TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kristo Loit IADB179572

Kasutajaliidese automaattestid Valitsusportaali näitel

Bakalaureusetöö

Juhendaja: Maili Markvardt
MSc

Tallinn 2022

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: Kristo Loit

23.09.2021

Annotatsioon

Käesolev bakalaureusetöö lahendab aktuaalset probleemi tarkvaraarenduse protsessis asutuse RMIT näitel. Valitsusportaali (edaspidi VP) keskkondade kasutajaliideste testimine toimub manuaalselt, mis kulutab asutuses liigselt aega ja rahalist ressurssi. Probleemi lahendamiseks otsustas asutus leida alternatiivi, milleks on automaattestimine.

Töö eesmärk on leida tõhus raamistik kasutajaliidese testimiseks ning tutvustada selle võimekust RMIT-ile. Ajakulu säästmiseks rakendatakse teste CI/CD keskkonnas, kus on võimalik kõik VP keskkonnad paralleelselt testida. Saadud testide tulemused analüüsitakse ja võrreldakse asutuse manuaaltestimise kuludega. Säästetud ajaga tagatakse ka rahaline tulu, sest investeeritakse vähem aega testimisele.

Projekti tulemus näitas, et automaattestidega säästeti ressursse. Asutuse liikmed leidsid, et automatiseerimine on tulevikus kasulik, ning planeerivad CI/CD efektiivsuse tagamiseks sellega jätkata.

Lõputöö on kirjutatud Eesti keeles ning sisaldab teksti 33 leheküljel, 9 peatükki, 16 joonist, 1 tabelit.

Abstract

Automated User Interface Tests for Valitsusportaal Websites

The thesis solves an acute problem in software development process at RMIT. User interface testing for Valitsusportaal (VP) is being done manually. It consumes a lot of resources like time and money. To solve this issue, the establishment thought of using an automated solution to test all VP environments quickly and efficiently.

The objective of this thesis is to acquire the best automation framework to test the user interface and present its capabilities for the establishment. In order to save time resource, the framework will be integrated into the CI/CD pipeline, which will guarantee parallel testing capabilities. In relation to that, RMIT will save in finances. Since VP pages have very similar design, it is possible to test all pages at once with one set of tests. This avoids writing multiple tests for different environments.

The first task was to find a suitable testing framework. The initial analysis of framework focused on 4 key factors: community, valid documentation, ease of use and testing speed. Out of the 4 tools examined, it was concluded that Cypress was the most suitable framework. Its security and visual testing capabilities were demonstrated to RMIT's architects and they approved its further usage. After that, 9 test specifications were written and integrated into CI/CD pipeline. Specifications tested the homepage, contact page and the search page.

Results from testing ensured, that automated user interface testing was vibale for VP. Time was saved with the tests and RMIT saw further usage for test automation.

The thesis is in Estonian and contains 33 pages of text, 9 chapters, 16 figures, 1 table.

Lühendite ja mõistete sõnastik

API	*Application programming interface,* rakendusliides
CI/CD	*Continuous Integration/Continuous Deployment*, Pidevintegratsioon/pidevarendus
CSS	*Cascading style sheets,* kaskaadlaadistik
DOM	*Document object model*, dokumendiobjektide mudel
HTML	*HyperText Markup Language,* hüperteksti märgistuskeel
Klient	Asutus, kellele RMIT osutab teenuseid
POC	*Proof of consept,* Eksperimentaallahendus, kontseptsiooni tõendus
RK	Riigikantselei
RMIT	Rahandusministeeriumi Infotehnoloogiakeskus
ROI	*Return of investment,* Investeeringu tasuvus
SQL	*Structured Query Langauage*, struktuurpäringukeel
SVG	*Scalable Vector Graphics,* mastabeeritav vektorgraafika
TAO	Tarkvaraarenduse osakond
URL	*Uniform Resource Locator*, internetiaadress
VP	Valitsusportaal

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tarkvaaraarenduses on kasutusel konveiermeetod, mida kutsutakse CI/CD-ks (*Continuous Integration/Continuous Deployment*). Selleks, et tagada konveieri töö kõrget kvaliteeti ja hoida kokku aega, on vaja järgida selles konveieris olevaid samme. Üks nendest etappidest on testimine, mis jääb pidevintegratsiooni (CI) lõppu, ning võib pidada kvaliteedikontrolliks.

Nende parameetrite – kvaliteet ja kiirus – tagamise eest vastutab autor käesolevas bakalaureusetöös, kirjutades automaatteste riigiasutusele RMIT (Rahandusministeeriumi Infotehnoloogiakeskus) Valitsusportaalide lahendustele (edaspidi VP) .

Autori ülesandeks on Valitsusportaali veebilehe testi plaanide loomine ja testide kirjutamine. Asutuses toimuvad veebilehe kasutajaliidese testimised manuaalselt. Veebilehel olevaid funktsioonide testimist sooritatakse käsitsi, jättes kasutamata tarkvaralisi võimalusi.

Käesolev bakalaureusetöö lahendab aktuaalset probleemi tarkvaraarenduse protsessis, kus aeglane protsess asendatakse tõhusama testimisviisiga, mis tagaks nõutud tulemuse kiiremini ja vähendaks tarne välpa.

Alternatiiv manuaaltestimisele on automaattestimine. Sobiva testimisraamistiku leidmiseks koostas autor analüüsi erinevatele automaattestimise raamistikele. Analüüsi käigus valiti välja kõige optimaalsem raamistik ning integreeriti asutuse CI/CD keskkonda. Oodatav resultaat RMIT-is on ajalise ja rahalise kulude vähendamine.

Bakalaureusetöös tutvustab autor põhjalikumalt probleemi sisu (peatükk 2). Seejärel kirjeldab autor automatiseeritud raamistike analüüsi protsessi (peatükk 4). Järgnevalt seletatakse lahti testimise meetodit (peatükk 6) ja raamistiku integreerimist CI/CD keskkonda (peatükk 7). Sellele järgneb testimise järgne andmeanalüüs (peatükk 8). Lõpetuseks arvutatakse, kui palju automaattestimisega aega säästeti või kulutati, võrreldes manuaalse testimisega. Lisaks kalkuleeritakse automaattestidest tulnud rahaline väärtus (peatükk 9).

2 Taust ja probleem

Infotehnoloogilises maailmas korduvtegevused kulutavad ebamõistlikult ressursse. RMIT esitas probleemi, mis seisnes VP testimise peale minevaid kulusid kokku hoida, tagades teenuse kvaliteeti. Probleem seisnes veebilehtede uuenduste tarnimise suures ajakulus, mis on tingitud manuaalsest testimisest.

Valitsusportaal on *Drupalil* põhinev veebiplatvorm, mis täna on valdav enam valitsuse asutuste välisveebi alusplatvormina kasutusel. Eesmärk on koondada ühtse funktsionaalsuse ja visuaaliga terviklikuks [1].

VP lahendus on kasutusel standardse teenusena riigiüleselt, mis pakub digitaalselt infoedastuse teenust olulisel infokandjal. Tähtis on selle teenuse terviklik haldamine ja selle eesmärgipärasus teenuse omanikule – organisatsioonide kommunikatsiooniüksused.

Teenuse kasutajad on veebilehtede külastajad (kodanikud, juriidilised isikud ja ametiasutuste teenistujad, samuti Eestist huvitatud välismaalased).

Valitsusportaali eesmärk on tagada info leitavus ja aktuaalsus, minimeerida sama info dubleerimist ning luua riigi ja selle valitsemise tervikkuvand, mistõttu antud teenuse kvaliteedi tõstmine on teenuse pakkujale ülioluline.

Asutuses kasutatav manuaalne testimisprotsess algab arenduspartneri programmikoodi tarnimisega giti repositooriumisse. Seejärel vaadatakse kood TAO (Tarkvaraarenduse osakond) poolt üle, siis tarnitakse ja paigaldatakse arenduskeskkonda. Paigaldus kontrollitakse RMIT-i poolt üle. Pärast kontrolli pöördutakse uuesti TAO poole, et teha tarne testkeskkonda, mida kontrollib RK. Adekvaatsete tulemuste korral tehakse tarne *live* keskkonda ehk kliendile kasutatavaks.

Eelmainitud etappide tõttu jääb uue lahenduse publitseerimine klientidele suure ajakulu taha: protsessi jooksul tuleb pöörduda mitme osakonna poole. Kogu üldine tarne protsess võtab aega asutuses keskmiselt nädal või poolteist. Piiratud ressurssidega tiimis kulutab

RMIT-i projektijuht tarneprotsessi ajal keskmiselt 8 tundi testimiseks. Vigade esinemisel kulutab DevOps vea tekke põhjuseks ja leidmiseks sama aja.

Tarne välba vähendamiseks otsustas RMIT leida lahenduse automaattestide kirjutamise abil. Automatiseeritud süsteemiga välditakse liigse inimressursi kasutamist, eksimisvõimalusi ja ajakulu. Autorile esitati ülesanne läbi viia olemasolevate kasutajaliideste testimisraamistike analüüs ja valida välja nende seas kõige tõhusam variant. Hiljem tutvustab autor asutusele valitud tehnoloogia võimekust läbi kontseptsiooni tõenduse. Seejärel kirjutati valmis testide kogum, millega on võimalik ära testida kõik valitsusportaalide veebilehed paralleelselt asutuse CI/CD keskkonnas.

