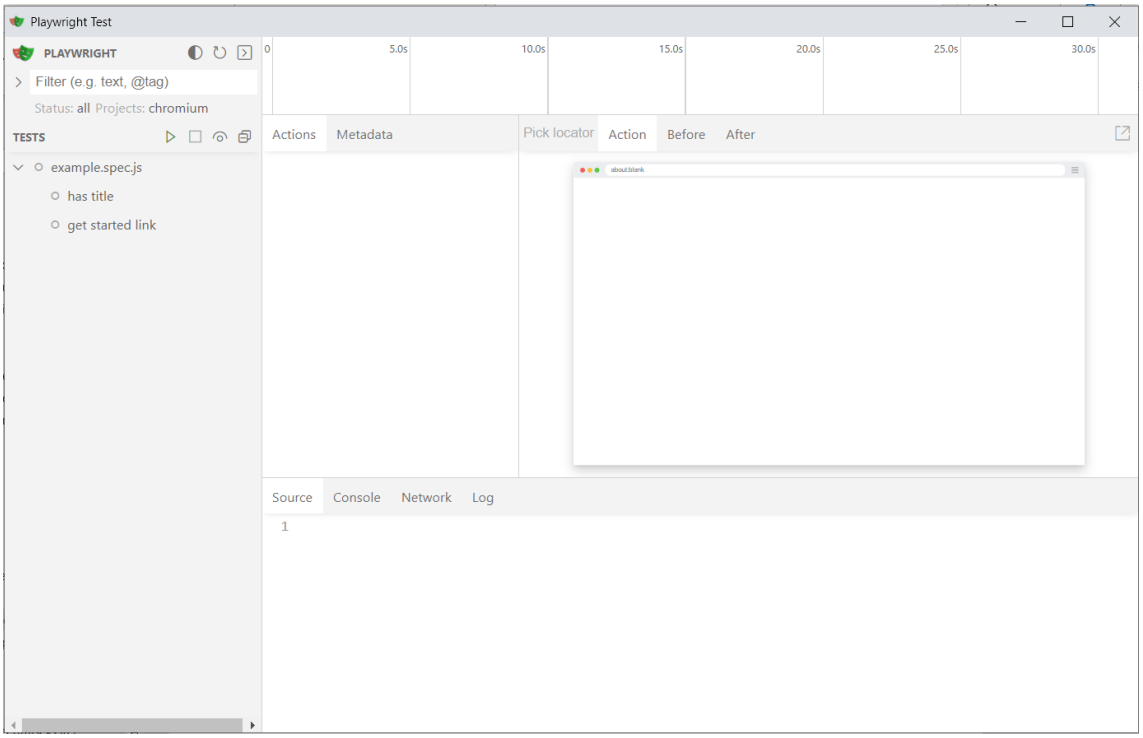
ja muu infoga „package.json“, millele tuginetakse raamistiku installeerimisel, ja et installeerida raamistik projekti. Teine käsk käivitas Cypressi visuaalse liidese [49].

Tegevused Playwright raamistiku installeerimiseks enne stopperi käivitamist:

1. Google otsingusse sisestati „Playwright *installation documentation*“.

Tegevused Playwright raamistiku installeerimiseks peale stopperi käivitamist:

1. Otsiti vajalik link;
2. Avati vajalik link;
3. Autor luges installeerimise juhendi läbi;
4. Avati juba lahtiolev projekt „ImpP“;
5. Avati uus terminal;
6. Sisestati käsk „npm init playwright@latest“, „package.json“ faili loomiseks ja raamistiku installimiseks;
7. Sisestati käsk „npx cypress open“, et avada raamistiku testide visuaalne liides, mida on näha joonisel 3.



Joonis 3. Playwright raamistiku visuaalne liides.

Playwright raamistiku installeerimine ja liidese avamine kulus kokku 2 minutit ja 10 sekundit. Google otsingumootoris oli sisse antud otsinguga vajalik link, nagu ka teise

raamistiku puhul, kohe esimesel kohal. Samuti oli installeerimise juhend väga lühike ja täpne. Raamistiku paigaldamiseks oli vajalik sisestada vaid üks käsk, mis tegi ise faili vaikesätetega ja installeeris seejärel raamistiku projekti. Teine käsk käivitas visuaalse liidese. Eriti mugav oli see, et käivitamise käsk oli leitav terminalist kohe peale raamistiku installeerimist [50].

5.2 Läbivtestide koodinäidete loomine ja nende võrdlus

Selleks, et võrrelda põhjalikumalt Cypress ja Playwright koodi kirjutamise mugavust ja dokumentatsiooni põhjalikkust ning kasutama hakkamise kiirust, kirjutab autor sama ülesandega ja ülesehitusega testi kummagi testimisraamistiku jaoks. Dokumentatsiooni põhjalikkust võrreldakse selle järgi, millises dokumentatsioonis on otsitud elemendi kasutamine paremini lahti seletatud ning kas otsitud informatsioon on dokumentatsioonis olemas. Testid tehakse ja proovitakse läbi projektides, kuhu paigaldati raamistikud peatükis 5.1. Kuna ettevõtte rakenduses viib üks nupuvajutus teiseni ja tekib pidev vajadus otsida lehekülgedel olevaid elemente ning sisestada tekstiväljadesse väärtusi, otsustas autor teha ühe pikema testi. Test peaks sisaldama vajalikule lehele minemist, teksti sisestamist ning seejärel kuhugi vajutades teiste lehekülgede läbimist ning lõpptulemuse kontrollimist. Et mitte läbida sisselogimise etappe, otsustati testid kirjutada Google otsingumootori jaoks, kuna see sisaldaks kõiki vajalikke etappe.

Test ise koosneb sellistest ülesannetest:

1. Test läheb „google.ee“ lehele;
2. Leiab üles otsingumootori otsinguvälja;
3. Kirjutab otsinguvälja „[Raamistiku nimi] *installation documentation*“;
4. Vajutab otsimist alustavale nupule „Google otsing“;
5. Leiab saadud lehelt vajaliku pealkirjaga lingi;
6. Läheb leitud lingi kaudu raamistiku dokumentatsiooni lehele;
7. Veendub selles, et ta on just selle raamistiku lehel.

