TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Olesja Tsihhotskaja 185933

AUTOMATISEERITUD TESTIDE KOOSTAMINE WWW.TALTECH.EE VEEBILEHE NÄITEL

Bakalaureusetöö

Juhendaja: Jekaterina Tšukrejeva
MSc

Tallinn 2021

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: Olesja Tsihhotskaja

18.05.2021

Annotatsioon

Käesoleva bakalaureusetöö eesmärk on võrrelda ja valida tööriist uue Taltechi veebilehe kasutajaliidese automatiseeritud testide kirjutamiseks, et uurida veebileht vigade olemasolule.

Bakalaureuse töö käigus oli läbi viidud erinevate testimismeetodite ja -vahendite teoreetiline võrdlus, oli tehtud ülevaade nende eelistest ja puudustest ning tulemuste põhjal oli tehtud valik Selenide raamistiku kasuks.

Teoreetilise teabe ja järelduste põhjal olid kirjutatud automatiseeritud testid Tudengiportaali peamenüü jaoks, kasutades kahte automatiseerimisvahendit - Selenide ja Selenium, ning oli tehtud nende võrdlus võttes arvesse testijatele kehtestatud nõudeid ja kasutusmugavust.

Töö tulemuseks on automatiseeritud testide loomine, mis kontrollivad kasutajaliidese elementide vastavust ning kontrollivad vigu kasutajaliideses.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 51 leheküljel, 3 peatükki, 35 joonist, 3 tabelit.

Abstract

Compilation of Automated Tests on the Example www.taltech.ee Website

The aim of this bachelor's thesis is to compare and select a tool for writing UI automated tests of the new Taltech website to investigate the existence of errors in the website.

The bachelor's thesis consists of three main parts.

The first part introduces the topic of the bachelor's thesis.

The second part is dedicated to the comparison of testing methods currently in use and the analysis of these methods, taking into account their pros and cons in order to choose the most appropriate one to be used for writing UI tests.

The third part is dedicated to the practical writing and comparison of automated tests using two automation tools, Selenide and Selenium, for the Student Portal (https://student.taltech.ee/) and their subsequent analysis.

The result of the work is the creation of automated tests that check the compliance of the user interface elements and check the UI for errors.

The thesis is in Estonian and contains 51 pages of text, 3 chapters, 35 figures, 3 tables.

Lühendite ja mõistete sõnastik

AJAX	*Asynchronous JavaScript And XML*
Agile	Paindlik programmeerimis metoodika
Allure	Raamistik, paindlik mitmekeelse testimise aruandlustööriist.
API	*Application Programming Interface*, funktsioonide ja protseduuride kogum rakenduste loomiseks
BDD	*Behavior-Driven Development*, käitumisel põhinev arendus
CI/CD	*Continuous integration and continuous delivery*
Cucumber	Avatud lähtekoodiga *Behavior Driven Development* raamistik
DevOps	Tarkvaraarenduse kultuur, mille eesmärk on ühendada tarkvaraarendus ja tarkvaraoperatsioonid
HTML	*HyperText Markup Language*, veebilehtede märgistuskeel
HTTP	*Hypertext Transfer Protocol*, andmete edastamise aluseks veebis
JMS	*Java DataBase Connectivity*, Java-rakenduste koostalitlusvõime standard
REST	*Representational state transfer*, tarkvaraarhitektuuri laad
Open-source	Avatud lähtekoodiga tööriist
Selenium WebDriver	Tasuta, vaba lähtekoodiga brauseri automatiseerimisvahend
SOA	*Service-oriented architecture*
SoapUI	Avatud lähtekoodiga rakendus veebiteenuste testimiseks
UI	*User interface*, kasutajaliides
Unit test	Programmeerimiskoodi väikeste osade testimine
UX	*User experience*, kasutajakogemus

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Digitaalse toote kvaliteedi määravad paljud kriteeriumid. Arendajad püüavad alati muuta toodet maksimaalselt kasutajasõbralikuks ja minimeerida vigu selle töös. Antud eesmärgi saavutamiseks hea koodi kirjutamine pole piisav. Juba arendusjärgus on hädavajalik läbi viia ka toote põhjalik testimine.

Bakalaureusetöö koosneb kolmest põhiosast.

Esimene osa avab bakalaureuse töö teema .