3 Strateegia

Liidese testimise jaoks uuris autor erinevaid allikaid automatiseeritud testimise kohta. Kogutud info põhjal loodi strateegia, kuidas võiksid VP keskkonnad üle minna manuaalselt testimiselt automatiseeritud testimisele.

3.1 Raamistiku analüüs

Kõigepealt tuleb automatiseerimise jaoks üles leida VP-le kõige optimaalsem raamistik.

1. Sobiv tööriist peab olema kogukonna poolt toetatud ja tema tööturul tööjõu nõudlus arvukas. Autor uuris Eesti tööturgu, vastava valdkonna foorumite liiklust ja tehnoloogia kasutajate hulka.
2. Koodi kirjutamine peab raamistikus olema mugav ja lihtsasti loetav.
3. Sobiv tööriist eeldab adekvaatset dokumentatsiooni, kus funktsioonid on näidete rohked ja lihtsasti mõistetavad.
4. Raamistik peab olema võimeline teste kiiresti käitama. Analüüsi käigus testisid raamistikud VP esilehel otsingu tegemist ja elemendi otsimist.

Käesolevas töös toob autor välja valitud raamistike kirjeldused ja nendevahelise võrdluse. Seejärel valib autor kõige optimaalsema variandi, millega praktilises osas jätkata.

3.2 Kontseptsiooni tõendus

Kui raamistiku analüüs on rohkem teoreetiline dokument, mis räägib raamistiku sisust, siis kontseptsiooni tõenduseks peetakse praktilist demonstratsiooni, mis näitab ära raamistiku võimalused, eripärad ja puudused. Kontseptsiooni tõenduse jaoks tuleb kirjutada lihtsamad testid (nt autentimine õigete ja valede andmetega) ja esitleda nende võimekust joosta nii *headless* keskkonnas kui ka konteineris. Kontseptsiooni tõendus on viimane etapp enne raamistiku kasutusele võtmist.

3.3 Kasutajalood

Autor dokumenteeris olulisemate funktsioonide testimiseks komponente kasutajalugudega Kasutajalugu on tööriist välearenduses, mida kasutatakse tarkvara

funktsiooni kirjelduse jäädvustamiseks kasutaja vaatenurgast. Selle eesmärk on ära kirjeldada, kuidas projekt lõppkasutajale väärtust tagab [2].

Lihtne näide kasutajaloost: „Kliendina tahan ma muuta lehe keelt, et kuvada sisu oma emakeeles (*ingl. k: „As a user, I want to change the page language, so I can view the contents of the page in my mother tongue.“*).“

Kasutajalooga saab testid ära kirjeldada, mis kontrollivad iga funktsiooni väljundit.

Näitena võib tuua VP seisukohast juhust, kui klient, kes soovib oma kasutajaandmetega ennast autentida ja selle tulemusel näha oma konto andmeid, siis tema kasutajaloo kaudu saab valmistada testi, mis üritab autentida ja kuvada kasutajaandmeid.

3.4 CI/CD

Autor oli vormistanud VP jaoks testid, peale mida integreeriti programmikood CI/CD keskkonda.

CI/CD on tarkvaraarenduses pidev meetod, mille abil ehitatakse, testitakse ja väljastatakse koodi muudatusi. See korduv protsess aitab vähendada vigade esinemist projektis, mis võivad tuleneda projekti eelmisest versioonist. Mainitud meetod aitab vähendada inimese sekkumist protsessi. CI/CD põhilised faasid on pidevintegratsioon (CI) ja pidevtarnimine (CD).

Pidevintegratsioonis rakendatakse ja testitakse muudatusi automaatselt ja pidevalt. Testid kinnitavad, et koodis läbi viidud muudatused oleksid õiged, ning oleksid vastavuses sätestatud standarditega.

Pidevtarnimine on pidevintegratsioonile järgnev etapp, kus rakendust kontrollitakse automatiseeritud testide poolt. Seejärel muudetakse rakendus klientidele kättesaadavaks. Rakenduse avalikustamine võtab aset pidevtarnimisega manuaalselt [3].

Kui testi programmikood on valmis kirjutatud, siis paigaldatakse see virtuaalsetesse konteineritesse. Igas konteineris on teatud parameetrid, mis on unikaalselt defineeritud, nt URL (*Uniform Resource Locator*), brauser, raporti sihtkaust jt. Idee on testi erineval leheküljel paralleelselt käivitada ja selle abil kõik VP keskkonnad ära testida.

3.5 Testi tulemuste dokumentatsioon

Autor kirjutab valmis pärast testi sooritust dokumentatsiooni testist saadud tulemustest. Selles osas on ära märgitud, missugused testid läbisid ja missugused põrusid. *Flaky* test on ebastabiilne test, mis võib anda valepositiivesid või valenegatiivseid tulemusi.

Autor kirjeldab ära ka ebastabiilsed testid, mis põhjustavad juhuslikke resultaate ja mida võiks nende eemaldamiseks teha. Autor toob ka välja rabedate elementide selektorid, mida võiks lähtekoodis defineeritud parameetrite vastu DOM (*Document object model*) elemendis välja vahetada. Rabedad selektorid on üldjuhul CSS (*Cascading style sheets*) klassid, id-d või omavahel kokku kleebitud CSS klassid. Rabedaid selektoreid ei tasu kasutada, sest liidese lähtekood võib muutuda ja seal olevad CSS klassid ei pruugi säilida või on neil kalduvus teiseneda. Selle tulemusel lähevad automaattestid tihitipeale katki ning tekib vajadus neid ümber kirjutada.

Kui testi tulemuste dokument on valmis kirjutatud, siis on tulevikus teistel VP arendajatel baasdokument, kus on olemas VP testimisel oluline info.

3.6 Tulemused

Automaattestimise tulemused võrreldakse RMIT-is kasutuses oleva manuaaltestimisega. Põhiliselt mõõdetakse aja- ja finantsilise kulude kokkuhoidu. Kõigepealt arvutatakse automaattestimise kiirus VP liidese testimisel. Kui automatiseerimisele kulub vähem aega kui manuaaltestimisele, siis arvutatakse välja asutuses alles jäänud finantsilised investeeringud.

Pikaajalist efektiivsust saab mõõta investeeringu tasuvust arvutades [10]. Selle tulemuse arvelt saab asutus veenduda, kas uue testimismeetodi arendusse tasub pikaajaliselt investeerida.

4 Raamistiku analüüs

Käesolevas töös uuris autor põhjalikumalt nelja raamistikku: Cypress [7], WebdriverIO [6], TestCafe [5] ja Nightwatch.js [4]. Autor testis ka populaarsemaid raamistikke nagu Jest, Mocha, Jasmine ja Robot Framework, kuid leidis, et need ei olnud kõige paremad kasutajaliidese testimiseks, sest need on mõeldud rohkem ühiktestide kirjutamise jaoks.

Praktilise analüüsi poolelt kirjutas autor kaks testi Valitsusportaali valitsus.ee veebilehe näitel, kus testiti otsingu sisestamist ja veebilehe päises elemendi olemasolu.

Autor hindas raamistike väärtusi vastavalt neljale punktile, mis on välja toodud peatükk 3, jaotis 3.1.

4.1 Nightwatch.js

Nightwatch.js on automatiseeritud kasutajaliidese testimisraamistik veebilehtedele. See on kirjutatud Node.js-is ja kasutab W3C Webdriver-i API-t (*Application programming interface*) erinevate veebisirvijatega suhtlemiseks. Tegemist on lõpliku lahendusega, mille eesmärk on lihtsustada testide kirjutamist ning pidevintegratsiooni ja pidevvalmidust. Nightwatch-i saab kasutada liidese-, ühik- ja integratsioonitestide kirjutamiseks [4].

Analüüsitavas raamistikus avastas autor, et Nightwatch.js-i dokumentatsioonis on informatsioon puudulik. Osadel funktsioonidel puudusid näited, mis muutis funktsioonide rakendamise mõistmist ja nende kasutamist keeruliseks.

Juhendis jääb arusaamatuks ka raamistiku ülesseadmine. Instruktsioonis on välja toodud viisid, kuidas automaatselt ja manuaalselt raamistiku sätteid konfigureerida. Autor leidis, et automaatsel konfigureerimisel tekib üks sätestamise fail, aga üks jääb kaduma. Seetõttu kasutas autor manuaalse konfigureerimise juhendit, kus tuleb failide sisu kopeerida projektis loodud failidesse. Kokkuvõttes võiks olla raamistikul töötav automaatne konfiguratsioonide failide genereerimise viis. Igasugune manuaalne tegevus võiks olla eemaldatud, kuna see tekitas raamistiku ülesseadmisel ebamugavusi.

Raamistiku eelis on tema testide käitamise kiirus ja lihtsasti arusaadav süntaks. Autori projektis jooksid testid kõige kiiremini võrreldes teiste raamistikega.