5.2.1 Cypress testi loomine

Esimese asjana tuli leida, kuidas testi üldse kirjutatakse. Selle osani dokumentatsioonis jõuti väga kiiresti. Tuli liikuda järgmisele leheküljele raamistiku paigaldamise juhendilt.

Oma testi põhjaks võttis autor näite lehelt „Writing Your First E2E Test“, mida on näha jooniselt 4.

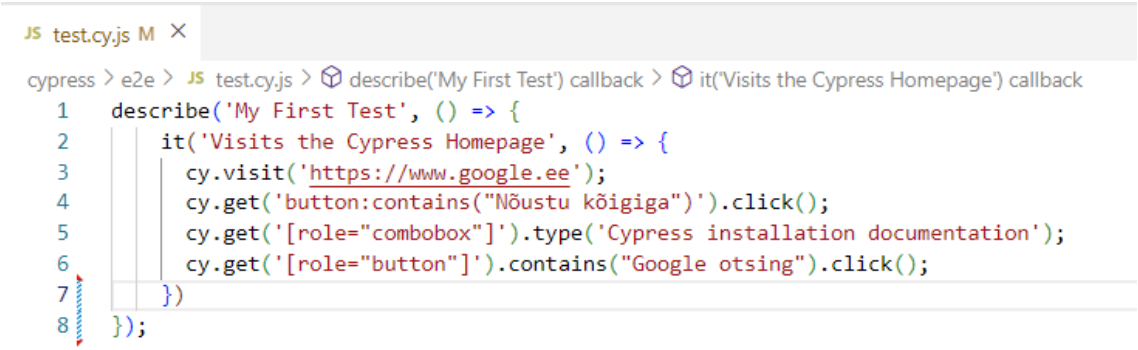
```
describe('spec.cy.js', () => {
  it('should visit', () => {
    cy.visit('/')
  })
})
```

Joonis 4. Cypress testi põhjana kasutatud näide [51].

Järgmise sammuna tuli otsida, kuidas leida lehelt soovitud elemente. Selleks kasutas autor dokumentatsiooni otsingut sisestades sinna väljendi *get by text*. Otsing viis autori kohe soovitud peatükini „How to Get DOM Elements“, kus oli palju erinevaid viise ja näiteid, kuidas erinevaid elemente lehelt kätte saab.

Peale elementide otsimist oli vaja leida, kuidas teha nii, et programm vajutaks nupule ja sisestaks välja soovitud teksti. Selleks sisestas autor otsingusse väljendid *Click* ja *Type*, mis viisid samuti kohe soovitud peatükkideni „click“ ja „type“.

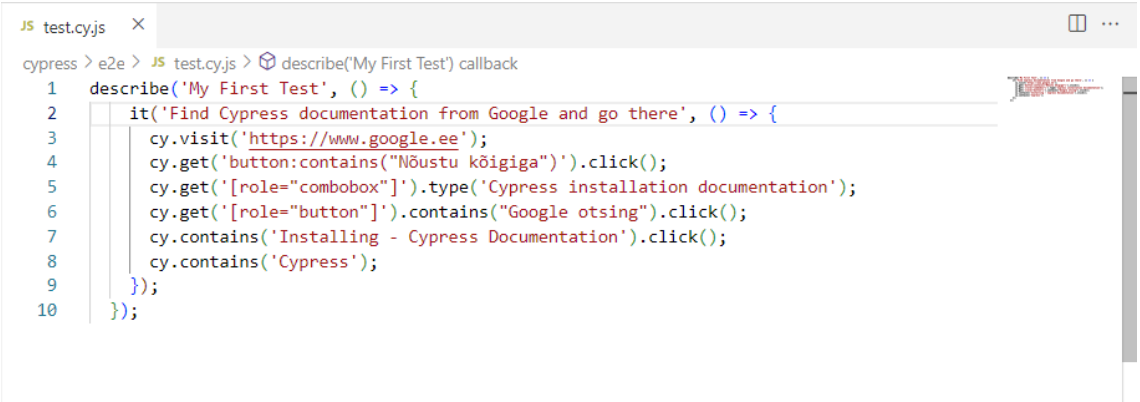
Jooniselt 5 on näha, milline nägi välja test, mis peale otsingu tegemist.



Joonis 5. Cypress test peale otsingu sooritamist.

Järgmise sammuna tuli leida, kuidas teha nii, et programm leiaks autorile vajaliku lingi pealkirjaga „Installing - Cypress Documentation“, veenduks, et see on just selle pealkirjaga ning seejärel vajutaks sellele. Selleks sisestas autor dokumentatsiooni otsingusse *Assertions*, mis viis kohe lehele „Assertions“, kus autor leidis endale vajaliku käsu.

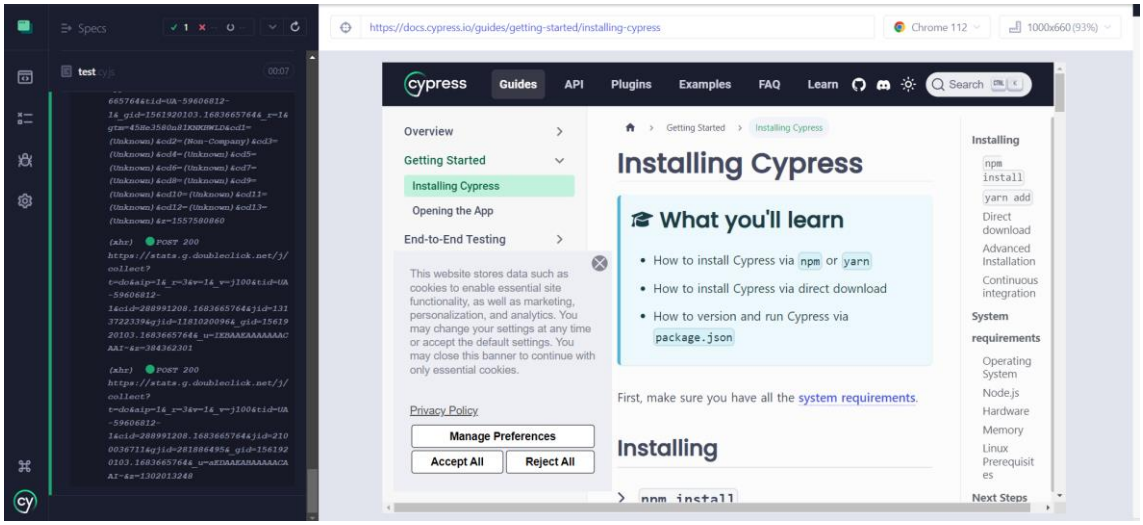
Jooniselt 6 on näha Cypress raamistiku lõplikku testi.