Teine osa on pühendatud praegu kasutatavatele testimisviisidele ja -meetoditele ning nende meetodite analüüsile, arvestades nende plusside ja miinustega. Seejärel analüüsitakse erinevaid automatiseeritud testimisvahendeid, võetakse arvesse nende eeliseid ja puudusi ning valitakse esitatud tööriistadest kõige sobivam, mis aitab kasutajaliidese testide kirjutamisel (UI-testimine).

Kolmas osa on pühendatud automatiseeritud testide praktilisele kirjutamisele ja võrdlemisele Tudengiportaalis (https://student.taltech.ee/) toimuvate vigade leidmiseks ja nende järgnevaks analüüsimiseks.

Antud bakalaureusetöö põhieesmärk on testide kirjutamine uue TTÜ veebilehe funktsionaalsuse automatiseerimiseks vigade leidmiseks ja tuvastamiseks ning nende uurimiseks. Autor läbis testija praktikat ettevõttes HireRight. Antud ettevõte tegeleb ankeediandmete kontrollimisega (candidates' pre-employment background screening). Praktika käigus autor sai ülesande koostada automatiseeritud teste ettevõtte veebilehe jaoks (Applicant center). Paraku sellepärast, et ettevõte töötab klientide isikuandmetega, kahjuks autoril ei lubanud kasutada tööandja koodi käesoleva lõputöö raames. Seetõttu otsustas autor funktsionaalsuse automatiseerimiseks kasutada Taltechi veebilehte, kuna Tudengiportaali funktsionaalsus on üsnagi sarnane nagu tööandja-ettevõte veebilehel. Umbes 10 aastat tagasi ettevõte HireRight läks üle automatiseeritud testimisvahendile Selenium. Tehnilisteks nõueteks olid lihtsus, paindlikkus, testimisvahendi laialdane kasutamine teistes ettevõtetes ning samuti võimalus kasutada seda tasuta.

Ülesannete hulka kuuluvad: keeruliste ja probleemsete kohtade tuvastamine automaattestimise abil. Autor võrdleb erinevaid automatiseeritud testimisvahendeid ja valib kõige sobivama valitud probleemi lahendamiseks. Lõpptulemuseks on kõige tõhusamate automaattestide loomine, mida soovi korral ja ressursside olemasolul oleks võimalik kasutada veebilehe töö parandamiseks.

1.1 Taust ja probleem

Aastast 2020 Taltechil on kasutusel uus veebileht. Lahenduste väljatöötamisel on olulised vajadused, antud juhul peaks veebileht olema võimalikult mugav ja intuitiivselt arusaadav nii tudengite ja õpetajate kui ka tavakülastajate jaoks. Käesoleva bakalaureusetöö autor soovib uurida uut veebilehte testija vaatepunktist, kes võtab endale üliõpilase rolli ja püüab teada saada, kui lihtne on antud elektroonilise toote kasutamine ja kas see vastab kasutaja nõuetele.

Üliõpilaste jaoks on oluline asjakohane ja kergesti kättesaadav informatsioon. Kui veebileht on vigane ja selle arendajad ei märganud riket õigel ajal, siis võib see põhjustada korrektse teabe saamise halvenemist, desinformatsiooni, mis omakorda võib mõjutada akadeemilisi tulemusi. Automaatne UI testimine arendusetapis võimaldab õigeaegselt leida vigu ja neid parandada. Kasutajatel ei teki probleeme veebilehe töö mõistmisega ja uuendatud teabe kättesaamisega.

1.2 Ülesande püstitus

Kui veebileht ei tööta korrektselt , see põhjustab põhikasutajate rahulolematust. Seetõttu on oluline tagada, et toode toimiks hästi ja oleks võimalikult veatu.

Antud töö eesmärk on kirjutada automaatsed testid vigade ja defektide tuvastamiseks, kasutades automaatset testimisvahendit järgmiste vigade jaoks:

- veebilehe elementides puudub või on vale teave;
- veebilehe jagude puudumine;
- veebilehe struktuuri erinevused erinevas keeleversioonides.

Ülesannete hulka kuuluvad järgmised aspektid:

- TTÜ uue veebilehe automaatsete testimisvahendite võrdlus ja lõplik valik;
- automatiseeritud testide kirjutamine valitud tööriistade abil;
- saadud testi tulemuse analüüs;
- testi tulemuse analüüsi põhjal sobivaima automatiseerimisvahendi määramine, mis tagaks korrektse töö ja veebis kuvamise.

