TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Maria Baltmischkis 154812IABB

AUTOMAATTESTIDE LOOMINE PUHKUSTE HALDUSE SÜSTEEMI SOMNO NÄITEL

Bakalaureusetöö

Juhendaja: Inna Švartsman
MSc

Tallinn 2018

Autorideklaratsioon

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

Autor: Maria Baltmischkis

21.05.2018

Annotatsioon

Käesoleva lõputöö eesmärgiks on uurida hetkel Somno OÜ-s toimuvaid manuaalse testimise protsesse ning leida testide automatiseerimiseks tööriist, millega luuakse kasutajaliidesele automaattestid.

Bakalaureusetöö teoreetilises osas kirjeldatakse tarkvara testimist ning eelkõige automaattestimist. Tuuakse välja kaks tööriista, nende kirjeldused ja kirjutatakse detailsemalt tööriistast, millega luuakse automaattestid.

Praktilises osas analüüsib ja loob autor testplaani, millele tuginedes kirjutab kasutajaliidese olulisematele funktsionaalsustele automaattestid. Automaattestid kirjutatakse veebipõhisele rakendusele Somno, mis võimaldab ettevõtetel hallata puhkuseid ja kontorist eemalolekuid. Testide loomiseks kasutatakse tööriista Laravel Dusk, mis on Seleniumil põhinev.

Töö oluliseks tulemuseks on detailselt kirjeldatud testlood ja nende põhjal loodud automaattestid, mis vähendavad ettevõttes testimise ajakulu ning aitavad kaasa tarkvara kvaliteedi tõstmisele.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 24 leheküljel, 5 peatükki, 8 joonist, 1 tabelit.

Abstract

Development of Automated Tests for on Leave Management System Somno

The aim of the thesis is to examine manual testing processes in Somno OÜ and to find a tool to create automated UI tests.

At first is described the concept of software testing and in particular automated testing. The author brings out two tools, their descriptions and describes in detail the one that is used to write automated tests.

The second part of the thesis consists of analyzing the system and writing a test plan. Test plan consists of testcases that are written to cover the most important functionalities of the system. Tests are written based on the testcases to test a cloud-based system Somno, which helps companies to manage vacations and on leave data in one place. The tests are written with a tool called Laravel Dusk, which is basically a wrapper around Selenium.

The most important result of the thesis is to reduce the time spent on manual testing and to make it more effective through implementing automated tests. That way software quality improves, and developers and testers can allocate their time to create new functionalities rather than testing every step manually.

The thesis is in Estonian and contains 24 pages of text, 5 chapters, 8 figures, 1 table.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface* Rakendusliides [8]
IDE	*Integrated Development Environment* Integreeritud arenduskeskkond [9]
Regressioon testimine	Testimise meetod, kus kontrollitakse, et muudatused ja uued funktsionaalsused ei ole muutnud olemasolevaid funktsionaalsuseid vigasteks. Jooksutatakse alati samasuguseid teste. [7]
UI	*User interface* Kasutajaliides [10]

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Testimine on tarkvara arenduse üks olulisemaid osi, sest see võimaldab leida vigu, tagab süsteemi korrektse toimimise ja näitab süsteemi kvaliteeti. See peaks olema sama oluline kui on tarkvara arendus või disaini loomine, kuid tihti alahinnatakse seda ning ei mõelda testimise lihtsustamisele. Testitakse manuaalselt, mis toob kaasa vigade mittemärkamise ja suure ajakulu [17].

Antud lõputöö eesmärgiks on uurida Somno OÜ-s toimuvaid manuaalse testimise protsesse ning leida testide automatiseerimiseks tööriist, millega luuakse kasutajaliidesele automaattestid, et vähendada manuaalsele testimisele kuluvat aega. Eesmärgiks on luua testplaan, millele tuginedes kirjutatakse kasutajaliidesele automaattestid.

1.1 Taust ja probleem

Idufirma Somno OÜ loodi 2015. aastal paariliikmelise meeskonnana, kus arendusprotsessid olid paigas, kuid valdav osa testimist tehti manuaalselt. Tarkvaraarendajad kirjutasid ühikteste, aga uute funktsionaalsuste ja väikeste muudatuste korral tuli kasutajaliidest testida manuaalselt. Alguses ei olnud testitavate funktsionaalsuste hulk väga suur, mistõttu oli vigu leida kerge. Süsteemi täienemisel oli aga näha, et inimene ei suuda kõiki testlugusid läbi teha. Tehakse läbi vaid kindel põhiprotsess, et veenduda, kas funktsionaalsus toimib ka peale uusi arendusi, kuid jäetakse kõrvale alternatiivstsenaariumid, mis ei pruugi anda positiivseid tulemusi. Kuna tegevused on korduvad, siis tekivad vead ning mõned olulised probleemid süsteemis jäävad märkamata. Samuti on testitavate funktsionaalsuste hulk suurenenud nii palju, et ajaliselt ei ole mõistlik jätkata manuaalse testimisega ning on vaja luua automaattestid, mis hoiavad inimressursi aega kokku ja läbivad alati samasuguseid samme.

1.2 Ülesande püstitus

Bakalaureusetöö raames luuakse kasutajaliidese testid, mis käivituvad automaatselt peale iga uut koodi muudatust. Testid luuakse vastavalt testplaanis kirja pandud nõuetele, mis hõlmavad olulisemaid funktsionaalsusi.

1.3 Metoodika

Püstitatud eesmärkide saavutamiseks analüüsib autor ettevõttes olemasolevaid testimise protsesse, uurib automaattestide loomiseks võimalikke tööriistu ja lahendusi ning valib sobivaima.

Lõputöö peatükis 2 kirjeldatakse tarkvara testimist ja detailsemalt automaattestimist. Tuuakse välja kaks tööriista, millega saab automaatteste luua ning valitakse üks. Valitud tööriista kasutamist kirjeldatakse detailsemalt ning tuuakse näide, kuidas seda kasutada.

Peatükis 3 luuakse testplaan, kuhu kuulub testplaani olemuse selgitamine ja selle loomine koos ettevõtte tutvustuse, testprotsessi ja valitud funktsionaalsuste kirjelduste ja nõuetega. Vastavalt testplaanile luuakse automaattestid.

Viimases, 4. peatükis analüüsitakse testide loomise käiku, nende tulemusi ning automaattestide efektiivsust võrreldes manuaalse testimisega.

2 Tarkvara testimine

Tarkvara testimine on tegevus, mis toimub peale lähtekoodis tehtud muudatuste tegemist ning mille käigus testitakse süsteemi erinevaid osi. See toimub selleks, et leida ja parandada võimalikult palju vigu enne kui muudatused jõuavad klientideni. Kui 1980. aastatel tähendas testimine tegevusi, mida tehti peale tarkvara klientidele kättesaadavaks tegemist, siis tänapäeval tähendab see koodi testimist ja kontrollimist enne kui see jõuab klientideni. See võimaldab probleeme leida varakult ning kulud vigade parandamiseks on väiksemad [22].

1990. aastate alguses tuli turule mitmeid testide haldamise tööriistu, mis muutsid testimise mugavamaks ning vähendasid manuaalselt tehtavat tööd. Nende abil oli võimalik teste uuesti jooksutada ja vaadata tulemusi raportitest, mis aitasid defekte tuvastada. Kui esialgu olid sellised tööriistad algelised ja tarkvaraarendaja pidi ise raporteid uurima ja sealt vigu leidma, siis tänapäeval on taolisi tööriistu edasi arendatud ning tarkvaraarendajal ei tule nii palju teha. Automaattestideks mõeldud keskkondi on võimalik paigaldada vähese vaevaga ning tööriistu saab ka omavahel integreerida, mis annab ettevõtetele võimaluse mitmed käsitsi tehtavad tegevused muuta automaatseteks [22].

2.1 Automaattestimine

Automaattestimine on tarkvara testimise meetod, mis kasutab spetsiaalseid tarkvaralisi tööriistu, et kontrollida testide läbitavust ja võrrelda saadud tulemusi oodatud tulemustega. Kogu tegevus toimub automaatselt ning tarkvaraarendaja või testija ei pea seda tegema manuaalselt [11].

Automaattestimist kasutavad ettevõtted, kus on vaja samasuguseid teste korrata igapäevaselt. Vastasel juhul peaks funktsionaalsuste täienemisel suurendama töötajate arvu, kes testiks manuaalselt [13]. Manuaalsel testimisel võivad inimesed teha vigu, sest kordavad samu teste igapäevaselt, näevad sarnaseid tulemusi ning võivad jätta mõned

sammud vahele ning ei märka seejärel tekkinud vigu. Automaattestid aga kordavad teste alati samamoodi ning tõenäosus vigu teha on väiksem [19].