Nighwatch.js-i puuduseks jääb tema kogukonna toetus. Erinevate juhtumite lahendamine projektis jäid lahtiseks, kuna raamistikul puudub piisav arv kasutajaid, kes panustaksid. Seisuga 31.08.2021 oli Stackoverflow foorumi andmete järgi Nightwatch.js-il 106 teemat. Tööturul puudus igasugune nõue sellele raamistikule.

4.2 TestCafe

Testcafe on avatud lähtekoodiga kasutajaliidese testimise raamistik, mis jookseb Node.js-i kasutades. Sellega töötamisel kohaneb kiiresti: testid on lihtsasti loetavad isegi javascripti oskuse puudumisel [5].

Testide kirjutamisel osutusid raamisiku arendajate poolt loodud juhendid väga suureks abiks. Dokumentatsioon on selgelt lahti kirjutatud ning funktsioonid esinevad näidetega.

Keskkonna sätestamine oli vaevatu. Paari käsuga on võimalik keskkond sätestada ja testide kirjutamisega alustada. Testide kirjutamine toimis vähese vaevaga, kuid autor leiab, et oleks võinud olla mugavam HTML (*HyperText Markup Language*) elementide valimise meetod.

Projektis katsetatavate testide kiirus oli keskmiselt 10 sekundit, mis on võrreldes teiste raamistikega aeglane.

Raamistiku kogukond eksisteerib väikestes arvudes nii kasutajate kui ka tööpakkumiste hulga poolest. Autorile osutus nende probleemide lahendamine keeruliseks, millele dokumentatsioon ei suutnud lahendust leida.

4.3 WebdriverIO

Raamistik on ehitatud kaasaegsete veebilehtede ja mobiilsete rakenduste automaatseks testimiseks. See võimaldab mitmes veebilehitsejas testida samaaegselt ehk sünkroonselt, abifunktsioonide kirjutamist lihtsamate lahenduste loomiseks. Raamistik kui projekt on avatud lähtekoodiga.

Raamistik on iseärasuste rohke ehk talle on loodud kogukonna poolt palju teeke. WebdriverIO arendajad toetavad rahvarohket kogukonda [6].

Dokumentatsioonis olid erinevad funktsioonid lahti seletatud ja kasutusvõimalused olid välja toodud näidetega. Seega ei olnud autoril keeruline aru saada funktsioonide iseärasustest ega paindlikkusest.

Raamistiku ülesseadmisel ei esinenud probleeme. Pärast paari käsu sisestamist on võimalik arendajal testide kirjutamist alustada. Installeerimisjärgselt tutvustab WebdriverIO kasutajale *Page Object Model* (POM) mustrit, mis on väga sarnane objektorienteeritud programmeerimisele. POM-is on jaotatud erinevad testimisfunktsioonid eraldi klassidesse, mis kutsutakse välja testijuhtumites. Muster vähendab koodi kordusi ja annab lihtsamini loetava kuju.

Autor leidis, et konsoolis olev informatsioon annab hea ülevaate. Kui testis leidub viga, siis on tänu konsoolis esinevale informatsioonile selle parandamine lihtne.

WebdriverIO eeliseks on tema kiire testide kompileerimis- ja käitamiskiirus. Autori projektis läbisid lihtsamad testid alla viie sekundi.

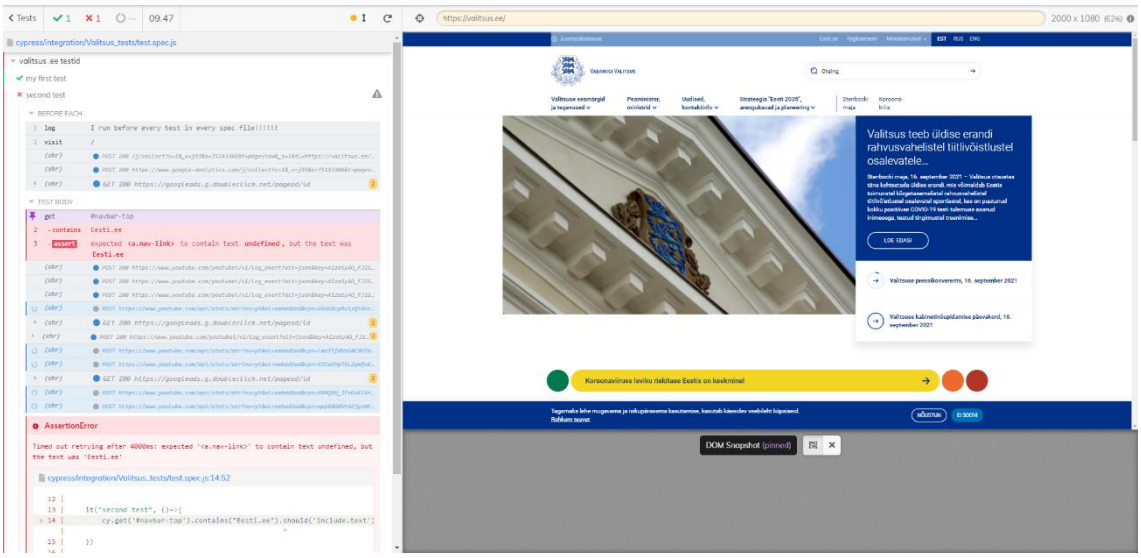
WebdriverIO-l on suur kogukond Gitteri keskkonnas, kus raamistiku kasutajad saavad elavas jututoas arutada erinevate probleemide üle, mis esinevad raamistiku kasutamisel. Kasutajate arvud ei ole kõige paremad populaarsemates foorumites nagu Stackoverflow. Tööpakkumiste arvukus Eestis on peaaegu nullilähedane.

4.4 Cypress

Cypress on järgmise generatsiooni avatud lähtekoodiga eesliidese testimise tööriist ehitatud kaasaegsete veebide testimise jaoks. Cypressiga saab kirjutada ühik-, integratsiooni- ja kasutajaliidese teste. Cypressi arendajad on loonud funktsioone testimise lihtsustamiseks: salvestada videoid ja pilte testide tulemustest, paralleeltestimine [7].

Analüüsi käigus leidis autor, et juhendid on informatiivsed ja annavad hea ülevaate erinevatest funktsioonidest, kus iga funktsioonile on lisaks seletustele toodud välja ka erinevaid näiteid raamistiku kasutamiseks. Cypressi näidete rohke ja terviklik juhend lihtsustas autoril praktilises testimises oma probleemidele lahendusi leida.

Autoril ei tulnud ette ebamugavusi raamistiku installeerimisel. Rakendamise juhend oli konkreetne. Cypressil on ka mugav süntaks testide kirjutamiseks, kus lihtsamad testid on loetavad isegi isikule, kes on vähe arendamise või testimisega kokku puutunud. Mugavust andis ka juurde raamistiku enda klient, kus kasutajal on mugav jälgida, mida kasutajaliideses testitakse ja mis võis olla läbikukkuva testi põhjus (Joonis 1).



Joonis 1. Cypressi testide käitamiseliides - *test runner*

Testid jooksid raamistikus kiiresti, kuid kompileerimise kiirus jäi võrreldes teiste raamistikega suhteliselt aeglaseks, sest testid laetakse veebilehte.

Cypressi üks suurimaid eeliseid on tema rahvarohke ja vastutulev kogukond. Töötamise käigus leidis autor erinevaid foorumi postitusi, mis aitasid teatud äärmiste juhtumite lahendamisega, mis ei leidunud raamistiku dokumentatsioonis. Lisaks suurele kogukonnale on raamistikul ka suur tööpakkumiste arv Eestis. Autor analüüsis LinkedIn CV keskust, mille käigus avastas, et Cypressil oli 31.08.2021 17 tööpakkumist.

Cypressil esinevad vead, mis võivad testide arendamisel tekitada ebamugavusi. Raamistik ei luba kasutada mitmeid veebilehitseja sakke. Liikudes ühelt veebilehelt teisele (nt facebook.com-ilt google.com-i veebilehele) on keeruline, sest Cypress ei toeta erinevate allikate vahelist ümbersuunamist, kui nende port, protokoll ja superdomeeni host on erinevad. Probleemi saab vältida, kirjutades erinevad allikad eraldi testidesse, kuid autor leiab, et see võib muuta testimise protsessi liiga keeruliseks ja ebamugavaks.

4.5 Võrdlustabel

Allpool on välja toodud raamistike vaheline võrdlustabel tabel 1.