Joonis 6. Autori kirjutatud Cypress test.

Test õnnestus ning läbis edukalt soovitud tee. Autori Cypressi raamistiku testi saab näha TalTechi GitLabi koodihoidla lingilt https://gitlab.cs.ttu.ee/jorudu/impc.

Joonisel 7 on näha Cypress raamistiku kasutajaliidest ja õnnestunud testi.



Joonis 7. Cypress raamistiku kasutajaliides ja õnnestunud test.

5.2.2 Playwright testi loomine

Nagu ka Cypress raamistiku puhul, tuli ka siin esimese asjana leida, kuidas testi vormistus ja struktuur välja näevad. Samuti, kuidas jõuda soovitud leheküljele ning leida sealt vajalikud väljad. Suurema osa kogu vajalikust infost, kuidas midagi teha, leidis autor järgmiselt leheküljelt „Writing tests“, mis järgnes paigaldamise juhendile. Autor sai vastuse peaaegu kõigile oma küsimustele esimesest kahest testinäitest. Seal polnud vaid näidatud, kuidas tekstivälja kirjutamise käsk välja näeb.

Jooniselt 8 on näha autori testi põhjaks kasutatud teste.

tests/example.spec.ts

```
import { test, expect } from '@playwright/test';

test('has title', async ({ page }) => {
  await page.goto('https://playwright.dev/');

  // Expect a title "to contain" a substring.
  await expect(page).toHaveTitle(/Playwright/);
});

test('get started link', async ({ page }) => {
  await page.goto('https://playwright.dev/');

  // Click the get started link.
  await page.getByRole('link', { name: 'Get started' }).click();

  // Expects the URL to contain intro.
  await expect(page).toHaveURL(/.*intro/);
});
```

Joonis 8. Autori Playwright testi põhjana kasutatud testide näited [52].

Autor otsustas katsetada Playwright dokumentatsioonisisest otsingut ning otsida sealt tekstivälja kirjutamise käsku. Selleks kirjutati otsingusse otsingusõna *type*. Nii Playwright kui ka Cypress otsingud on väga sarnased, kuid kahjuks kohe õige tulemuseni see kord ei jõutud. Esimese asjana viis otsing autori „TypeScript“ lehele. Samuti ei viinud õige tulemuseni ka otsing *Type text*. Õige tulemuseni jõuti lõpuks otsinguga *Type text characters*, mis viis lehele „Writing tests“ alapeatükini „Basic actions“. Näidatud käskude tabelist leidis autor soovitud käsu ja jõudis sellele vajutades soovitud lehe „Locator“ alamosani „type“.

Joonisel 9 on näha autori kirjutatud Playwright raamistiku test.

test.spec.js

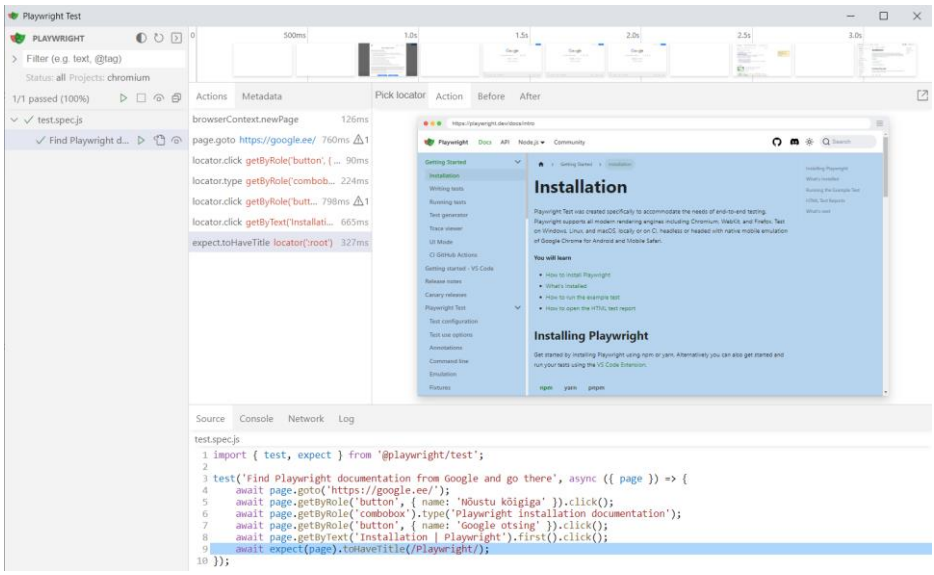
tests > test.spec.js > ...

```
1  import { test, expect } from '@playwright/test';
2
3  test('Find Playwright documentation from Google and go there', async ({ page }) => {
4      await page.goto('https://google.ee/');
5      await page.getByRole('button', { name: 'Nõustu kõigiga' }).click();
6      await page.getByRole('combobox').type('Playwright installation documentation');
7      await page.getByRole('button', { name: 'Google otsing' }).click();
8      await page.getByText('Installation | Playwright').first().click();
9      await expect(page).toHaveTitle(/Playwright/);
10 });
```

Joonis 9. Autori kirjutatud Playwright test.