Automaatteste saab jooksutada regulaarselt ning iga uue muudatusega koodis võimaldab see leida probleeme varajases staadiumis ning vähendab töötajate aega, mida kulutataks vea leidmiseks ja parandamiseks [17]. Testid aitavad ettevõtetel vigu ennetada, mis vähendavad klientideni jõudvate defektide arvu. Seeläbi tõuseb tarkvara kvaliteet ja vähenevad toomis- ja hoolduskulud [22].

Hästi töötavate automaattestide aluseks on korrektselt kirjeldatud tarkvara nõuded, analüüs ja testplaan, millele tuginedes saab luua töötavad testid, mis kontrollivad soovitud osi süsteemist. Nõuetele vastavus on oluline selleks, et loodaks testid, mis kontrollivad, kas süsteem teeb seda, mida on analüüsis ja testplaanis kirjeldatud [22].

Antud lõputöös kasutatakse testide loomiseks Laravel Dusk tööriista, mis on oma olemuselt mähis ümber Seleniumi. See võimaldab kasutada lihtsat API-t, testimaks veebirakenduse kasutajaliidese poolt [16].

2.2 Selenium IDE ja WebDriver

Selenium on brauserite automatiseerimiseks kasutatavate tööriistade komplekt. See töötab kõigi peamiste brauseritega nagu Firefox, Internet Explorer, Google Chrome, Safari ja Opera [21].

Seleniumi puhul teatakse kahte kõige levinumat tööriista – Selenium IDE ja WebDriver. IDE on avatud lähtekoodiga projekt, mida testijad ja tarkvaraarendajad saavad kasutada brauseripõhiste regressioonitestide tegemiseks. Seda kasutatakse selleks, et salvestada töövoogude samme ning arendajad saavad selle abil vältida koodi edasist regressiooni. IDE puuduseks on see, et see töötab ainult Firefox brauseriga ning teste saab teha vaid selle põhjal, mida kuvatakse ekraanile [13]. Antud raamistik on laialt kasutusel, sest teste saab salvestada Firefox veebibrauserisse lisatava laienduse abil, mis koostab skripti vastavalt testimise käigus tehtavatele sammudele. Skripti saab hiljem eksportida ka populaarsetesse programmeerimiskeeltesse nagu Java, C#, Ruby ja Python [18].

WebDriver on tööriist, millega saab kirjutada teste ning mis töötab kõigi brauseritega. Skripte on võimalik kirjutada erinevates programmeerimiskeeltes nagu Java, Python,

Ruby, C#, PHP, Perl ja JavaScript. WebDriveri puhul saab kasutada erinevaid atribuute, mida on HTML-is ja CSS-is kasutatud. See teeb testid efektiivsemaks ning testitavaid elemente on kergem tuvastada. Testide käivitamisel on võimalik valida, kas käivitatakse terve test korraga või mõni kindel osa [4].

2.3 Laravel Dusk

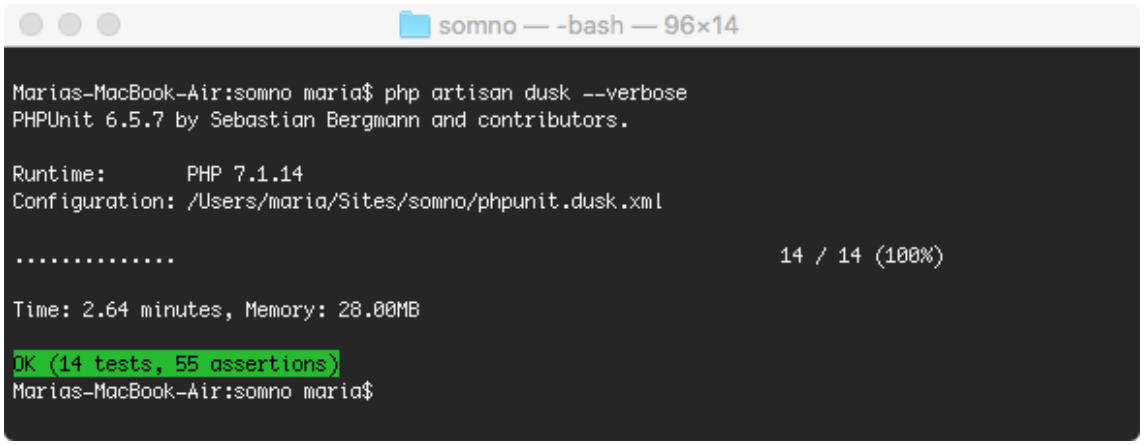
Antud lõputöös ei kasutata automaattestide kirjutamiseks Selenium IDE-t ega WebDriverit, vaid Laravel Duski, mis toetub Seleniumile. Sarnaselt Seleniumile on võimalik ka Laravel Dusk tööriistaga valida, millist osa testist soovitakse parasjagu testida ning alati ei pea käivitama tervet testi korraga. Laravel Duski kasutatakse veebirakenduste testimiseks, mis on kirjutatud Laravel raamistikule [16].

Dusk toetub veel ChromeDriverile ja Facebook Php-webdriverile, mis teeb tööriista kasutamise lihtsamaks ja ei ole vaja üles seadistada Seleniumit ennast. Testid jooksevad automaatselt Chrome brauseris, kuid neid on võimalik jooksutada ka teistes brauserites, sealhulgas nn *headless* brauserites, millel ei ole graafilist kasutajaliidest. Üks olulisemaid Duski tunnuseid on võime oodata näiteks JavaScripti komponendi või CSS selektori laadimise järgi. Testiga ei minda edasi enne, kui määratud element on ilmunud ekraanile. Vigade tekkimisel teeb Dusk olukorrast ekraanipildi ja salvestab selle maha. Ekraanipildilt aga ei ole võimalik iga kord viga tuvastada ning teise võimalusena salvestatakse maha ka brauseri konsoolis kuvatud veateade, mis on detailsem ja aitab tarkvaraarendajal või testijal vea tuvastada [12].

2.3.1 Laravel Duski kasutamine

Laravel Duskiga saab uue testi luua käsuga `php artisan dusk:make DepartmentTest`.

Kõikide testide käivitamiseks tuleb kasutada käsku `php artisan dusk` ja ühe kindla testi käivitamiseks `php artisan dusk tests/Browser/DepartmentTest.php` [5]. Järgneval joonisel (Joonis 1) on näha kõikide testide käivitamine käsurealt.



Joonis 1. Testide käivitamine käsureal.

Järgnevalt esitatud koodinäide (Joonis 2) on Laravel Duski abil kirjutatud kasutajaliidese automaattest rakendusele Somno, millega testitakse osakonna kustutamist, testlugu F03. Vastavalt testloos kirjeldatud eeltingimustele luuakse uus kasutaja, kellele määratakse testloos kirja pandud roll ning luuakse osakond, mis kustutatakse testi käigus.

```
public function testDeleteDepartment()
{
    $user = TestHelper::createAdminUser();
    factory(Department::class)->create();

    $this->browse(function (Browser $browser) use ($user) {
        $browser->loginAs($user)
            ->visit(new Departments())
            ->waitFor('.fa-trash-o')
            ->click('.fa-trash-o')
            ->waitFor('@confirmDialog')
            ->press('Yes, delete it!')
            ->seeNotification('The department was deleted
successfully!')
             ->assertMissing('.team-members');
    });
}
```

Joonis 2. Osakonna kustutamise testi kood.

Meetodiga `loginAs()` logitakse kasutaja süsteemi sisse ilma, et peaks kasutajat autentima kasutades sisselogimise veebivormi. Veebivormi kasutamise korral teeks see iga testi läbimise aeglasemaks. Veebileht, mille test peab avama, määratakse ära meetodiga `visit()`, millele antakse argumendina kaasa lehekülje klass. Lehekülje klassis on ära defineeritud aadress, kuhu brauser peab navigeerima. Lisaks sellele on klassis defineeritud otseteed (*shortcuts*), mis võimaldavad anda keerulistele CSS-i

selektoritele arusaadavamad nimed. Näiteks on testis kasutatud otseteed `@confirmDialog`, mida kasutatakse süsteemis kuvatavate teadete jaoks. Antud näites oodatakse teate järgi, mis kuvab „*The department was deleted successfully!*“.

Lisaks eelnevale toimub peale iga testi läbimist andmebaasi tühjendamine, et ei tekiks andmete konflikti ning teste oleks võimalik korrata ja uuesti jooksutada. See on lahendatud kasutades Laravel raamistiku andmebaaside migratsioonide süsteemi.

3 Testplaan

Testplaan on dokument, mis kirjeldab tarkvara testimise eesmärke, ulatust, lähenemisviise ja fookust. Selle kirjapanek on kasulik viis, kuidas läbi töötada vajalikud eesmärgid, et tarkvara oleks nõuetele vastav. See annab hea ülevaate kogu meeskonnale, kuidas ja mismoodi jõutakse vastuvõetava tooteni [22]. Plaani koostamine aitab planeerida projekti testimist ning muudab selle kvaliteetseks protsessiks. Testimise planeerimine on ka osa sellest, et muuta testimine korratavaks. Kui protsesse ei ole võimalik korrata, ei ole võimalik neid täiustada. Peale muudatusi tuleb jooksutada alati samu teste, vastasel juhul ei ole teada, kas uuendused on kaasa toonud vigu [19].