Tabel 1. Raamistike võrdlustabel

Väärtus	**Parameeter**	**Cypress**	**WebdriverIO**	**TestCafe**	**Nightwatch.js**
Juhendite terviklikkus	Informatiivsus (näited)	Piisav	Piisav	Piisav	Ebapiisav
Arendaja mugavus	Installeerimine	Lihtne	Lihtne	Lihtne	Keeruline
	Konfigureerimine	Lihtne	Lihtne	Lihtne	Ebamugav
	Kirjutamismugavus	Mugav	Mugav	Mugav	Mugav
Võimekus	kiirus	4 s	3 s	9 s	5 s
Kogukond seisuga 31.08.2021	Githubi kasutajate arv	257 000	34 600	9000	127 000
	NPMJS allalaadimiste arv nädalas 16.08 – 22.08.2021	2 565 000	861 000	243 000	183 000
	Stackoverflow foorumi teemade arv	4973	1361	1479	106
	LinkedIn tööpakkumised Eestis	17	1	1	0

4.6 Analüüsi kokkuvõte

Autori hinnangul osutus kõige paremaks testimisraamistikuks Cypress. Esmahinnangul oli Cypressi väga lihtne üles seadistada, sest puudus igasugune liigne manuaalne konfigureerimine. Testide kirjutamiseks on loodud kasutajasõbralik süsteem, kus Cypressil on oma kasutajaliides nimega *Test Runner*, millega saab ära näha testis kõik toimuvad protsessid. *Test Runneris* on kuvatud testitav veebileht, kus näeb ära kõik testi skripti tulemused. Lisaks annab *Test Runner* põhjaliku tagasiside testi ebaõnnestumisel. Cypressil on olemas kogukas kogukond teiste raamistikega võrreldes. Suure kasutajaskonna abiga on võimalik projekti kirjutamisel ette juhtuvaid probleeme kiiremini lahendada. Lisaks leidis autor, et Cypressil on terviklik dokumentatsioon koos põhjalike seletustega ja näidetega.

Põhjused, miks teised raamistikud ei osutunud valituks, olid enamasti probleemid dokumentatsioonis ja väike kogukond. Dokumentatsioon on raamistiku üks tähtis osa ning mitteterviklik dokumentatsioon muudab keeruliseks funktsioonide arusaamise ja probleemide tekkimise põhjused. Kolmanda osapoole abi on nõrk, kuna nendel raamistikel puudub populaarsus. Vähese töönõudluse tõttu ei ole mõistlik valida raamistiku, mis ei ole tööturul nõutud.

5 Kontseptsiooni tõendus

Enne raamistiku täielikku kasutuselevõttu nõuti ka autorilt kontseptsiooni tõenduse loomist. See annab kinnituse, et autori poolt valitud raamistikuga on võimalik VP keskkonnad ära testida. Selleks palus RMIT-i arhitektuurinõukogu, et autor testiks valitsuse arenduskeskkonnas ära järgmised testitavad juhud:

- Esilehe CSS-i olemasuolu kontroll
- Sisselogimine õigete kasutajaandmetega
- Autentimine valede kasutajaandmetega

5.1 Liidese disaini testimine

Asutus tõi esile VP esilehe disaini kontrolli, kuna seal on ette tulnud juhtumeid, kus CSS ei lae veebilehel ära. Selle olemasolu kontrollimiseks on 2 võimalust: funktsionaalne ja visuaalne testimine.

Autor otsustas funktsionaalset testimist mitte rakendada. Selle kontrollimisviisi all mõeldakse skriptide kirjutamist raamistikus, mis kontrollivad DOM-i id-de ja klasside disaini atribuute, mille puudumisel visatakse ette veateade.

Funktsionaalselt testimisel esinevad puudused testide haldamises, sest veebi disaini pidevalt muudetakse ning osade atribuutide väärtused võivad muutuda, ära kaduda või juurde tekkida. Lisaks muutuvad skriptid funktsionaalse testimisega liiga pikaks ja keeruliselt loetavaks [8].

Alternatiiv funktsionaalsele lähenemisele on kasutada visuaaltestimist, mille kasuks ka autor otsustas. Selle kontrollimisviisi põhimõte on teha lehest pilt, mida saab testi baaspildiga võrrelda. Visuaaltestimist on võimalik rakendada kahte moodi:

1. Kolmanda osapoole tööriista kasutamine
2. Kogukonna poolt loodud teegi või pistikprogrammi abil

Kolmanda osapoole tööriistaks valis autor Applitoolsi. See on pilveteenus, mis tagab palju paindlikkust CSS-i testimisel. Tööriistaga saab DOM momentvõtteid luua, mille sisuks on testitava vaate ressursid (CSS, SVG, HTML jne), mis saadetakse edasi

Applitoolsi pilve. Seal töödeldakse hetkvõte vastavalt testija soovile. Kui arendaja tahab momentvõtteid erinevates suurustes ja erinevatele brauseritele teha, siis oskab Applitoolsi pilv vastavalt arendaja loodud programmikoodi abiga need hetkvõtted luua. Baaspiltide abil on võimalik võrrelda testis loodud pilte omavahel ja tulemus edastatakse pilveteenuse liidesesse, kus on võimalik tulemusi kontrollida, tagasi lükata või meelde jätta, et vältida tulevikus kõrvalekaldeid. Applitools pakub veel funktsionaalsusi teatud DOM elementide vältimiseks, kui rakenduses peaksid esinema andmeväljad, mis tihti muutuvad (nt kellaaeg, reklaamid jne). Selle jaoks saab arendaja kirjutada skripte, mis ignoreerivad DOM elemente ja kontrollivad ainult vajalikke välju.

Kuigi pilveteenus pakub palju kasulikke funktsionaalsusi VP liidese testimiseks, siis pidi autor selle kasutamist järgnevate põhjuste pärast vältima:

1. Hetkvõtted võivad sisaldada konfidentsiaalset sisu, sest testitava vaate ressursid saadetakse kolmanda osapoole pilve, mis pole asutusega seotud. Selle tulemusel leidis asutus, et võib tekkida ohtlikke andmelekkeid.
2. Asutuse CI/CD keskkond jookseb piirangutega serveris, kus ei looda väliseid ühendusi. Seetõttu pole asutuse nõuetekohaselt võimalik Applitoolsi pilveteenusega ühendust luua

Visuaaltestimist jätkas autor, kasutades pistikprogrammi Cypress Image Snapshot. Pistikprogrammidel tuleb jälgida, et nad ei põhineks raamistiku varasematel versioonidel, kuna aegunud pistiku kasutamine võib vigu kaasa tuua. Selle kontseptsiooni tõenduse puhul otsis autor pistikut, mis võimaldab erineva suurustega pilte vaatetest teha erinevatele brauseritele. Peale selle omab pistik funktsiooni DOM elementide peitmiseks, sest nende sisu võib erinevates testimistes muutlik olla.

Cypress Image Snapshot täitis autoripoolsed nõuded. Pistikuga sai skripti kirjutada, mis võimaldas teha nii töölaua suurusest kui ka mobiilvaatest hetkvõtteid kasutatavates brauserites (Google Chrome, Microsoft Edge, Mozilla FireFox), mida autor kasutas veebilehe testimisel.

Hetkvõtteid tasub luua vastavalt brauserile, sest igal veebisirvija omab iseärasusi. Eriti leidub erinevusi teksti kuvamisel. Näiteks, kui luua hetkvõte Google Chrome'i sirvijas, siis suure tõenäosusega kukub test läbi, võrreldes Internet Explorerile kuuluvale

baaspilidga. Põhjuseks on fondi kuvamise erinevused. Programm tuvastab, et pildid ei klapi omavahel ja test kukub läbi.

Probleemid tekivad mitme juhtumi käivitamisel Cypressiga, sest raamistik ei võimalda jooksutada ühes käitusajas (ingl. k *runtime*) mitut veebisirvijat korraga. Seda probleemi annab vältida, kirjutades Dockeri skripti, kus saab ära defineerida konteineid ning. neid hiljem kasutada testimises paralleelselt, mis loovad vastavad tulemused erinevatele brauseritele.

5.2 Autoriseerimise testimine

Autor testis VP-s sisselogimist õigete ja valede kasutajaandmetega funktsionaalselt, kirjutades kokku kuus testi.

5.2.1 Õigete andmetega autentimise näide

Esimene test katsetab õigete kasutajaandmetega sisselogimist, mille tulemuseks on kasutaja kontolehe kuvamine, ja veebiaadressis on näha kasutaja id-d. Kasutaja sisselogimise funktsiooni on kujutatud joonisel 2. Funktsioon kasutab kahte parameetrit: kasutajanimi ja parool. Funktsioon logib sisse nuppu vajutades. Funktsioonile järgneb testi juhtum, mis rakendab eelnevat funktsiooni. See sisestab Cypressi konfiguratsiooni failis defineeritud autentimise muutujad – kasutajanimi ja parool – vastavatesse väljadesse ning teostab sisselogimist. Järgmisena kontrollib test sisselogimise edukust, kontrollides kasutaja id olemasolu internetiaadressi sisus.

```
Cypress.Commands.add('login', (username, password)=>{
    cy.get('[data-drupal-selector=edit-name]')
      .type(username,{log:false});
    cy.get('[data-drupal-selector=edit-pass]')
      .type(password,{log:false});
    cy.get('#edit-submit').click();
})

//test.spec.js
//tests the login with the login function and
//asserts that the endpoint is valid
it('should log in with valid credentials', function () {

cy.login(Cypress.env('username'),Cypress.env('password'))
    cy.url().should('include','/user/'+Cypress.env('id'))
```

Joonis 2. Sisselogimise testi kood

Autor avastas nüansi autentimise testimises: Valitsusportaali kasutaja saab luua maksimaalselt kaks sisselogimise sessiooni. Kolmanda sessiooni loomine osutub võimatuks ja autentimise lehel tekib veateade. See põhjustab probleemi aktiivsete sessioonide veateate olemasolu testimist ja kasutaja sisselogimist. Kui aktiivseid sessioone on kokku kaks, kukub sisselogimise test läbi, aga veateate test läbib. Kui testida sisselogimist ja seejärel testida aktiivsete sessioonide olemasolu, siis test põrub väheste sessioonide puudumise tõttu. Arendaja ei saa aktiivseid sessioone mõjutada, sest need ligipääsud jäävad serveri poolele. Autor tõi selle probleemi välja asutuse arhitektidele. Võimalikud arendused on kaalumisel, kus testide arendaja saab vanu sessioone eemaldada.