Autori eesmärk on tagada veebilehe automaatika abil vigade puudumist Tudengiportaali veebirakenduses. Vigade elimineerimine ja õigeaegsete tooteuuenduste teostamine suurendab kasutajate rahulolu tootega, aitab neil võimalikult kiiresti saada ajakohast ja usaldusväärset teavet ning kasutada ka funktsionaalseid funktsioone (näiteks: kalender, post). Lisaks vigade leidmine automaatrežiimis oluliselt lihtsustab ja kiirendab veebilehe uute versioonide väljaandmist.

1.3 Metoodika

Käesolev bakalaureusetöö sisaldab nii teoreetilist kui ka praktilist osa.

Teoreetilises osas käsitletakse automaatse testimise rolli lõppkasutaja jaoks, uuritakse kasutajaliidese testimise (UI testimine) teoreetilisi allikaid.

Autor võrdleb manuaalset ja automatiseeritud testimist, analüüsib mõlema meetodi eeliseid ning puuduseid ja selgitab välja, millist metoodikat oleks kõige otstarbekam

kasutada Taltechi kasutajaliidese testimiseks.

Autor analüüsib mõningaid kättesaadavaid automatiseeritud testimistööriistu, et leida veebilehe testimiseks optimaalset meetodit, ning võrdleb neid Seleniumiga, kuna seda tööriista kasutab tema tööandja-ettevõte.

Selgitatakse välja iga testimstööriista eelised ja puudused ning tulemuste põhjal valitakse välja kõige sobivam vahend probleemi lahendamiseks, mida võrreldakse Seleniumiga nii teoorias ja ka praktikas.

Autor koostab testimiseks kahe valitud tööriista abil projekti, mis sisaldab järgmisi samme:

- Testikeskkonna ettevalmistamine (brauseri seadistamine);
- Parameetriliste testide loomine Junit'il;
- Junit'i parameetriliste testide (baastestide väljatöötamise standard) kasutamine sisendbaasina valitud kasutajaliidese (UI) testimisvahendi kasutajaliidese testide käivitamiseks;
- Visuaalse aruande koostamine Allure'i põhjal (laialt kasutatav raamistik aruannete loomiseks koos statistikaga testi teostamise kohta ja teabe kuvamiseks kõigi läbitud sammude kohta).

Saadud võrdluse põhjal on võimalik teha valik konkreetse vahendi kasuks.

2 Testimise tutvustus

Iga digitaalne toode läbib enne lõppkasutajani jõudmist mitmeid erinevaid teste. Lõputöö käesolevas peatükis kavatseb autor teha ülevaade praegu kasutatavate testide tüüpide ja meetodite klassifikatsioonist, analüüsida nende tüüpide plusse ja miinuseid ning saadud järelduste alusel valida kõige sobivam antud töö eesmärgi jaoks. Ehk nimelt testide kirjutamiseks, mis jälgivad veebilehe elementide toimivust ja kättesaadavust, näiteks nagu teabeplokid ja veebilehe teistesse jaotistesse navigeerimise nupud.

2.1 Testimise väärtus

Enne testimist käitub digitaalne toode ettearvamatult. Pole teada, kas kõik toimib nii, nagu oli ette nähtud. Enne veebilehe/rakenduse käivitamist või uuendamist tehtavad testid aitavad veenduda, et kood pole haavatav ega vigane. Seetõttu on vajalik testimise läbiviimine. Uue funktsionaalsuse lisamisel (mistahes funktsionaalsus, isegi väga väike, näiteks nagu fondi muutmine, nupu lisamine), on vaja kindel olla, et brauseris kõik kuvatakse õigesti ja on kasutajatele kättesaadav ning arusaadav[1].

Testimine aitab leida toote nõrku kohti kõigis etappides, mis annab võimaluse esitada tootele uusi nõudmisi. Vastavalt *Nielsen Norman Group*'i andmetele[2], kvaliteedi tagamine mõjutab suuresti kasutajakogemust. Kui funktsionaalsus ei toimi ettenähtud viisil, hakkavad kasutajad kahtlustama oma arusaamist veebilehe töös ja võivad hakata kasutama ebaefektiivseid kõrvalteid.

Testi tulemus näitab, kui loogiline projekt on ja kas selle kasutamise etapid on selged ja üheti mõistetavad. Kui toodet pole testitud, siis mõni funktsioon võib mitte töötada, just testimine toob ilmsiks kõik toote kitsaskohad.