Testplaan koosneb üldjuhul sellistest komponentidest, mida ettevõte peab kõige olulisemaks ning mis vajab dokumenteerimist. Üldlevinud testplaani osad on testitava rakenduse kirjeldus, funktsionaalsused mida testitakse, testlood, oodatud tulemused, tööriistad, mida kasutatakse, protsessi kirjeldus, kuidas teste käivitatakse ja ajakava [14].

Antud lõputöös luuakse automaattestid vastavalt testplaanis sätestatud testlugudele.

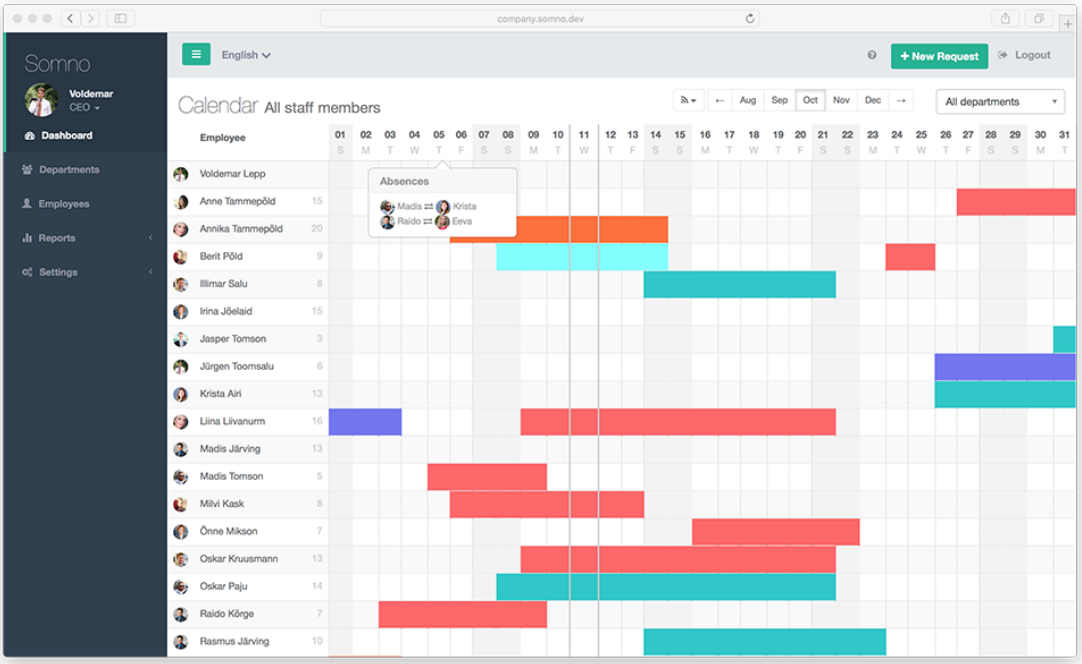
3.1 Testitava süsteemi kirjeldus

Käesolevas lõputöös testitakse ja analüüsitakse veebipõhise süsteemi Somno kasutajaliidest. Antud süsteem võimaldab ettevõtetel ja nende töötajatel esitada ja hallata puhkuseavaldusi ning kontorist eemalolekuid. Igal töötajal on enda ligipääs, mis võimaldab tal esitada avaldusi, neid tühistada või muuta, vaadata puhkusepäevade jääki ja näha, kes on parasjagu töölt eemal, kellel on tulemas sünnipäevad või tööaastapäevad. Süsteemi on võimalik lisada erinevaid osakondi ja määrata kasutajatele, millisesse osakonda nad kuuluvad. Vastavalt süsteemis määratud kinnitusahelale, saavad teatud kasutajad ka avaldusi kinnitada või tagasi lükata.

Süsteemis on neli erinevat rolli, kellel on erinevad õigused. Käesolevas lõputöös on testide kirjutamisel välja toodud administraator ja tavakasutaja. Administraator pääseb

ligi süsteemihaldusele ja saab esitada töötajate eest puhkusetaotlusi. Tavakasutajad saavad hallata enda puhkusetaotlusi, kinnitada või tagasi lükata taotlusi, kus nad on määratud asendajaks või kinnitajaks ning vaadata rakenduses olevat informatsiooni.

Järgeval ekraanipildil (Joonis 3) on näha rakenduse avalehe vaade.



Joonis 3. Rakenduse avalehe vaade.

3.2 Testide käivitamise protsess

Testide käivitamise protsess toimub ettevõtetes erinevalt, sõltuvalt infrastruktuurist, tarkvarast ja meeskonnast. Kindel testimistsükkel lepitakse kokku ettevõtte siseselt ja seega on töötavaid lahendusi olemas väga palju [20]. Turul on mitmeid tööriistu ja rakendusi, mis aitavad teste käivitada, jooksutada regulaarselt ning testide läbimisel paigaldada muudatused *live* serverisse.

Antud lõputöö rakenduses Somno on kasutusel Codeship, Codacy, Bitbucket ja Slack, mis suhtlevad omavahel ja moodustavad töötava terviku.

Codeship – pakub pidevvalmiduse (*continuous delivery*) teenust, kus keskendutakse kiirusele, usaldusväärsusele ja lihtsusele. Codeshipi saab üles seadistada nii, et redaktsioon ja rakenduse paigutus (*build*) toimuks GitHub või Bitbucket teenustes hoiustatud versioonihaldusest. Codeship pakub palju erinevaid variante, kuidas nende

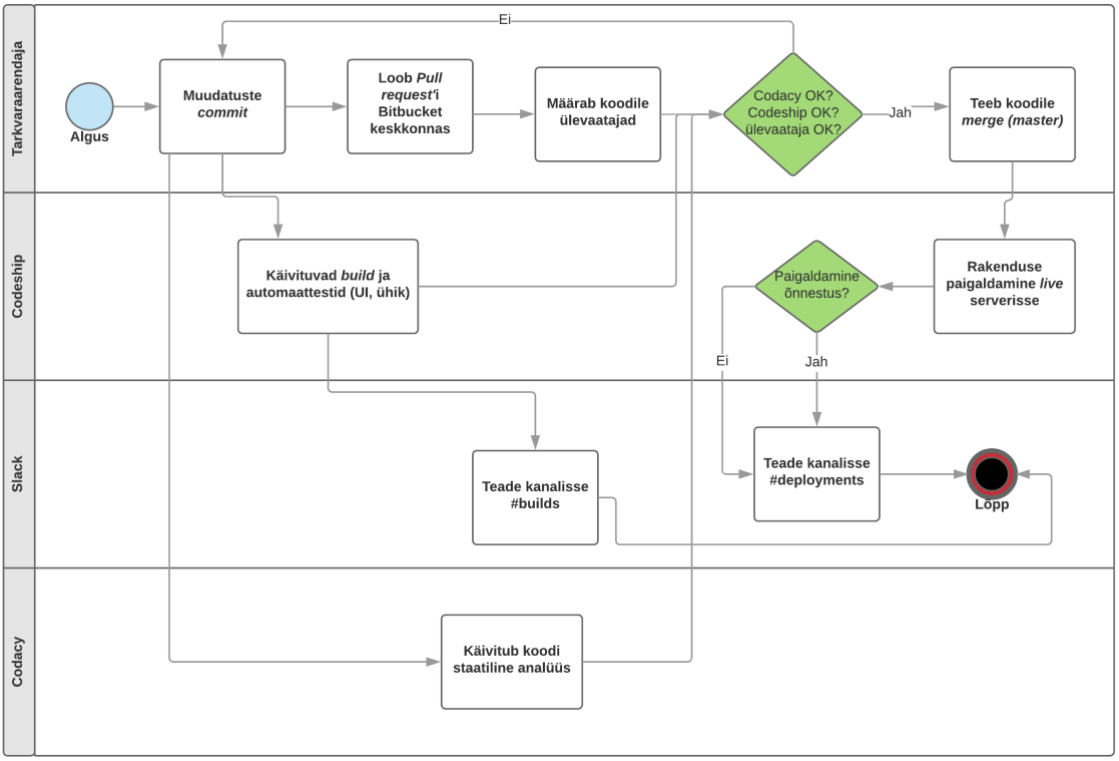
teenust üles seada ja tööle saada vastavalt enda ettevõttes olevatele protsessidele [3]. Somno puhul käivitatakse *build* kohe kui repositooriumisse tekib rakendusest uus redaktsioon. Esmalt paigaldatakse programm Codeshipi virtuaalmasinasse ning seejärel käivitatakse automaatselt ühiktestid ning käesolevas töös loodud kasutajaliidese testid.