5.2.2 Valeandmetega autentimise näide

Järgmised kaks testi (Joonis 3) kirjeldavad valeandmetega sisselogimist. Esimeses testis üritab test sisse logida juhusliku kasutajanime ja parooliga. Tulemuseks peab olema veateade. Teine test on idee poolest sarnane, kuid kasutajanimi on õige ja parool vale. Tulemuseks on sama veateade, mis valede andmetega autentimisel.

```
Cypress.Commands.add('errorLogin', () => {
    cy.get('.alert').should('include.text','Tekkis viga:')
    cy.get('.form-item--error-message')
        .should('include.text','Tundmatu kasutajanimi või
parool. ')
})

it('should fail on login with invalid credentials', function ()
{
    cy.login('user','pswd');
    cy.errorLogin();
});
it('should fail on login with invalid password', function () {
    cy.login(Cypress.env('username'),'pswd');
    cy.errorLogin();
});
```

Joonis 3. Valede andmetega autentimine

Neljas test (Joonis 4) kujutas endast sisendväljade testimist, kus tühjade väljade korral, peab esinema JavaScripti valideerimismärkus kasutajale, mis nõuab sisendväljade

täitmist. Kui väli ei sisalda teksti, siis esineb ingliskeelne märkus: „*Please fill out this field*“ (eesti k: „Palun, täitke see väli.“). Kui väli sisaldab andmeid, siis hoiatust ei esine.

```
Cypress.Commands.add('checkValidation', (param)=>{
    cy.get('[data-drupal-selector=edit-'+param+']')
      .invoke('prop', 'validationMessage')
       .should('eq','Please fill out this field.')
    cy.get('[data-drupal-selector=edit-
'+param+']').type('testField')
    cy.get('[data-drupal-selector=edit-'+param+']')
      .invoke('prop', 'validationMessage').should('eq','');
});

it('should trigger a validation message on empty fields',
function (){
    cy.get('[data-drupal-selector="user-login-
form"]').within(()=>{
        const variables = ['name', 'pass']
        variables.forEach(value => cy.checkValidation(value))
    });
});
```

Joonis 4. Sisendite valideerimine

Samas testiga tekkis ka olukord, kus autor käitas testi Cypressi kliendis ja valideerimisväljad olid inglise keeles. Sellest lähtuvalt oli autor kirjutanud valideerimismärkuse inglise keeles. *Headless*’ilt, ehk käsurealt käivitades töötasid brauserid eesti keeles, mis tekitasid *false positive* ehk väärkinnituse. Väärkinnitus tähendab seda, et test leidis rakenduses vea, mis tegelikult ei ole defekt. Selle olukorra lahendamiseks otsustas autor lähtekoodi konfigureerida, andes igale brauserile ühe keele, mida käitamisel kasutada.

5.2.3 SQL süstimise näide

Autor pidas ka oluliseks võimalikke SQL (*Structured Query Langauage*) süstimise ohtusid testida (joonis 5). SQL süstimine tähendab seda, et väljadesse sisestatakse SQL päringuid ja selle tulemusel on võimalik vääral kasutajal saada ligipääs andmetele või funktsionaalsustele, mis ei kuulu talle ja ohtu seada asutuse andmete konfidentsiaalsuse.

SQL süstimist saab sooritada autentimisväljadesse SQL päringuid sisestades, mille tulemusel juhtub tavalise sisselogimise või veateate esinemise asemel midagi muud (nt väär kasutaja saab sisse logida ennast administraatorina). Autor testis seda sisestades SQL

süstimise sõnesid järjest autentimise väljadesse. Kui autentimise tagajärjel on lehe vastus viga koodiga 500, siis on keskkonnas potentsiaalne süstimise oht. Kui sama viga ei esine, siis süstimine ei töötanud.

```
it('should not throw error 50X on random symbol input', function
() {
    const injectSymbols = [
        '\'or 1=1--+',
        '\' or \'1\'=\'1' ,'1\' or 1=1-- -',
        '--',
        '#',
        'admin\'--',
        '/*Comment Here*/',
        '/*!32302 10*/',
        ';']
    injectSymbols.forEach(value => {
        cy.login(value, value)
        cy.url().should('include', '/user/login');
        cy.visit('/user');
    });
});
```

Joonis 5. SQL süstimise test

Probleem, mis võib tekkida mitmete päringute sisestamisel, on kasutaja ajutine keeld sisselogimiseks. See võib põhjustada olukordi, kus autentimise testid annavad valepositiivse tulemuse.

Viiendaks testis autor pikkade sõnede sisestamist väljadesse kontrollides (Joonis 6.), kas sõned ületavad lubatud väljade tähtede arvu. Kasutajanime maksimaalne lubatud pikkus on kuuskümmend ja paroolil sada kakskümmend kaheksa ühikut. Test üritab väljadesse kirjutada lubatust pikemaid sõne. Kui test leiab, et tekst on lubatust pikem, siis esineb veateade. Test õnnestub juhul, kui trükimärkide arv ei ületa väljas sätestatud tähemärkide arvu.

```
Cypress.Commands.add('checkInputLength', (param, value,
length)=>{
    cy.get(param).type(value).invoke('val')
      .should('have.length', length);
    cy.get(param)
})

it('should not exceed character limit in input', function () {
    cy.checkInputLength('[data-drupal-selector=edit-name]',
Cypress.env('longUsername'), 60)
    cy.checkInputLength('[data-drupal-selector=edit-pass]',
Cypress.env('longPassword'), 128)
});
```

Joonis 6. Sisendipikkuse test

5.3 Cypressi käivitamine konteineris

CI/CD keskkonna jaoks katsetas autor oma loodud Cypressi skripte Dockeri konteineris ja Gitlabis, et demonstreerida Cypressi võimekust konteineri käivitamisel ja võimalusi paralleeltestimiseks.

Esimene samm oli Cypressi testimine Dockeri konteineris (Joonis 7). Selleks kirjutas autor skripti, mis kasutab Cypressi *image*'it (edaspidi pilt). Pildis on Cypress, brauserid ja operatsioonisüsteem alla laetud. Konteineris on defineeritud muutujad. Üks muutuja loob ühenduse arenduskeskonnaga, mis sisaldab proksiühenduse aadressi. Teine muutuja sisaldab kasutaja autentimiseks olulist parooli testitavas lehes. Kasutajanimi on defineeritud Cypressi konfiguratsiooni kaustas, kuid on võimalik lisada konteineri muutujate hulka. Parool, proksi ja veebiaadress on joonisel 7. muudetud, et hoida asutusega seotud konfidentsiaalne info varjatuna.

```
services:
  vp-dev:
    image: "cypress/included:8.6.0"
    environment:
      - CYPRESS_baseUrl=dev.vportal
      - HTTP_PROXY=someproxy
            - CYPRESS_password=fakepass
    working_dir: /e2e
    command: "--browser chrome"
    volumes:
      - ./:/e2e
```

Joonis 7. Dockeri skript

Tulemuseks on Dockeri konteiner, kus jookseb autori loodud test VP arenduskeskkonna jaoks.

Järgmisena proovis autor käivitada mitut konteinerit korraga. Selle jaoks tuli ära defineerida *job*-id (edaspidi töö), mis loovad konteinerid. Käsuploki parameetri (ingl. k c*ommands*) all defineeris autor iga töö jaoks erinevad internetiaadressid, kus üks peab testima arenduskeskkonda ja teine testimiskeskkonda.

Tulemusena testisid konteinerid ära mõlemad keskkonnad. Kuna VP disain on igal keskkonnal väga väikeste varieeruvustega, siis põhifunktsioonide paralleelselt testimine oli nendel lehtedel edukas ning see kindlustas enamgi Cypressi edasist kasutamist nii VP testimiseks kui ka rakendamist CI/CD keskkonnas.

6 Testide kirjutamine Valitsusportaali keskkondadele

Autor kirjeldab antud peatükis testide planeerimist ja kirjutamist VP-le, testide struktuuri. Peale nende tuuakse välja testjuhtumite sisu lühikirjeldus ja testi tulemuste väljundid, mis kajastuvad testidest genereeritud raportites.

6.1 Testide planeerimine

Autorile anti ülesandeks testida VP esileht, kontaktide ja otsingu tulemuste leht. Esilehe testimiseks kasutas autor VP 2.0 stiiliraamatut [9], kus on toodud välja esilehe kohustuslikud komponendid: 0 taseme menüü, päis, e-teenused, infoplokk 1 ja 3, kaasamine ja jalus. Kuna need komponendid on erinevatel VP lehtedel funktsionaalsuse ja stiili poolest samasugused, siis teoorias peaks olema võimalik igale komponendile kirjutada üks spetsifikatsioon, mis on võimeline testima ära kõik VP esilehed. Sama on ka kontaktide ja otsingu lehtedega, kus liidese disain ja funktsionaalsused on kõigil VP juhtumitel sama.