Test läbis edukalt kõik soovitud sammud. Autori Playwright raamistiku testi saab näha TalTechi GitLabi koodihoidlas lingilt https://gitlab.cs.ttu.ee/jorudu/impp.

Joonisel 10 on näha edukalt läbitud testi ja Playwright raamistiku kasutajaliidest.



Joonis 10. Playwright raamistiku kasutajaliides ja edukalt läbitud autori test.

5.3 Järeldused

Peatükid 5.1 ja 5.2 annavad ülevaate, kui keeruline on Cypress ja Playwright raamistikke paigaldada, kasutusele võtta ja luua mitme sammuga teste. Mõlema raamistiku laadimine projekti ja kasutusele võtmine on äärmiselt kerge ja nende installeerimine projektidesse võtab väga vähe aega.

Testide loomisel olid mõlemal raamistikul omad mugavused ja ebamugavused. Cypress raamistikul oli autori vajadustele paremini vastav dokumentatsioon, kuna nende otsing toimis autori märksõnadega paremini. Vaatamata sellele, on mõlemal väga hea ja põhjalik dokumentatsioon. Playwright raamistikul olid väga mugavad valmis tehtud raamistiku kindla struktuuriga käsud. Paraku on neid käske väga vähestele atribuutidele ja olemasolevad keskenduvad rohkem sellistele atribuutidele, mida ei pruugi elementide kirjelduses olla, näiteks tekst või *placeholder*. Rohkem kasutusel olevatele atribuutidele, nagu identifikaator ja klass, selliseid lühendatud käske pole. Cypress raamistiku kood oli arusaadav, kuid testi kirjutamisel tekkis väike probleem elemendi nähtavusega raamistikule, kuna teine element kattis selle ära. Seetõttu tuli muuta elemendile lähenemist, mida Playwright raamistikus ei juhtunud.

Raamistikud olid nii paigaldamise kui ka koodi kirjutamise vaatepunktist üsna võrdsed. Seetõttu otsustas autor teha küsitluse MRPeasy ettevõtte testijate tiimi juhiga Olga

Januštšenok ja vanemtestijaga. Küsitluse eesmärgiks oli välja selgitada, millist raamistikku soovivad ettevõtte töötajad ise rohkem kasutada. Autor rääkis neile suuliselt peatükkides 4.2 ja 4.3 leitud eelistest ja puudustest. Peale selle koostas autor Google Forms küsitluse, kuhu pani peatükkides 5.2.1 ja 5.2.2 kirjutatud testid ja küsimused nende loetavuse kohta (vt Lisa 2). Samuti lisas sinna küsimused raamistike graafiliste kasutajaliideste kohta, milline kasutajaliides tundub neile mugavam. Kuna piltide lisamisest poleks kasutajaliidese puhul piisanud, tegi aautor neile kasutajaliidese ja raamistiku demo. Mõlemad ettevõtte töötajad pooldasid rohkem Cypress raamistikku. Nende jaoks oli mugavam, et Cypress raamistikus salvestatakse iga testi tulemus eraldi videona, mille tulemusena saab kohe vajaliku testi läbimise lahti võtta. Samuti oli ka Cypress raamistiku kasutajaliides nende jaoks intuitiivsem ja arusaadavam. Playwright raamistiku kasutajaliidesel ei meeldinud päriselus testi jooksutava akna suurus. Akna maksimaalne suurus oli liiga väike. Samuti liiga suur osa kasutajaliidese ruumi oli kaetud osadega, mis pole testijate jaoks töö käigus eriti olulised ning neid polnud võimalus ka ära peita, näiteks ülemine testide ajajoon ja testide loetelu. Kuna Cypress testi koodis oli vähem korduvaid osasid, loeti see ka loetavamaks ja arusaadavamaks (vt Lisa 3).

Kuna autori võrdluse tulemusena olid mõlemad nii Cypress kui ka Playwright sobilikud, sai otsustavaks töötajate küsitlemine. Ettevõttele MRPeasy kõige paremini sobivaks testimisraamistikuks automaattestide kirjutamiseks valiti Cypress testimisraamistik.

5.4 Cypress ja Playwright raamistike analüüs

Tabelis 2 on välja toodud koondtabel raamistike Cypress ja Playwright kõige alguses paika pandud kriteeriumite vastavuse kohta. Kui raamistik vastab kriteeriumile, on kasti märgitud „+“, kui ei vasta, siis „-“, ning, kui raamistik vastab osaliselt kriteeriumile on kirja pandud, mis oli puudu.

Tabel 2. Cypress ja Playwright raamistike võrdlus kõigi olemasolevate nõuete põhjal.

	Cypress	**Playwright**
Integreeritav sidusarenduskeskkonnaga	+	+
Toetab *headed* ja *headless* brauserite režiime	+	+

	Cypress	**Playwright**
Toetab alla laetud failide testimist, näiteks PDF	+	+
Toetab JavaScripti	+	+
Täpne ja põhjalik dokumentatsioon	+	+
Laialdaselt kasutatav ehk populaarne	+	+
Tasuta kasutamise võimalus	+	+
Raamistiku graafiline kasutajaliides peab olema töötajatele mugav	+	-
Võimalikult arusaadav testide kirjutamise süntaks ettevõtte testijatele	+	Arusaadav, kuid koodis esineb igal real kordusi, mis teevad kirjutamise ebamugavamaks
Kiire paigaldamisega	+	+
Kerge kasutuselevõtmisega	+	+

Kriteeriumite võrdlemiseks tabelis 2 on võetud raamistikud Cypress ning Playwright, sest just need raamistikud vastasid peatükis 4.7 tehtud võrdluse kriteeriumitele.

Nii Cypress kui ka Playwright raamistikud vastavad kriteeriumite esimesele poolele:

- Võimalus integreerida sidusarenduskeskkonnaga;
- peab toetama nii *headed* kui ka *headless* brauserite režiime;
- peab sisaldama võimalust testida alla laetud faile, nagu PDF;
- soovituslikult võiks toetada JavaScripti, kuna kvaliteedikontrolli tegijatel on sellega eelnev kogemus;
- tasuta kasutamise võimalus;
- peab olema täpse ja põhjaliku dokumentatsiooniga;
- peab olema laialdaselt kasutatav ehk populaarne.