Codacy – kvaliteedi tagamise tööriist, mis teostab koodi staatilist analüüsi kontrollides stiili, turvalisust, keerukust ning seda, et koodis ei oleks korduvaid programmilõike. See võimaldab ettevõtetel vähendada koodi ülevaatamisele kuluvat aega, sest rakendus teavitab leitud probleemidest. Tööriista on võimalik integreerida mitmete teiste süsteemidega nagu GitHub, Bitbucket, JIRA, Slack jne [2].

Bitbucket – veebipõhine versioonihalduse majutusteenus. Bitbucket on mõeldud lähtekoodile ja tarkvara projektidele, mis kasutavad kas Mercuriali või Giti versioonihaldussüsteeme [1].

Slack – suhtlusprogramm, mis võimaldab koondada kogu informatsiooni ühte kohta. Slackis saavad meeskonnaliikmed omavahel privaatselt suhelda või moodustada grupivestluseid. Lisaks sellele on Slackil väga palju partnereid ja võimalusi süsteeme üksteisega integreerida. Näiteks on võimalik luua kanal, kuhu Codeship saadab teate, kui kood on paigutatud *live* serverisse või kui süsteemis on ette tulnud vigu, mis nõuavad kohest tähelepanu [6].

Somnos töötavaid protsesse kirjeldab järgnev joonis (Joonis 4).



Joonis 4. Testide käivitamise protsess.

3.3 Testitavad funktsionaalsused ja testlood

Antud lõputöö raames luuakse testid kõige olulisematele funktsionaalsustele, mis katavad ära osa süsteemist, mis on kasutajate poolt igapäevaselt enim kasutusel. Järgnevas peatükis on kirja pandud testlood, mis sisaldavad endas kindlaid tegutsejaid, eeltingimusi, järeltingimusi ja testimise käiku nii põhistsenaariumi(te)na kui ka alternatiivstsenaarimi(te)na.

Testlood on kirja pandud tuginedes raamatus „*Software Testing and Continuous Quality Improvement*“ [22] välja toodud testlugude struktuurile.

F01: Uue osakonna lisamine

Tegutsejad: administraator, rakendus

Eeltingimused:

1. Kasutaja on sisse logitud ja tal on administraatori õigused.
2. Kasutaja asub leheküljel “Osakonnad“.

Järeltingimused:

1. Uus osakond on lisatud.

Põhistsenaarium:

1. Kasutaja klikib nupul „Lisa osakond“.
2. Rakendus kuvab osakonna lisamise modaalakna.
3. Kasutaja täidab vabateksti väljad:
    a. Osakonna nimi;
    b. Kirjeldus.
4. Kasutaja klikib nupul „Lisa järgmine“.
5. Valib listist osakonna kinnitaja.
6. Klikib nupul „Lisa osakond“.
7. Rakendus kuvab teate „Uus osakond lisatud süsteemi“.
8. Kasutaja näeb lisatud osakonda osakondade leheküljel.

Alternatiivstsenaariumid:

1a. Kasutaja klikib nupul „Vajuta siia, et lisada oma esimene osakond“:
    a. Rakendus kuvab osakonna lisamise modaalakna;
    b. Kasutaja klikib nupul „Sulge“;
    c. Rakendus sulgeb lisamise modaalakna.

3a. Kasutaja jätab kõik lahtrid tühjaks:
    a. Klikib nupul „Lisa osakond“;
    b. Rakendus kuvab veateate „Sisestatud andmetes esineb puudusi“.

6a. Kasutaja klikib nupul Lisa järgmine”:
    a. Rakendus lisab osakonna süsteemi.
    b. Kuvab tühjade väljadega modaalakna uue osakonna lisamiseks.
    c. Kasutaja täidab vabateksti väljad:
        i. Osakonna nimi;
        ii. Kirjeldus.
    d. Klikib nupul „Lisa osakond“.
    e. Rakendus kuvab teate „Uus osakond lisatud süsteemi“.

Ekraanipilt osakonna lisamise modaalaknast on kuvatud Lisas 1 (Joonis 9).

F02: Osakonna redigeerimine
Tegutsejad: administraator, rakendus
Eeltingimused: 1. Kasutaja on sisse logitud ja tal on administraatori õigused. 2. Kasutaja asub leheküljel “Osakonnad“.
Järeltingimused: 1. Osakonna informatsioon on muudetud.
Põhistsenaarium: 1. Kasutaja klikib pliiatsi ikoonil. 2. Rakendus kuvab redigeerimise modaalakna. 3. Kasutaja muudab sisu väljal „Osakonna nimi“. 4. Kustutab sisu väljal „Kirjeldus“. 5. Klikib prügikasti ikoonil ja kustutab sellega lisatud kinnitaja. 6. Klikib nupul „Salvesta“. 7. Rakendus kuvab teate „Andmed on muudetud“.

F03: Osakonna kustutamine
Tegutsejad: administraator, rakendus
Eeltingimused: 1. Kasutaja on sisse logitud ja tal on administraatori õigused. 2. Kasutaja asub leheküljel „Osakonnad“. 3. Osakonda ei ole lisatud ühtegi töötajat.
Järeltingimused: 1. Osakond on süsteemist kustutatud.
Põhistsenaarium: 1. Kasutaja klikib prügikasti ikoonil. 2. Rakendus kuvab kinnitusakna „Oled sa kindel, et soovid seda osakonda kustutada?“ ning nupud „Tühista“ ja „Jah, kustuta!“. 3. Kasutaja klikib nupul „Jah, kustuta!“. 4. Rakendus kuvab teate „Osakond on edukalt kustutatud!“.

F04: Uue töötaja lisamine
Tegutsejad: administraator, rakendus
Eeltingimused: 1. Kasutaja on sisse logitud ja tal on administraatori õigused. 2. Kasutaja asub leheküljel „Töötajad“.
Järeltingimused: 1. Uus töötaja on lisatud.
Põhistsenaarium: 1. Kasutaja klikib nupul „Lisa uus töötaja“. 2. Rakendus kuvab töötaja lisamise modaalakna. 3. Kasutaja täidab väljad: a. Eesnimi; b. Perenimi; c. Ettevõttesisene ID; d. Sünnipäev; e. Ettevõttes alates; f. E-mail; g. Skype; h. Telefon. 4. Klikib nupul „Lisa töötaja“. 5. Rakendus kuvab teate „Uus töötaja lisatud!“. **Alternatiivstsenaarium:** 3a. Kasutaja jätab kõik lahtrid tühjaks: a. Klikib nupul „Lisa töötaja“. b. Rakendus kuvab veateate „Sisestatud andmetes esineb puudusi“.

Ekraanipilt uue töötaja lisamise modaalaknast on kuvatud Lisas 1 (Joonis 9).

F05: Töötaja profiili redigeerimine
Tegutsejad: administraator, rakendus
Eeltingimused: 1. Kasutaja on sisse logitud ja tal on administraatori õigused. 2. Kasutaja asub leheküljel „Töötajad“.
Järeltingimused: 1. Profiilil on andmed muudetud.
Põhistsenaarium: 1. Kasutaja klikib pliiatsi ikoonil. 2. Rakendus kuvab redigeerimise modaalakna. 3. Kasutaja muudab sisu väljadel „Aadress“, „Skype“, „Telefon“. 4. Klikib nupul „Salvesta“. 5. Rakendus kuvab teate „Töötaja andmed uuendatud“.

F06: Töötaja deaktiveerimine
Tegutsejad: administraator, rakendus
Eeltingimused: 1. Kasutaja on sisse logitud ja tal on administraatori õigused. 2. Kasutaja asub leheküljel „Töötajad“.
Järeltingimused: 1. Töötaja on deaktiveeritud.
Põhistsenaarium: 1. Kasutaja klikib pliiatsi ikoonil. 2. Rakendus kuvab redigeerimise modaalakna. 3. Kasutaja klikib vaheaknal „Seaded“. 4. Rakendus kuvab seaded. 5. Kasutaja klikib nupul „Deaktiveeri kasutaja“. 6. Rakendus kuvab teate „Töötaja deaktiveeritud“. 7. Kasutaja klikib nupul „Sulge". 8. Rakendus ei kuva deaktiveeritud kasutajat töötajate listis.

Ekraanipilt töötaja deaktiveerimise modaalaknast on kuvatud Lisas 1 (Joonis 10).

F07: Puhkuseavalduse esitamine
Tegutsejad: tavakasutaja, rakendus
Eeltingimused: 1. Kasutaja on sisse logitud. 2. Süsteemi on lisatud puhkusetüüp. 3. Kasutajal on piisav puhkusepäevade jääk.
Järeltingimused: 1. Taotlus on esitatud.
Põhistsenaarium: 1. Kasutaja klikib nupul „+ Uus taotlus“. 2. Rakendus kuvab modaalakna avalduse esitamiseks. 3. Kasutaja täidab väljad: a. Periood; b. Eemaloleku liik; c. Asendaja; d. Tasu maksmine. 4. Klikib nupul „Saada kinnitamisele“. 5. Rakendus saadab avalduse kinnitamisele vastavalt kinnitamisahelas määratud inimesele.