Kohustuslikud osad tuleb testida vastavalt stiiliraamatus esitatud tingimustele. Raamatus puuduvad kontaktlehe kohustuslikke elementide loetelu ja komponentide testimise täpsed nõuded. Puuduvate nõuete puhul, arutas autor VP projektijuhiga, millised testid igal komponendil on vajalikud. Lepiti kokku, et stiiliraamatu põhjal tuleb valmis kirjutada eraldi testimisel põhinev dokument, mis koosneb komponentide kasutajalugudest. Samasse dokumenti saab lisada kasutajalood nende komponentide testidest, mis ei ole kirjeldatud VP stiiliraamatus. Teostatav dokument lihtsustab kohustuslike komponentide testimise tulevastele testi arendajate jaoks.

6.2 Testide struktuur

Cypressis on testid jaotatud 9 spetsifikatsiooni faili, mis koosnevad omakorda testijuhtumitest. Spetsifikatsiooni fail on kogumik testijuhtumeid, mis sisaldavad testimisega seotud funktsioone. Kodulehele kuuluvad 7 spetsifikatsiooni faili on pandud ühte kasuta nimega Homepage. Failid on nummerdatud ühest seitsmeni, alustades 0-taseme menüüst ning lõpetades jalusega. Siis on tagatud komponentide järjekord, kus testide spetsifikatsioone on arendajal hiljem lihtsam lugeda. Kontaktlehe ja otsingulehe spetsifikatsioonid on kirjutatud eraldi.

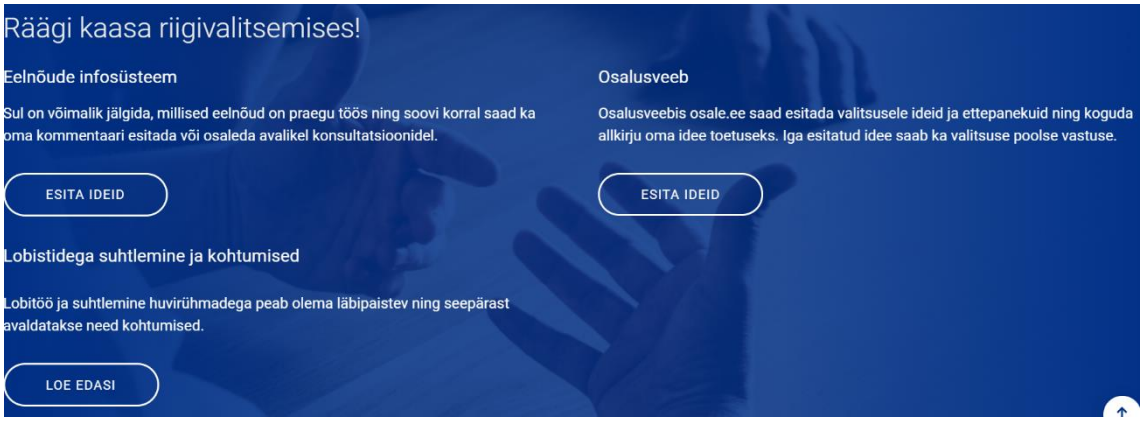
Testid on kirjutatud funktsioonidesse koodi korduste vältimiseks, mis lihtsustab spetsifikatsioonide sisu loetavust.

6.3 Põhitestid

VP testimisel määrati autorile põhilised faktorid: sisu kontroll, piltide olemasolu, töötavad lingid, funktsioonid. Igal faktoril on omale vastavad kasutajalood ja testid.

6.3.1 Sisu kontroll

Teksti sisaldavatel komponenditel tuleb kontrollida sisu olemasolu. Näiteks kodulehe kaasamise plokis (Joonis 8) on näha lõike koos pealkirja ja tekstiga.



Joonis 8. Kaasamine

Seda testib autor testjuhu kirjutades (Joonis 9), mis võtab kaasamisplokist teksti sisuga DOM elemendid ja kontrollib, et plokis olev tekst ei oleks võrdeline nulliga.

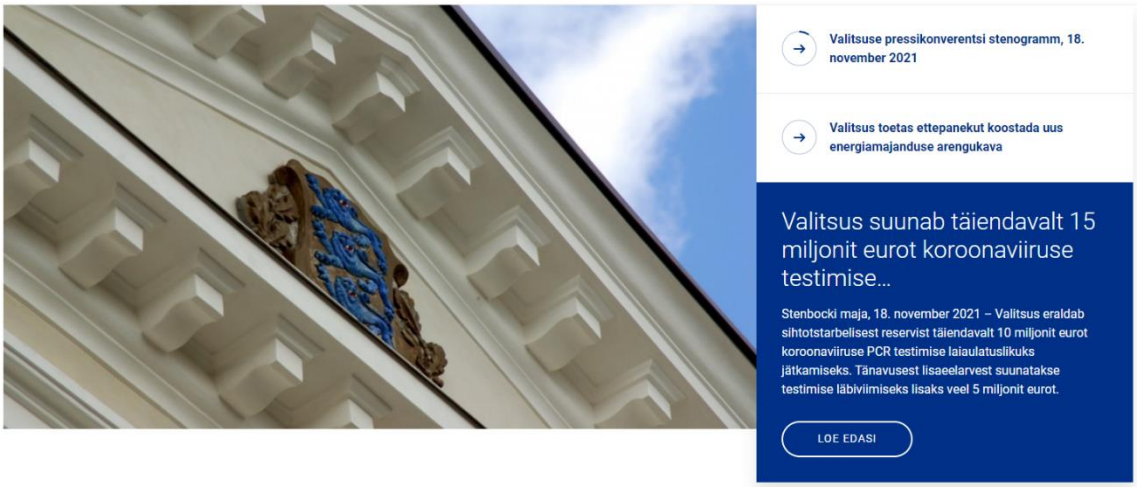
```
it('subjects should have text content', function () {
    cy.get('.pre-footer-vp').find('.card-text')
        .each((paragraph, num, list) => {
            expect(paragraph.text().length).to.be.above(0)
        })
});
```

Joonis 9. Lõigu sisu test

Tavaline sisukontrolli kasutajalugu näeb välja järgmine: „Külastajana tahan ma lugeda kaasamises sisu, et olla teemaga kursis.“

6.3.2 Piltide olemasolu

Osad komponendid nagu uudiste infoplokk ja kaasamine sisaldavad erinevaid pilte. Komponendis Infoplokk 1 (Joonis 10) on näha pilti uudisest, mis on parajasti aktiivne.



Joonis 10. VP uudiste infoplokk

Pildi täpset sisu on keeruline testida, sest nagu ka teksti sisu kontrollimisel võib sisu olla erinev. Selle asemel saab kontrollida, kas pilt on komponendis kuvatud või mitte. Pildi kontrollimiseks teeb autor API kutse, mille sisuks on DOM elemendist saadud pildi URL. Vastuseks peab olema kood 200, mis näitab, et pilt on laetud. Ebaõnnestumise korral vastab API veateatega, mille kood on 404, ja test põrub.

```
it('should have news image', function () {
    cy.get('.swiper-slide-active > .swiper-slide-lg-vp__image')
        .should('have.css', 'background-image')
    cy.getElementCollection('.swiper-slide-lg-vp__image',
'style')
});
```

Joonis 11. Pildi olemasolu test

VP pildi testimise tegi keeruliseks URL-ide erinev määramine atribuutidesse. Osad URL atribuudid sisaldasid lõppsõlme, mida oli lihtne kasutada API päringu tegemiseks, kuid mõned URL-id olid pandud *styles* atribuudi alla, mis sisaldas lisaks veebiaadressile ka erinevaid sümboleid. Selle erijuhu jaoks tuli kirjutada funktsioon, mis lõikab sõnest vajaliku URL-i välja ning kasutab seda pildi olemasolu testimiseks.

Autor kirjutas kasutajaloo pildi olemasolu testimiseks järgnevalt: „Külastajana tahan ma kuvada uudisest pilti, et saaksin tutvuda visuaalse sisuga.“

6.3.3 Töötavad lingid

Töötavate linkide testimine on sarnane jaotises 8.3.2 kirjeldatuga, sest testitavad lingid kasutavad samasugust Cypressi funktsiooni, mida kasutatakse joonisel 11.

Linkide testimise kasutajalugu on järgmine: „Külastajana tahan ma uudist edasi lugeda, et tutvuda loo detailidega.“

6.3.4 Funktsioonid

Erinevalt sisu, piltide ja linkide testimisest pidi autor erinevate funktsionaalsuste jaoks mitu funktsiooni kirjutama.

0-taseme menüü testimisel on näha, et VP kasutab erinevate keelte kasutamisvõimalust, milleks on eesti, inglis-, ja vene keel.

Joonis 12. VP 0-taseme menüü

Selle testimiseks paneb autor testjuhud tsüklisse, mis kasutab objekte sisaldavat järjestikku. Objekt sisaldab kolme parameetrit: kolmetäheline lühend keelest, elemendi sisu ja lõppsõlme. Tsüklis käiakse läbi kõik objektid, kus lehe keel muudetakse vastavalt keele lühendi väärtusele. Seejärel kontrollitakse, et lehe sisu oleks vastavas keeles ja lõppsõlm kohane. Joonis 13 esitab keele testi.