Kriteeriumite teise poole vastavuse kontrollimiseks kasutas autor võtmemõõdikutena raamistike paigaldamist, testide kirjutamist ja ettevõtte testijate arvamust.

Nende tulemusena Cypress raamistiku paigaldamine projekti võttis 2 minutit ja 5 sekundit ning Playwright raamistiku paigaldamine 2 minutit ja 10 sekundit. Paigaldamise vahe oli 5 sekundit, mis on nii vähe, et autor otsustas, et mõlemad raamistikud vastavad kriteeriumile, et raamistiku peab olema võimalik kiiresti paigaldada.

Seda, kas raamistik vastab kriteeriumile kerge kasutuselevõtmise kohta, otsustati selle järgi, mitu sammu oli vaja teha peale paigaldamist, et esimest testi kirjutama asuda. Teste võis hakata kirjutama kohe peale paigaldamist ehk ühtegi sammu tegelikult polnudki. Sellest võib järeldada, et mõlemad raamistikud vastavad etteantud kriteeriumile kerge kasutuselevõtmise kohta.

Selleks, et otsustada, milline raamistik vastab kahele katmata jäänud kriteeriumile, võimalikult arusaadav testide kirjutamise süntaks ettevõtte testijatele ja raamistiku graafiline kasutajaliidese mugavus, otsustas autor viia läbi küsitluse ettevõtte testijatega.

Küsitluse tulemusena selgus, et Cypess raamistiku disain ja mugavus meeldivad töötajatele rohkem, kui Playwright raamistiku omad. Samale arvamusele jõuti ka teise kriteeriumi koha pealt. Cypress raamistiku süntaks oli töötajatele loetavam ja kergem aru saada.

6 Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli jõuda järeldusele, milline testimisraamistik automaattestide kirjutamiseks ettevõttes MRPeasy OÜ kasutusele võtta.

Tööd alustati ettevõtte kirjeldusest, et mõista paremini ettevõtte vajadusi ning töötsüklit. Sellele järgnes manuaalse ja automaattestimise kirjeldamine, et anda ülevaade, miks üht või teist kasutatakse. Samuti anti ülevaade ka nende peamistest kontseptsioonidest. Töö teises pooles keskenduti TESTRIG uurimuse kaheksal enim nõutud 2023. aastal testimisraamistike võrdlusele. Selleks, et neid võrrelda koostati koostöös ettevõttega kriteeriumid, mille põhjal raamistikke võrrelda. Autor jagas kriteeriumid kaheks: need, mida saab infootsingu tulemusena leida, ja need, mille kontrollimiseks on vaja midagi ise teha, kuna seda on parem ise läbi teha.

Peale raamistike võrdlust nende kriteeriumite põhjal, mida oli võimalik leida infootsinguga, jäi võrdlusesse kaheksast raamistikust kaks. Nendeks olid Cypress ja Playwright. Peale seda asus autor kriteeriumite teise poole juurde. Selleks, et kontrollida neid kriteeriume tuli autoril proovida mõlemad raamistikud projektidesse paigutada, et saada teada, kui kaua võtab aega raamistike paigaldamine. Samuti tegi autor raamistikele ka kaks samasuguste sammudega testi, et testida raamistike kasutuselevõtmist ning neid teste hiljem oma küsitluses ettevõtte testijatega kasutada. Küsitlus tehti eesmärgiga saada sisendit ettevõtte töötajatelt, kuna kaks kriteeriumit olid seotud nende eelistuste-ja mugavusega.

Käesolev töö andis ettevõtte MRPeasy OÜ IT-osakonnale olulist teavet ja aitas välja valida läbivtestide jaoks sobiva töövahendi. Analüüsi ja võrdluse põhjal selgus, et kõige sobivam raamistik ettevõttele läbivtestide kirjutamiseks on Cypress. Ettevõtte võttis käesoleva töö tulemused vastu ja Cypress raamistikku hakati paigaldama ettevõttesse 2022. aasta detsembris. Nende viie kuu jooksul on MRPeasy OÜ testijate tiimi juhi Olga Januštšenok sõnul raamistik vastanud kõigile töötajate ootustele ja vajadustele. Autor plaanib ka edaspidi teha koostööd selle ettevõttega ning hoida ennast kursis raamistiku toimimisest ning töötajate rahulolust.

Kasutatud kirjandus

[1] MRPeay OÜ, „Pilvepõhine tootmistarkvara väikestele ja keskmistele tootjatele (10 - 200 töötajat),“ [Võrgumaterjal]. Available: https://www.mrpeasy.com/et/. [Kasutatud 15 aprill 2023].

[2] AS CREDITINFO EESTI, „MRPEASY OÜ,“ 31 märts 2023. [Võrgumaterjal]. Available: https://www.e-krediidiinfo.ee/12617332-/. [Kasutatud 17 aprill 2023].

[3] MRPeasy OÜ, „Hinnakiri,“ [Võrgumaterjal]. Available: https://www.mrpeasy.com/et/hinnakiri/. [Kasutatud 15 aprill 2023].

[4] Atlassian, „Kanban,“ 2023. [Võrgumaterjal]. Available: https://www.atlassian.com/agile/kanban. [Kasutatud 18 aprill 2023].

[5] S. Hoogerwerf, „4 Important Kanban Principles to Remember When Leading an Agile Team,“ 6 aprill 2022. [Võrgumaterjal]. Available: https://projectmanagementacademy.net/resources/blog/4-important-kanban-principles-to-remember-when-leading-an-agile-team/. [Kasutatud 18 aprill 2023].

[6] Atlassian, „Jira Software,“ 2023. [Võrgumaterjal]. Available: https://www.atlassian.com/software/jira/guides/boards/overview#what-is-a-jira-board. [Kasutatud 19 aprill 2023].

[7] International Business Machines Corporation, „What is software testing?,“ 2023. [Võrgumaterjal]. Available: https://www.ibm.com/topics/software-testing. [Kasutatud 27 aprill 2023].