F08: Puhkuseavalduse kinnitamine
Tegutsejad: tavakasutaja, rakendus
Eeltingimused: 1. Kasutaja on sisse logitud. 2. Kasutaja on määratud taotluse kinnitajaks.
Järeltingimused: 1. Avaldus on kinnitatud.
Põhistsenaarium: 1. Kasutaja klikib nupul „Kinnita“. 2. Rakendus kuvab teate „Puhkuse taotlus kinnitatud“.

F09: Puhkuseavalduse muutmine
Tegutsejad: tavakasutaja, rakendus
Eeltingimused: 1. Kasutaja on sisse logitud. 2. Kasutaja on sisestanud avalduse, mille alguskuupäev on tulevikus.
Järeltingimused: 1. Taotlus on muudetud.
Põhistsenaarium: 1. Kasutaja klikib pliiatsi ikoonil. 2. Rakendus kuvab redigeerimise modaalakna. 3. Kasutaja muudab välja „Periood“ ja „Asendaja“. 4. Kasutaja klikib nupul „Salvesta“. 5. Rakendus saadab avalduse uuesti kinnitamisele.

F10: Puhkuseavalduse tagasilükkamine
Tegutsejad: tavakasutaja, rakendus
Eeltingimused: 1. Kasutaja on sisse logitud. 2. Kasutaja on määratud taotluse kinnitajaks.
Järeltingimused: 1. Taotlus on tagasi lükatud.
Põhistsenaarium: 1. Kasutaja klikib nupul „Lükka tagasi“. 2. Rakendus kuvab modaalakna. 3. Kasutaja sisestab tagasilükkamise põhjuse. 4. Klikib nupul „Lükka tagasi“. 5. Rakendus kuvab teate „Puhkuse taotlus kinnitatud“

Ekraanipilt puhkuseavalduse tagasilükkamise modaalaknast on kuvatud Lisas 1 (Joonis 11).

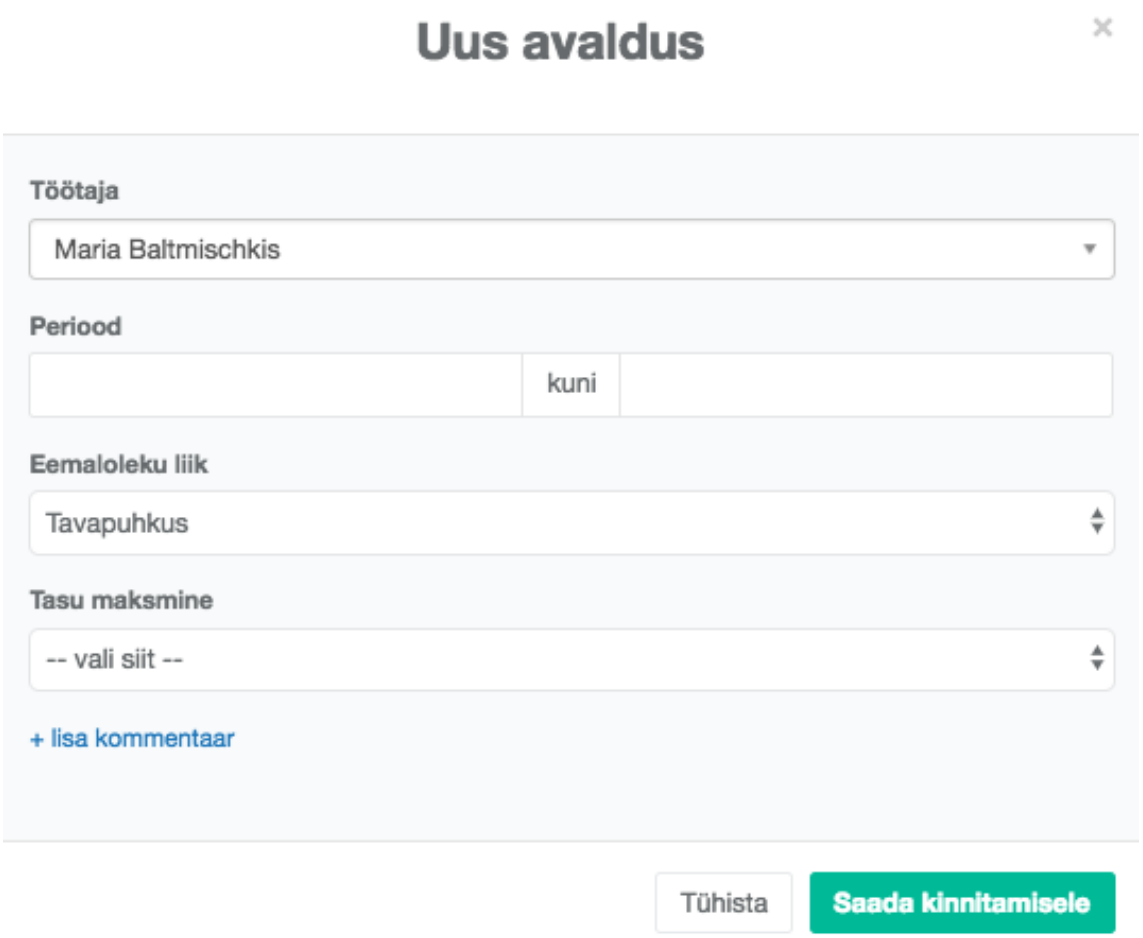
F11: Puhkuse taotluse tühistamine
Tegutsejad: tavakasutaja, rakendus
Eeltingimused: 1. Kasutaja on rakendusse sisse logitud. 2. Kasutaja on sisestanud taotluse, mille alguskuupäev on tulevikus.
Järeltingimused: 1. Taotlus on tühistatud.
Põhistsenaarium: 1. Kasutaja klikib prügikasti ikoonil. 2. Rakendus kuvab tühistamise modaalakna. 3. Kasutaja klikib nupul „Jah, tühista taotlus“. 4. Rakendus kuvab teate „Taotlus on tühistatud“.

F12: Rakendusse sisselogimine
Tegutsejad: kasutaja, rakendus
Eeltingimused: 1. Rakenduses on olemas sisse logitav kasutaja. 2. Kasutaja asub sisselogimise leheküljel.
Järeltingimused: 1. Kasutaja on sisse logitud.
Põhistsenaarium: 1. Kasutaja täidab väljad: a. E-mail; b. Parool. 2. Klikib nupul „Logi sisse“. 3. Rakendus kuvab rakenduse avalehe vaate.

3.4 Testide loomine

Testide loomisel on tuginetud automaattestimise manifestile, milles kirjeldatakse, et testid peavad olema selged, sõltumatud, korratavad, vajalikud, hooldatavad ja efektiivsed [15]. Sellele tuginedes on Somno kasutajaliidese automaattestid tehtud sõltumatuteks, kus ei eeldata, et testis kasutatav informatsioon on juba eelnevalt mõne teise testi poolt salvestatud või eelnevalt andmebaasis olemas. Iga testi alguses luuakse vajalikud andmed ning see tagab iseseisvuse. See vähendab testide mitte läbimise riski, sest kui üks test ei läbi, ei näita see kohe, et ka kõik järgmised testid on vigased, kuna testid ei ole omavahel seotud. Kood on kirjutatud võimalikult lühidalt ning selgelt, et iga lause oleks arusaadav.

Järgmisena on esitatud testloo F07 modaalakna ekraanipilt (Joonis 5) ja kood (Joonis 6).



Joonis 5. Puhkuseavalduse esitamise modaalaken.

```
class AddRequestTest extends DuskTestCase
{
    use DatabaseMigrations;

    public function testAddNewRequestWithApprovalChain()
    {
        $user = TestHelper::createAdminUser([
            'forename' => 'Peter',
            'surname' => 'Pan',
        ]);
        $leaveType = factory(LeaveType::class)->create();
        (new LeaveTypeApprovalChain)-
>initializeChain($leaveType);
        factory(LeaveAllowance::class)->create([
            'user_id' => $user->id,
            'leave_type_id' => $leaveType->id
        ]);
        $substituteUser = factory(User::class)->create();

        $this->browse(function (Browser $browser) use ($user,
$substituteUser) {
            $browser->loginAs($user)
                ->visit(new Dashboard())
                ->press('New Request')
                ->waitFor('.modal.inmodal.in')
                ->assertSee('New Request')
                ->assertMissing('.days-current-leave')
                ->type('date_start', Carbon::today()-
>addDays(2)->format('d.m.Y'))
                ->type('date_end', Carbon::today()->addDays(4)-
>format('d.m.Y'))
                ->select('salary_type', '1')
                ->within(new ChosenSelect('#request-
substitute'), function (Browser $browser) use ($substituteUser)
{
                    $browser->selectValue($substituteUser-
>getRawName());
                })
                ->press('Send for approval')
                ->waitForText('Leave request added!', 30);
        });
    }
}
```