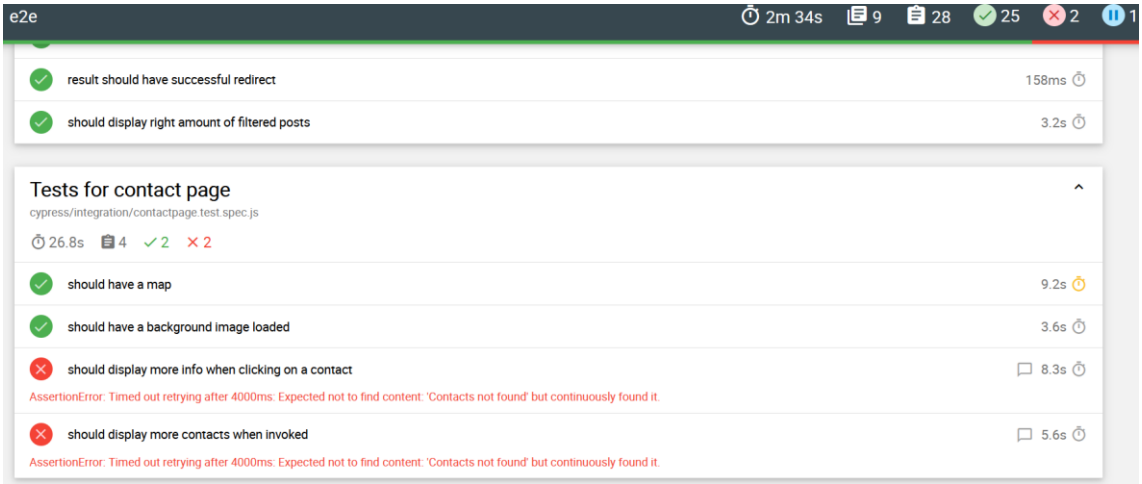
```
const i18n = [
    {lang: 'RUS', result: 'Доступность', endpoint: 'ru'},
    {lang: 'ENG', result: 'Accessibility', endpoint: 'en'},
    {lang: 'EST', result: 'Juurdepääsetavus', endpoint: ''}
]

i18n.forEach(item => {
    it('should be able to change language to ' + item.lang, function
() {
        cy.assertLanguage(item.lang, item.result, item.endpoint)
    });
})
```

Joonis 13. Keele test

Funktsionaalse testimise kasutajalood erinevad üksteisest, sest funktsioonid täidavad varieeruvaid ülesandeid. 0-taseme menüü keelemuutmise kasutajalugu on järgmine: „Eestikeelse/inglisekeelse/venekeelse külastajana tahan ma muuta esilehe keelt, et kuvada infot emakeeles.“

6.4 Raport

Pärast testi käitamist genereeritakse raport, mis on kujutatud joonisel 14. Raport on HTML dokument, mis kuvab testi tulemused. Resultaati saab kasutada analüüsiks, et tuvastada, millised testid õnnestusid või põrusid. Analüüsis saab hiljem välja tuua, millised testid olid vigased ehk milliste testide tulemused olid kas valepositiivsed või valenegatiivsed. Hiljem saab arendaja selle põhjal parandada vigaseid teste.



Joonis 14. Raport testi tulemustest

7 Testid CI/CD keskkonnas

Üks töö eesmärkidest on ajakulu kaotamine. Selle saavutamiseks integreeriti kasutajaliidese testid CI/CD keskkonda, mis võimaldab käivitada teste konteinerites. Kuna VP keskkondade liidesed on ühesugused siis on võimalik luua iga keskkonna jaoks eraldi konteiner ja käitada neid paralleelselt.

Protsess algab moodulite installeerimisega kasutades Dockerfile-i (Joonis 15). Fail laeb alla vajaliku kujutuse (*ingl k: image*) raamistiku töötamiseks. Allalaetud sisu koopia kantakse töökasuta (*ingli k: workdir*). Seejärel installeeritakse vajalikud moodulid visuaaltestimiseks ja raportite genereerimiseks.

```
FROM cypress/included:9.1.1
COPY . /e2e

WORKDIR /e2e

RUN npm i --save-dev cypress-mochawesome-reporter
RUN npm install --save-dev cypress-image-snapshot --force
```

Joonis 15. Docker build programmkood

Alloleval joonisel (Joonis 16) on välja toodud *docker-compose* fail, mis loob konteineri VP arenduskeskkonna testimiseks. Kõigepealt paigaldatakse vajalikud moodulid, mille protsess on esitletud joonisel 15. Seejärel antakse testitavale keskkonnale veebiaadress koos autentimistunnustega. Lõpuks testitakse leht ja tulemused kantakse *results* kausta, kus saab kuvada testide tulemuse raportit.

```
services:
  vp-dev:
    build:
      network: host
      context: ./cypress-data
      dockerfile: Dockerfile
    working_dir: /e2e
    environment:
      - CYPRESS_baseUrl=${user}:${pass}@dev.vportal
    command: "--browser chrome –reporter-options
    reportDir=path,reportFilename=filename"
    volumes:
      - .results/:/e2e/cypress/reports
```

Joonis 16. Konteineri loomine docker-compose failiga

Joonisel 16 on välja toodud lihtsuse mõttes ühe konteineri eksemplar. Mitme keskkonna testimise jaoks saab luua sarnaseid konteinereid nagu *vp-dev*. Erinevatel konteineritel on oma veebiaadress, testide genereerimise kaust ja raporti nimi.

8 Testide analüüs

VP testimistulemused kajastuvad raportites, mis genereeriti eraldi iga keskkonna jaoks. Raportite põhjal loodi kokkuvõttev dokument, kus on kirjeldatud kõikide keskkondade testimistulemused. Selles peatükis on dokumendi sisu kokku võetud formaadis

Keskmiselt kulus keskkondade testimiseks 2 minutit. Testide läbimise protsent oli 78% ning sellest lähtuvalt läbikukkumise protsent 22%. Valepositiivseid ja valenegatiivseid teste ei esinenud.

Vigane test esines kontaktlehe testimisel, kus kontrolliti kontakttabeli ridade suurendamise funktsionaalsust. Probleem seisnes selles, et testis kasutusel olev klass oli arendajate poolt eemaldatud, mille tõttu ei leidnud Cypress lehel olevat tabelit. Vigase testi parandamiseks kulus 60 minutit asendades puuduva CSS klassi olemasolevaga.

Autor pööras tähelepanu rabedatele muutujatele ning soovitas automattestide kindluse jaoks määrata testitavatele elementidele muutujad. See väldiks probleemide tekkimist, kui DOM klassid või id-d eemaldatakse uutega, sest elemendil on küljes kindel muutuja.

Teine vigane test tekkis e-teenuste akna testimisel. Seal kontrollis funktsioon, et e-teenuse aken ei oleks kuvatud, kui ta pannakse kinni. Raamistik andis veateate, et aken on nähtav, kuid raport tagastas pildi ilma e-teenuste aknata. Tegemist oli Cypressi iseärasusega. Probleemi lahendas autor, andes enne akna otsimist ootamise käsu. Seejärel teostas funktsioon kontrolli ja tagastas, et aken ei ole lehel nähtav.

Kõige rohkem kulutas aega keele muutmise test. Testimise käigus esines, et muutes lehel olevat keelt inglise või vene keelele, toimub vahetus pikemalt, kui eesti keelde. Põhjuseks toodi välja, et eesti keel on brauseri pinus. Seetõttu on eesti keelses vaates lehe kuvamine kiirem, kui tesites keeltes, sest eesti keel on mälsut loetav.

9 Automaattestimisest saadud tulemused

Käesolevas töös mõõtis autor manuaaltestimise ja automaattestimisele kulunud aega ning rahalist säästu. Tulemuste põhjal saab järeldada, kas automaattestimine lahendaks ajakulu probleemi ja tasuks teenuse parema kvaliteedi. Lisaks on mõõdetud investeeringu tasuvus, mis näitab ära pikaajalist automaatestide kasu.

9.1 Säästetud aeg

Säästetud ajaga on lihtne ära põhjendada automaattestide rakendamist, sest säästetud ajaga kaasneb rahaline kokkuhoid. Säästetud aega saab kvantitatiivselt leida, arvutades manuaal- ja automaattestimisele kulunud aegade vahe.

Asutuse sõnul pole paralleeltestimisel võimalik testida kõiki keskkondi korraga, sest protsess nõuab palju riistvaralist ressursi. Küllaga on võimalik testida 5 keskkonda paralleelselt ning seejärel testida ülejäänud samamoodi. Kui 5 keskkonna paralleeltestimisele kulub keskmiselt 2 minutit siis 20 keskkonna testimiseks kulub 2 * 4 = 8 minutit

VP testide läbiviimiseks kuluvad ajad on järgmised:

- Manuaaltestimine – 8 tundi.
- Automaattestimine – 8 minutit

Automaattestid vähendavad tööjõu ressursi kulu järgnevalt:

- Projektijuhi säästetud aeg = 8h – 8 min = 8h
- Devops arendaja kaotab palju töötunde näiteks vea leidmise ja parandamisega käsitsi testides. Automaattestimisega on võimalik võita umbes pool sellele kulunud ajast. See tähendab seda, et automaattestide rakendamisel väheneb devopsi ressurss 4 h võrra.