[8] S. Pitter, „Types of software testing,“ 2023. [Võrgumaterjal]. Available: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing. [Kasutatud 27 aprill 2023].

[9] L. R. M. S.-G. Ricardo Colomo-Palacios, „Beyond Technical Skills in Software Testing: Automated versus Manual Testing,“ Østfold University College, 25 september 2020. [Võrgumaterjal]. Available: https://dl.acm.org/doi/pdf/10.1145/3387940.3392238. [Kasutatud 28 aprill 2023].

[10] J. Unadkat, „Manual Testing Tutorial for Beginners,“ BrowserStack, 9 märts 2023. [Võrgumaterjal]. Available: https://www.browserstack.com/guide/manual-testing-tutorial. [Kasutatud 22 aprill 2023].

[11] M. D. Silva, „Pros and Cons of Manual Testing,“ 8 veebruar 2022. [Võrgumaterjal]. Available: https://uilicious.com/blog/pros-and-cons-manual-testing/#what-is-manual-testing-and-what-are-its-benefits. [Kasutatud 27 aprill 2023].

[12] T. Hamilton, „Manual Testing Tutorial: What is, Types, Concepts,“ 8 aprill 2023. [Võrgumaterjal]. Available: https://www.guru99.com/manual-testing.html. [Kasutatud 28 aprill 2023].

[13] T. Hamilton, „What is BLACK Box Testing? Techniques, Types & Example,“ 8 aprill 2023. [Võrgumaterjal]. Available: https://www.guru99.com/black-box-testing.html. [Kasutatud 28 aprill 2023].

[14] T. Hamilton, „White Box Testing – What is, Techniques, Example & Types,“ 25 märts 2023. [Võrgumaterjal]. Available: https://www.guru99.com/white-box-testing.html. [Kasutatud 28 aprill 2023].

[15] T. Hamilton, „Unit Testing Tutorial – What is, Types & Test Example,“ 4 märts 2023. [Võrgumaterjal]. Available: https://www.guru99.com/unit-testing-guide.html. [Kasutatud 28 aprill 2023].

[16] T. Hamilton, „What is System Testing? Types with Example,“ 4 märts 2023. [Võrgumaterjal]. Available: https://www.guru99.com/system-testing.html. [Kasutatud 28 aprill 2023].

[17] T. Hamilton, „Integration Testing: What is, Types with Example,“ 4 märts 2023. [Võrgumaterjal]. Available: https://www.guru99.com/integration-testing.html. [Kasutatud 28 aprill 2023].

[18] T. Hamilton, „What is User Acceptance Testing (UAT)? Examples,“ 8 aprill 2023. [Võrgumaterjal]. Available: https://www.guru99.com/user-acceptance-testing.html. [Kasutatud 28 aprill 2023].

[19] E. Pelivani ja B. Cico, „A comparative study of automation testing tools for web applications,“ %1 *2021 10th MEDITERRANEAN CONFERENCE ON EMBEDDED COMPUTING*, BUDVA, MONTENEGRO, 2021.

[20] E. Kinsbruner, A Frontend Web Developer's Guide to Testing, Birmingham: Packt Publishing Ltd., 2022.

[21] S. Bose, „Automation Testing Tutorial: Getting Started,“ 13 veebruar 2023. [Võrgumaterjal]. Available: https://www.browserstack.com/guide/automation-testing-tutorial. [Kasutatud 29 aprill 2023].

[22] P. Pedamkar, „What is Automation Testing?,“ 2023. [Võrgumaterjal]. Available: https://www.educba.com/what-is-automation-testing/?source=leftnav. [Kasutatud 30 aprill 2023].

[23] P. Pedamkar, „Benefits of Automation Testing,“ 2023. [Võrgumaterjal]. Available: https://www.educba.com/benefits-of-automation-testing/. [Kasutatud 1 mai 2023].

[24] T. Joseph, „What Are the Limitations of Automation Testing?,“ 24 märts 2021. [Võrgumaterjal]. Available: https://blog.qasource.com/resources/what-are-the-limitations-of-automation-testing. [Kasutatud 3 mai 2023].

[25] A. Patt, „The Pros and Cons of Automated Testing,“ 27 mai 2021. [Võrgumaterjal]. Available: https://test.io/resources/blog/the-pros-and-cons-of-automated-testing. [Kasutatud 3 mai 2023].

[26] M. D. Silva, „Pros and Cons of Automated Testing,“ 31 jaanuar 2022. [Võrgumaterjal]. Available: https://uilicious.com/blog/pros-cons-automated-testing/#what-are-the-cons-of-automated-testing. [Kasutatud 3 mai 2023].

[27] Prolifics, Inc., „Types of automated testing explained,“ 2022. [Võrgumaterjal]. Available: https://prolifics.com/types-of-automated-testing/. [Kasutatud 02 mai 2023].

[28] S. Bose, „Functional Testing : A Detailed Guide,“ 11 mai 2021. [Võrgumaterjal]. Available: https://www.browserstack.com/guide/functional-testing. [Kasutatud 4 mai 2023].

[29] T. N. Geek, „What is Smoke Testing?,“ 10 veebruar 2024. [Võrgumaterjal]. Available: https://www.browserstack.com/guide/smoke-testing. [Kasutatud 4 mai 2023].

[30] S. Jain, „What is Non-Functional Testing?,“ 3 veebruar 2023. [Võrgumaterjal]. Available: https://www.browserstack.com/guide/what-is-non-functional-testing. [Kasutatud 4 mai 2023].

[31] S. Bose, „Performance Testing: A Detailed Guide,“ 23 juuli 2022. [Võrgumaterjal]. Available: https://www.browserstack.com/guide/performance-testing. [Kasutatud 4 mai 2023].

[32] Kitakabee, „Automated Regression Testing: A Detailed Guide,“ 10 veebruar 2023. [Võrgumaterjal]. Available: https://www.browserstack.com/guide/automated-regression-testing. [Kasutatud 4 mai 2023].