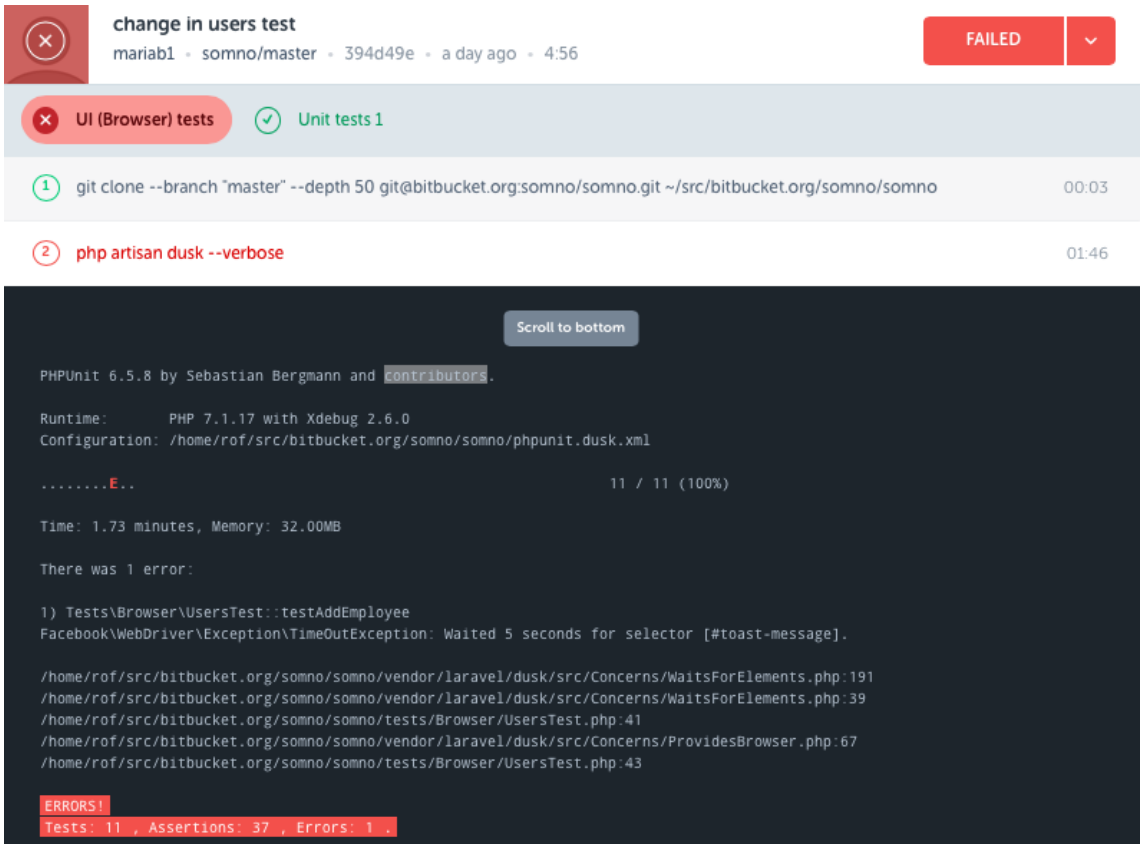
Joonis 6. Puhkuseavalduse esitamise testi kood.

Kõiki loodud teste saab vaadata GitHub keskkonnast lingilt https://github.com/mariab1/somno-ui-tests.

4 Loodud automaattestide analüüs

Käesolevas lõputöös loodi testid vastavalt testplaanis kirjeldatud testlugudele, mis koostati enne testide kirjutamist. See tegi testide kirjutamise mugavamaks, sest testlugudes on kirjas, mida täpselt testist oodatakse, mis on eeltingimus(ed), järeltingimus(ed) ja läbitavad sammud, mis tuleb testi kirja panna ning läbi testida. Küll aga tuli testide loomise käigus välja, et eeltingimusi oli mõne testloo puhul rohkem kui esialgselt dokumentatsiooni kirjutati ning autor pidi tegema vastavad muudatused ka testlugude kirjeldustesse.

Kasutajaliidese testimisel Laravel Duskiga tuli leida igale testitavale elemendile õige CSS-i selektor, et test suudaks leida õige tekstivälja või nupu, mida parasjagu oli vaja testida või mille ilmumist ekraanile tuli oodata. See oli testide loomise juures üks keerulisemaid osi, sest leheküljel, kus testi sooritati, võis olla mitu elementi sama selektoriga ja seetõttu ei läinud test läbi. Test valis sellisel juhul esimese elemendi leheküljel, millel antud selektor küljes oli ning test ei läbinud edukalt. Vigade vältimiseks ja testi muutmiseks stabiilseks tuli teha programmi lähtekoodis muudatusi, lisades elementidele detailsemaid selektoreid. Eelkõige lisati klasse, mille abil oli võimalik testis soovitud elementi määratleda ning andis positiivse tulemuse. Järgnevalt (Joonis 7) on toodud näide Codeship rakendusest, kus test ei läinud läbi ning andis veateate, sest selektorit ei suudetud leida.



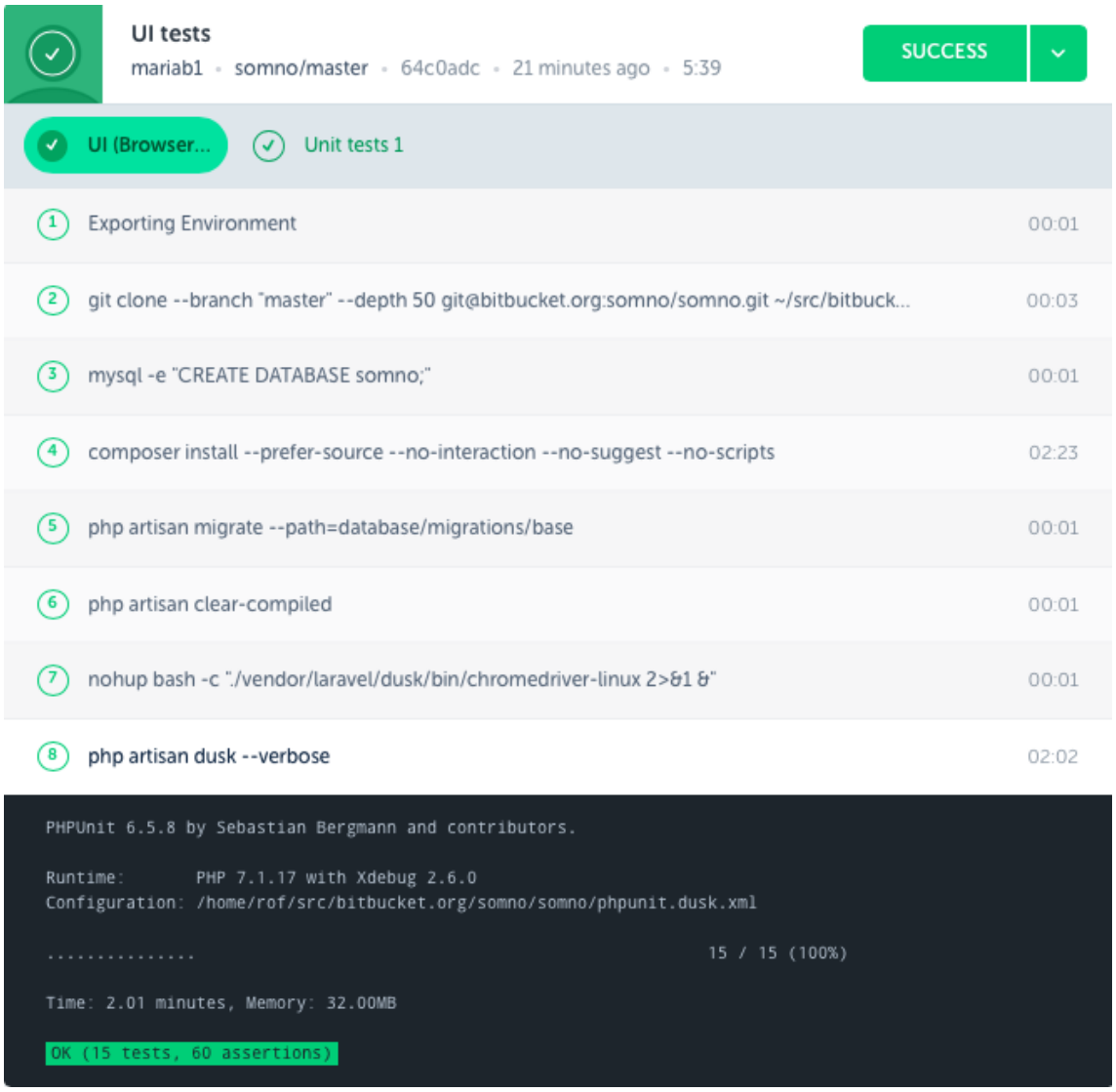
Joonis 7. Veateade Codeship rakenduses.