9.2 Säästetud raha

Asutuse informatsiooni kohaselt tehakse tarneid keskmiselt 4 korda kuus. Säästetud aeg ja rahaline väärtus korrutatakse neljaga.

Rahaliselt säästaks ühe tarneprotsessi vältel asutus järgmiselt:

- Projektijuhi pealt 8h * 33€ = 300 €
- DevOpsi pealt 4h * 45€ = 200€
- Ühe tarneprotsessi jooksul säästetakse 200 € + 300 € = 500€.
- Kuu jooksul säästetakse 500 € * 4 = 2000€.
- Aasta jooksul säästetakse 2000 € * 12 = 24 000 €.

9.3 Investeeringu tasuvus

Käesolev jaotis on kirjutatud investeeringu tasuvuse juhendit kasutades [10] ning mõõdab, kas automaattestide kasutamine tasub RMIT-is pikaajaliselt ära.

Investeeringu tasuvus ehk ROI (*Return of investment*) on võrdeline automaattestide säästetud aja ja automaatestidele kulunud aja jagatisega. ROI valem on kujutatud valemis (1).

$$ROI = \frac{Säästetud\ aeg}{Investeeritud\ aeg} \quad (1)$$

Säästud arvutatakse, leides ühe manuaaltesti (MTA) ja automaattesti käitamise (ATA) vahe. See korrutatakse testide hulga (TA) ja käitamiskordadega (TJA).

$$Säästud = (MTA - ATA) * TA * TJA$$

Käivitamise ajad kalkuleeritakse, kasutades 3 punkti hinnangu süsteemi ja arvutades nendevahelise keskmise väärtuse [11].

$$MTA = \frac{7{,}5 + 20 + 45}{3} = 24\ min$$

$$ATA = \frac{0{,}039 + 3{,}2 + 5{,}2}{3} = 3\ sek\ \approx 0{,}05\ min$$

Teste ehk testjuhtumeid oli kokku 32. Kokku käitati teste 20 korda, sest 20 VP keskkonda oli vaja paralleelselt ära testida. Seega on säästetud minutite arv järgmine:

$$Säästetud\ aeg = (24 - 0.05) * 32 * 20 = 15\ 000\ min$$

Automattestimise investeering mõõdetakse, leides testimisraamistikule kulunud nädalate arvu (KEA). See liidetakse ühe automaattestimise arendamisele kulunud aja (ÜTA) ning testide arvu korrutisega (TA). Seejärel liidetakse saadud tulemus hooldamisajaga (HA).

$$Investeeritud\ aeg = KEA + ÜTA * TA + HA$$

Asutuse andmete kohaselt võtab raamistiku ehitamiseks aega umbes 5 minutit. Kuna KEA väärtus leitakse nädalates, siis 5 minutit on võrdne 0,0005 nädalaga. Ühe testi arendamiseks kulus keskmiselt 45 minutit. Teste oli kokku 32.

Hooldamisaeg on võrdeline ühe testi hooldamiseks kuluva aja, läbikukkunud testide protsendi, testide arvu ja käitamiste arvu korrutisega. Vigaste testide protsent oli 20%.

$$HK = 60 * 0{,}2 * 32 * 20 = 7500\ min$$

Mõõdetud tulemustega arvutatakse investeeritud aeg:

$$Investeeritud\ aeg = 0{,}0005 + 45 * 32 + 7500 = 9000\ min$$

Saadud säästude ja investeeritud minutitega arvutatakse nende jagatis.

$$ROI = \frac{15\ 000}{9000} = 1{,}66$$

Saadud aegade jagatis on 1,66 millest võib järeldada, et VP automaattestimisega säästetakse rohkem aega (15 000 minutit ehk 250 tundi) kui kulutatakse selle arendamisele ja haldamisele (9000 minutit ehk 150 tundi). See tõestab, et VP liidese testimisel on efektiivsem kasutada automaattestimist, sest automatiseerimine tasus 66% rohkem väärtust, kui sellesse investeeriti.

Kokkuvõte

Käesolev bakalaureusetöö lahendab aktuaalset probleemi tarkvaraarenduse protsessis asutuse RMIT näitel. Valitsusportaali (edaspidi VP) keskkondade kasutajaliideste testimine toimub manuaalselt, mis kulutab asutuses liigselt aega ja rahalist ressurssi. Probleemi lahendamiseks otsustas asutus leida võimaluse, milleks on automaattestimine.

Uue testimismeetodi jaoks koostati raamistike analüüs. Neljast liidese testimise tehnoloogiatest valiti välja Cypress. Tema eeliseks oli mahukas kogukond (kasutajad ja tööturg) ja terviklik dokumentatsioon, võrreldes teiste tööriistadega. Raamistiku võimekust tutvustati asutuse arhitektidele. Neile oli tähtis, et Cypressiga oleks võimalik tuvastada VP CSS-i olemasolu ning sooritada kasutaja autentimist ja tuvastada sellega ilmnevaid turvariske.

Seejärel kirjutati 3 testi, mille eesmärk oli testida esi-, kontakt- ja otsingu tulemuste leht. Pärast seda integreeriti testid CI/CD keskkonda, et oleks võimalik paralleelselt testida kõik VP keskkonnad. Paralleelsuse saavutamine oli töö üks tähtsamaid osi, sest VP veebilehed on üles ehitatud ühesuguse liidesega. Sellest lähtudes oli võimalik kirjutada minimaalne arv spetsifikatsioone ja testjuhtumeid VP testimiseks. Saadud tulemustest kirjutati valmis analüüs, kus kajastusid testimisel esinenud probleemid, näiteks valepositiivsed või valenegatiivsed olukorrad. Lisaks tuvastati teste, mille käitamise protsess on kauakestev.

Automaatestidest saadud tulemused võrreldi manuaaltestimisega. Arvutuste kaudu leiti, et automatiseerimine toob asutusele investeeringu tasuvuse väärtusega 1,66. See tähendab seda, et asutus säästab automaattestimisega ajaliselt 66% kui sellese investeeritakse. Lisaks hoitakse kokku projektijuhi ja devops arendaja aega ja tasustamist.

Asutus otsustas automatiseerimisest saadud positiivsete tulemuste tõttu automaatteste edaspidi rakendada teistele veebilehtedele. Valitsusportaalile planeeritakse arendada sisu loomise ja kustutamise teste. Asutuses on soov tarnida testija rolle ning kokku panna meeskond, kes haldaksid automaatteste tulevikus.

Kasutatud kirjandus

[1] Pilv.Riigikantselei, *Valitsusportaal.* 2020. [Online] Loetud addressil: https://pilv.riigikantselei.ee/index.php/s/d7pf3oS6ay3dxiw?path=%2FArhitektuuripaneel Kasutatud: 17.11.2021

[2] Search Software Quality, *user story.* 2020. [Online] Loetud addressil: https://searchsoftwarequality.techtarget.com/definition/user-story Kasutatud: 15.112021

[3] Gitlab, *CI/CD Consepts.* 2019. [Online] Loetud addressil: https://docs.gitlab.com/ee/ci/introduction/ Kasutatud: 05.11.2021

[4] Nightwatch.js, *What is Nightwatch?* 2021. [Online] Loetud addressil: https://nightwatchjs.org/gettingstarted/ Kasutatud: 03.10.2021

[5] TestCafe, *Why people love TestCafe?* 2021. [Online] Loetud addressil: https://testcafe.io Kasutatud: 04.10.2021

[6] Webdriver.IO, *What is Webdriver.IO?* 2021. [Online] Loetud addressil: https://webdriver.io/docs/what-is-webdriverio Kasutatud: 03.10.2021

[7] Cypress, *Why Cypress?* 2021. [Online] Loetud addressil: https://docs.cypress.io/guides/overview/why-cypress#In-a-nutshell Kasutatud: 01.10.2021

[8] Cypress, *Visual Testing,* 2021. [Online] Loetud addressil: https://docs.cypress.io/guides/tooling/visual-testing Kasutatud: 20.11.2021

[9] Pilv.Riigikantselei, *VP 2.0 Stiiliraamat,* 2020. [Online] Loetud addressil: https://pilv.riigikantselei.ee/index.php/s/d7pf3oS6ay3dxiw?path=%2FVP%202.0%20stiiliraamat#pdfviewer Kasutatud: 17.11.2021

[10] Slalom_build, *What is the ROI of my test automation?,* 2020. [Online] Loetud aadressil: https://medium.com/slalom-build/what-is-the-roi-of-my-test-automation-10ae7bf0d9ed Kasutatud: 30.11.2021

[11] ProjectManegement, *Triangular Distribution – Three-point estimating technique,* 2019. [Online] Loetud aadressil: https://www.projectmanagement.com/contentPages/wiki.cfm?ID=540506&thisPageURL=/wikis/540506/Triangular-Distribution---Three-point-estimating-technique#_=_ Kasutatud: 30.11.2021

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

Mina, Kristo Loit

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Kasutajaliidese automaattestid Valitsusportaali näitel" , mille juhendaja on Maili Markvardt
   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.

29.12.2021

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.