[33] T. Hamilton, „Keyword Driven Testing Framework with Example,“ 11 märts 2023. [Võrgumaterjal]. Available: https://www.guru99.com/keyword-driven-testing.html. [Kasutatud 4 mai 2023].

[34] T. Hamilton, „What is Data Driven Testing? Learn to create Framework,“ 18 märts 2023. [Võrgumaterjal]. Available: https://www.guru99.com/data-driven-testing.html. [Kasutatud 4 mai 2023].

[35] TESTRIG, „TOP WEB APPLICATION TESTING TOOLS OF 2023(MOST DEMANDED),“ 2023. [Võrgumaterjal]. Available: https://www.testrigtechnologies.com/know-the-best-web-application-testing-tools-in-2020/. [Kasutatud 5 mai 2023].

[36] GH ja A. Badkar, „Popular Test Automation Frameworks,“ 9 märts 2023. [Võrgumaterjal]. Available: https://www.browserstack.com/guide/best-test-automation-frameworks. [Kasutatud 5 mai 2023].

[37] npm trends, „cypress vs selenium-cypress vs ghost-inspector vs lambda-tester vs playwright vs selenium-webdriverwebdriver,“ 5 mai 2023. [Võrgumaterjal]. Available: https://npmtrends.com/cypress-vs-ghost-inspector-vs-lambda-tester-vs-playwright-vs-selenium-webdriver. [Kasutatud 5 mai 2023].

[38] J. Unadkat, „Cypress vs Selenium: Key Differences,“ 14 veebruar 2023. [Võrgumaterjal]. Available: https://www.browserstack.com/guide/cypress-vs-selenium. [Kasutatud 5 mai 2023].

[39] Selenium, „Page object models,“ 9 november 2022. [Võrgumaterjal]. Available: https://www.selenium.dev/documentation/test_practices/encouraged/page_object_models/. [Kasutatud 6 mai 2023].

[40] Cypress.io, „Key Differences,“ 2023. [Võrgumaterjal]. Available: https://docs.cypress.io/guides/overview/key-differences. [Kasutatud 6 mai 2023].

[41] K. Pathak, „Cypress Vs Playwright Vs Selenium Comparison,“ 12 jaanuar 2023. [Võrgumaterjal]. Available: https://kailash-pathak.medium.com/playwright-vs-selenium-vs-cypress-a-detailed-comparison-d3a07ee1d002. [Kasutatud 6 mai 2023].

[42] E. Kurt, „Which Automation Tool is the Best: Selenium Web Driver| Cypress | WebdriverIO | TestCafe | Playwright,“ 30 juuni 2021. [Võrgumaterjal]. Available: https://medium.com/codex/which-automation-tool-is-the-best-selenium-cypress-webdriverio-testcafe-playwright-c56c6f22df1f. [Kasutatud 5 mai 2023].

[43] A. Enin, „Advantages and Disadvantages between Cypress and Playwright,“ 5 juuli 2022. [Võrgumaterjal]. Available: https://adequatica.medium.com/advantages-and-disadvantages-between-cypress-and-playwright-6b29e3989108. [Kasutatud 6 mai 2023].

[44] AltexSoft, „The Good and the Bad of Katalon Studio Automation Testing Tool,“ 15 juuni 2021. [Võrgumaterjal]. Available: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-katalon-studio-automation-testing-tool/. [Kasutatud 6 mai 2023].

[45] PeerSpot, „Katalon Studio vs Ranorex Studio vs Tricentis Tosca comparison,“ aprill 2023. [Võrgumaterjal]. Available: https://www.peerspot.com/products/comparisons/katalon-studio_vs_ranorex-studio_vs_tricentis-tosca. [Kasutatud 6 mai 2023].

[46] Tricentis Tosca, „About Tricentis Tosca,“ 2023. [Võrgumaterjal]. Available: https://documentation.tricentis.com/tosca/1520/en/content/resources/webhelp/about_tricentis_tosca.htm. [Kasutatud 6 mai 2023].

[47] Y. Sharma, „What are the advantages and disadvantages of the Ranorex Testing Tool | Software Testing tool,“ 7 märts 2022. [Võrgumaterjal]. Available: https://tutorialslink.com/Articles/What-are-the-advantages-and-disadvantages-of-the-Ranorex-Testing-Tool-Software-Testing-tool/3321. [Kasutatud 6 mai 2023].

[48] M. Shah, „Ranorex: Pros & Cons of GUI Test Automation Tools,“ 9 november 2020. [Võrgumaterjal]. Available: https://www.impactqa.com/blog/ranorex-pros-cons-of-gui-test-automation-tools/. [Kasutatud 6 mai 2023].

[49] Cypress.io, „Installing Cypress,“ 2023. [Võrgumaterjal]. Available: https://docs.cypress.io/guides/getting-started/installing-cypress. [Kasutatud 7 mai 2023].

[50] Microsoft, „Installation,“ 2023. [Võrgumaterjal]. Available: https://playwright.dev/docs/intro#installing-playwright. [Kasutatud 7 mai 2023].

[51] Cypress.io, „Writing Your First E2E Test,“ 2023. [Võrgumaterjal]. Available: https://docs.cypress.io/guides/end-to-end-testing/writing-your-first-end-to-end-test. [Kasutatud 11 mai 2023].