Testide esialgsel loomisel kirjutas autor igasse testi rakendusse sisselogimise, kus avati rakendus ja sisestati nii e-mail kui ka parool, mille järel tuli oodata lehe laadimist. Sellise lahendusega jooksid testid aeglaselt ja muudatusena viis autor sisse iga testi algusesse Laravel Duski meetodi `loginAs()`, millega logitakse kasutaja süsteemi sisse ilma, et peaks koodi kirjutama sisselogimise andmeid. Lisaks võimaldas selle meetodi kasutamine vähendada testide läbimiseks kuluvat aega, hoides iga testi pealt kokku ligikaudu 1-2 sekundit.

Järgnevas tabelis (Tabel 1) on välja toodud testimise ajakulu, mida tehakse manuaalsel testimisel ning võrdluseks on lisatud aeg, mida kuvab Codeship peale automaattestide jooksutamist. Testide läbimist edukalt kuvatakse Codeshipi ekraanipildil (Joonis 8). Manuaalsel testimisel on lõputöö autor läbinud kõik testlood samas mahus nagu seda tehakse automaattestide puhul. Manuaalsel testimisel mõõdeti aega rakendusega Toggl.

Tabel 1. Manuaalse testimise ja automaattestimise ajakulu võrdlus.

	Manuaalne testimine	**Automaattestimine**
Testide läbimise ajakulu	6 minutit ja 12 sekundit	2 minutit ja 1 sekund



Joonis 8. Automaattestide jooksutamine Codeship rakenduses.

Ajakulu võrdlusest on näha, et automaattestimise puhul läbivad testid kiiremini ning seejuures hoiab testija või tarkvaraarendaja kõigi testlugude testimisel aega kokku, mis oligi oluline eesmärk antud lõputöös. Selle tulemusena saab testija või tarkvaraarendaja pühendada aega funktsionaalsuste täiendamisele ja/või uute automaattestide loomisele.

Hetkel loodud teste saaks edaspidi muuta veel kiiremini läbitavaks, kui iga testiga ei avataks brauserit uuesti ja ei pandaks seejärel ka kinni. Lisaks sellele muudaks kiirust paremaks kui andmebaasi migratsioonid oleksid koondatud nii, et ühes failis oleks kirjeldatud kogu andmebaasi struktuur viimasel kujul. See aga on olulisem tulevikus, kui testlugusi ei ole enam 12, vaid rohkem ning seejuures on kiirus tähtis.

5 Kokkuvõte

Antud lõputöö eesmärgiks oli uurida Somno OÜ-s toimuvaid manuaalse testimise protsesse ning leida automaattestide loomiseks tööriist, millega kirjutada automaattestid ning vähendada nende abil testimisele kuluvat aega ja kiirendada seeläbi vigade leidmist enne koodi paigutust *live* serverisse.

Lõputöö käigus loodi testplaan, mis sisaldas endas testitava süsteemi kirjeldust, testide käivitamise protsessi ning 12 testlugu, mille põhjal loodi automaattestid. Autor tõi välja ja kirjeldas nii Selenium IDE-t, Selenium WebDriverit kui ka Laravel Duski. Automaattestid kirjutati kasutades tööriista Laravel Dusk, mis tugineb Seleniumile, sest Somno on kirjutatud kasutades Laravel raamistikku ning seetõttu tegi antud tööriista kasutamine testide loomise mugavaks.

Igale testloole kirjutas autor automaattesti, mis kontrollis kõiki testloos kirja pandud tingimusi. Testide kirjutamisel osutus kõige keerulisemaks õigete CSS selektorite kasutamine, sest mitmel elemendil oli küljes sama selektor ning seetõttu testid ei läbinud edukalt. Tuli teha muudatusi rakenduse lähtekoodis ja lisada selektoreid juurde, et testid jookseksid korrektselt ja annaksid soovitud tulemusi. Antud muudatus võimaldas leida leheküljelt sobiva elemendi ning testid ei andnud läbimisel veateadet.

Lõputöös püstitatud eesmärk vähendada testimisele kuluvat aega sai oodatud tulemuse: manuaalsele testimisele kulus 6 minutit ja 12 sekundit, automaattestid läbisid Codeship rakenduse vahendusel 2 minutit ja 1 sekund, mis on praeguseks Somno OÜ-s aidanud vähendada testimisele kuluvat aega.

Lõputöö raames loodud automaatteste on võimalik tulevikus muuta kiiremaks, kui andmebaasi migratsioonid oleksid koondatud nii, et ühes failis oleks kirjeldatud kogu andmebaasi struktuur viimasel kujul.

Kasutatud kirjandus

[1] „Bitbucket,“ Atlassian Pty Ltd, [Võrgumaterjal]. Available: https://bitbucket.org/. [Kasutatud 29.04.2018].

[2] „Codacy,“ Qamine limited, [Võrgumaterjal]. Available: https://www.codacy.com/. [Kasutatud 29.04.2018].

[3] „Codeship,“ Codeship Inc., [Võrgumaterjal]. Available: https://codeship.com/. [Kasutatud 29.04.2018].

[4] „Edureka,“ 05.05.2017. [Võrgumaterjal]. Available: https://www.edureka.co/blog/selenium-tutorial. [Kasutatud 23.04.2018].

[5] „Laravel,“ [Võrgumaterjal]. Available: https://laravel.com/docs/5.6/dusk. [Kasutatud 04.04.2018].

[6] „Slack,“ Slack Technologies, [Võrgumaterjal]. Available: https://slack.com/partners. [Kasutatud 29.04.2018].

[7] „SmartBear Software,“ SmartBear Software, [Võrgumaterjal]. Available: https://smartbear.com/learn/automated-testing/what-is-regression-testing/. [Kasutatud 29.04.2018].

[8] „Standardipõhine tarkvaratehnika sõnastik,“ Cybernetica AS, [Võrgumaterjal]. Available: https://stats.cyber.ee/term/2426-api. [Kasutatud 20.04.2018].

[9] „Standardipõhine tarkvaratehnika sõnastik,“ Cybernetica AS, [Võrgumaterjal]. Available: https://stats.cyber.ee/term/3248-ide. [Kasutatud 15.04.2018].

[10] „Standardipõhine tarkvaratehnika sõnastik,“ Cybernetica AS, [Võrgumaterjal]. Available: https://stats.cyber.ee/term/1806-user-interface. [Kasutatud 20.04.2018].

[11] „Techopedia,“ Techopedia Inc, [Võrgumaterjal]. Available: https://www.techopedia.com/definition/17785/automated-testing. [Kasutatud 02.05.2018].

[12] C. Oki, „Scotch,“ Scotch.io LLC, 28.06.2017. [Võrgumaterjal]. Available: https://scotch.io/tutorials/introduction-to-laravel-dusk . [Kasutatud 20.04.2018].

[13] D. Burns, Selenium 1.0 Testing Tools: Beginner’s Guide, Birmingham: Packt Publishing, 2010.

[14] D. Graham, E. V. Veenendaal, I. Evans ja R. Black, Foundations of Software Testing: ISTQB Certification, London: Cengage Learning Emea,

2008.

[15] G. Meszaros, S. M. Smith ja J. Andrea, Extreme Programming And Agile Methods - Xp/Agile Universe, Berlin: Springer-Verlag Berlin Heidelberg, 2003.

[16] J. Montag, „Medium,“ 28.05.2017. [Võrgumaterjal]. Available: https://medium.com/@josiasmontag/automated-browser-testing-with-laravel-dusk-and-browserstack-4edc5fff32a0.

[17] M. Fewster ja D. Graham, Software Test Automation, New York: Addison-Wesley Professional, 1999.

[18] P. Sams, Selenium Essentials, Birmingham: Packt Publishing, 2015.

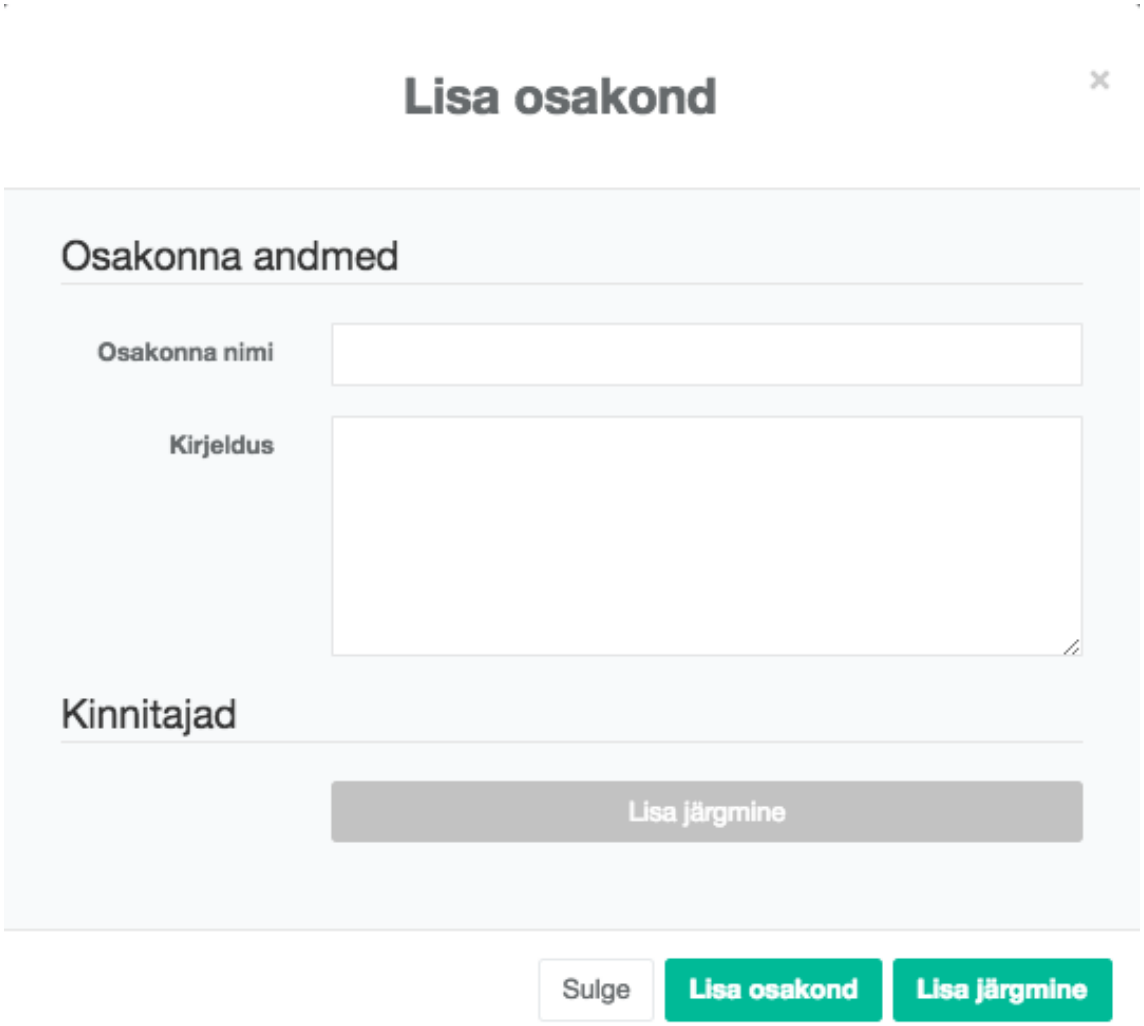
[19] S. McConnell, Code Complete, Redmond: Microsoft Press, 2004.

[20] U. Eriksson, „ReQtest,“ ReQtest AB, 19.05.2016. [Võrgumaterjal]. Available: https://reqtest.com/testing-blog/the-a-to-z-guide-to-the-software-testing-process/) . [Kasutatud 15.04.2018].

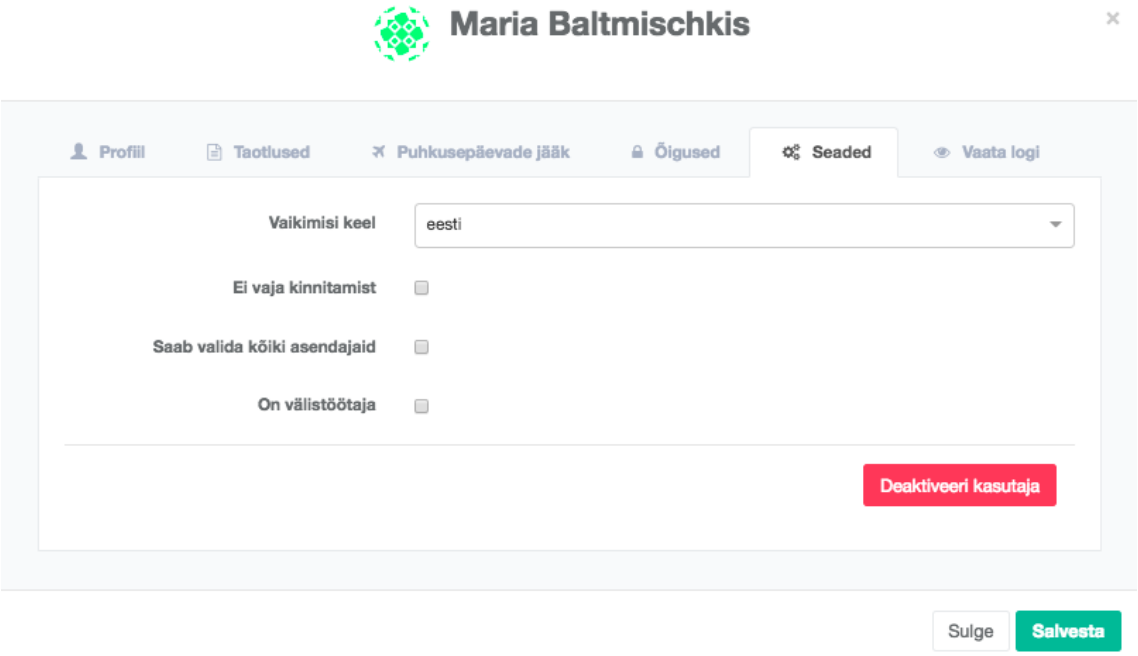
[21] U. Gundecha, Selenium Testin Tools Cookbook - Second Edition, Birmingham: Packt Publishing, 2015.

[22] W. E. Lewis, Software Testing and Continuous Quality Improvement, Boca Raton: Auerbach Publications, 2009.

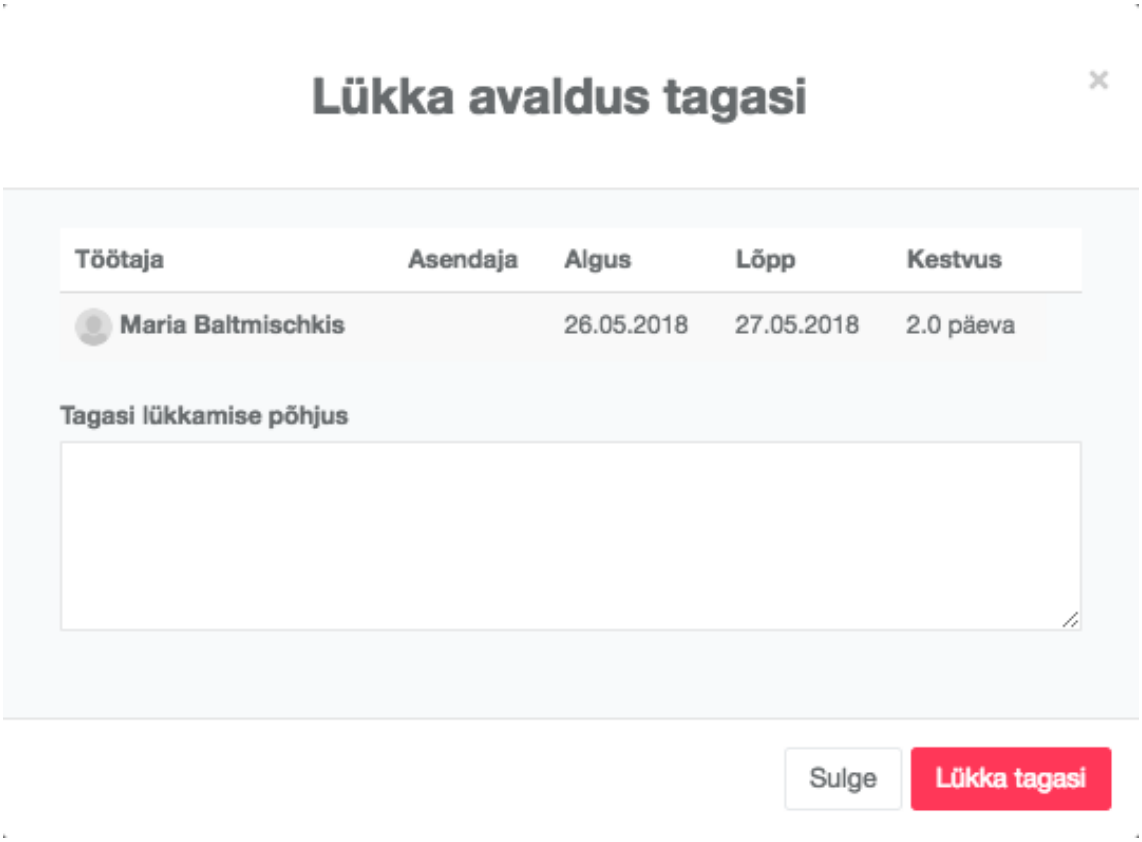
Lisa 1 – Rakenduses kuvatavad modaalaknad



Joonis 9. Uue osakonna lisamise modaalaken.



Joonis 10. Töötaja deaktiveerimise modaalaken.



Joonis 11. Puhkuseavalduse tagasilükkamise modaalaken.