[52] Microsoft, „Writing tests,“ 2023. [Võrgumaterjal]. Available: https://playwright.dev/docs/writing-tests. [Kasutatud 11 mai 2023].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Jolanta Rudus

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Kasutajaliidese automaattestimise raamistike analüüs ja võrdlus ettevõttes MRPeasy OÜ kasutusele võtmiseks“, mille juhendaja on Jekaterina Tšukrejeva.
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

17.05.2023

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Küsitlus ettevõtte MRPeasy OÜ testijatele

MRPeasy OÜ testijate küsitlus testimisraamistiku valimiseks

* Viitab kohustuslikule küsimusele

Cypress raamistiku test

test.cy.js

cypress > e2e > test.cy.js > describe('My First Test') callback

```
describe('My First Test', () => {
    it('Find Cypress documentation from Google and go there', () => {
        cy.visit('https://www.google.ee');
        cy.get('button:contains("Nõustu kõigiga")').click();
        cy.get('[role="combobox"]').type('Cypress installation documentation');
        cy.get('[role="button"]').contains("Google otsing").click();
        cy.contains('Installing - Cypress Documentation').click();
        cy.contains('Cypress');
    });
});
```

Playwright raamistiku test

test.spec.js

tests > test.spec.js > ...

```
import { test, expect } from '@playwright/test';

test('Find Playwright documentation from Google and go there', async ({ page }) => {
    await page.goto('https://google.ee/');
    await page.getByRole('button', { name: 'Nõustu kõigiga' }).click();
    await page.getByRole('combobox').type('Playwright installation documentation');
    await page.getByRole('button', { name: 'Google otsing' }).click();
    await page.getByText('Installation | Playwright').first().click();
    await expect(page).toHaveTitle(/Playwright/);
});
```

Millise raamistiku kood tundub teile loetavam? *

- ◯ Cypress
- ◯ Playwright
- ◯ Mitte kumbki

Miks? *

Your answer

Millise raamistiku graafiline kasutajaliides tundub mugavam? (Töö autor näitas ettevõtte testijatele kasutajaliidese demo) *

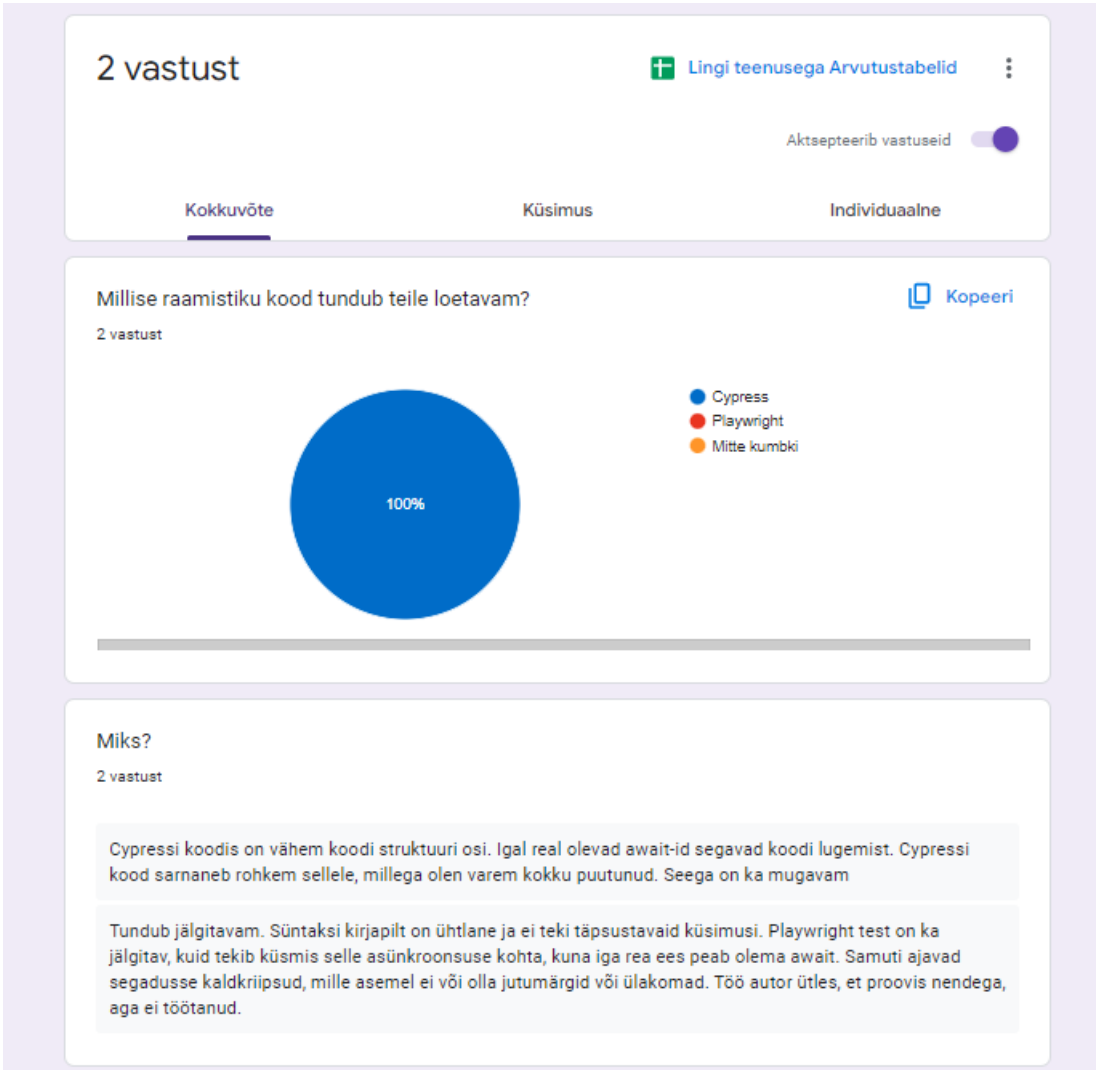
- ◯ Cypress
- ◯ Playwright
- ◯ Mitte kumbki

Miks? *

Your answer

Submit Clear form

Lisa 3 – Küsitluse tulemuste statistika



Millise raamistiku graafiline kasutajaliides tundub mugavam? (Töö autor näitas ettevõtte testijatele kasutajaliidese demo)

Kopeeri

2 vastust

Miks?

2 vastust

Playwright testi jooksutava akna suurus onliiga väike. Liiga suur osa kasutajaliidese vaatest on kaetud osadega, mis pole testijate jaoks töö käigus eriti olulised. Näiteks ülemine testide ajajoon ja testide loetelu.

Kui valida mugavuse järgi, siis meeldib rohkem Cypress. Vähem ebavajalikke osasid ja intuitiivsem.