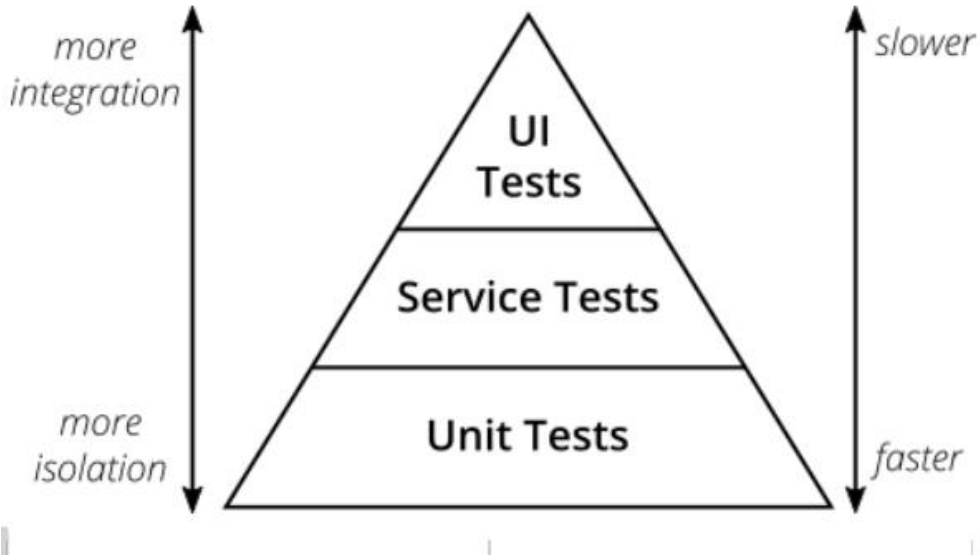
Testimine aitab:

- säästa ressursse, et hoida digitaaltoodet töötavas seisundis;

- tagada koodi turvalisust, juhul kui arendajad tulevikus soovivad selles midagi muuta;
- parandada arhitektuuri ja parandada koodi usaldusväärsust.

2.2 Automaattestimise püramiidi mõiste

Testpüramiidi kontseptsiooni tutvustas Mike Cohn oma raamatus *"Succeeding With Agile. Software Development Using Scrum*"[3]. Kontseptsioon esindab testimise automatiseerimise kolme taseme kirjeldust, nende omavahelist suhet ja suhtelist tähtsust[4].



Joonis 1. Näide:Testimise püramiid.

.

Mike Cohn'i originaalne testipüramiid koosneb kolmest tasandist[5]:

- Ühikutestid kontrollivad, kas üksik üksus (testitav subjekt ehk unit) töötab ootuspäraselt. See on veebilehe funktsionaalsuse üksikute väikeste osade testimine isolatsioonis. Ühikutestide arv komplektis ületab oluliselt muude testide arvu.
- Teenindustest (API testimine). Sellised testid kontrollivad rakenduse juurdepääsetavate veebiteenuste kättesaadavust. Kõige sagedamini töö nendega on korraldatud API-taotluste abil [6].

- Kasutajaliidese testid (*UI ehk User Interface Tests*) kontrollivad süsteemi reageerimise õigsust lõppkasutaja toimingutele. Need testid kontrollivad kogu süsteemi töötamist. Need simuleerivad kasutaja käitumist: programm avab brauseri, külastab lehte, kontrollib funktsionaalsust ja kuvamist. Test kontrollib veebilehtede tööd ja aitab veenduda, et kõik veebilehe funktsioonid – lingid, otsingud ning küpsised töötaksid täpselt nii, nagu on ette nähtud. Selliste kontrollimiste hulka kuuluvad näiteks testid, mis kontrollivad süsteemi reageerimist tekstiväljade täitmisele või nuppudele vajutamisele. Tavaliselt iga süsteemi funktsionaalsus, mis võiks olla testitud *Unit*'i, komponendi või teenindustestide abil, peaks olema kontrollitud, kasutades nimetatud testimise tüüpi. Kasutajaliidese testid peaksid keskenduma ainult kasutaja graafilise liidesega interaktsiooni jooksul tekkinud vigade tuvastamisele.

Cohn'i kontseptsiooni kohaselt väheneb testi sooritamise kiirus alt üles (üksustestimisest kasutajaliidese testideni). Mida kõrgem on tase, seda vähem teste on. Selle jaoks, et rakendusel oleks kiire ja muutustele reageeriv testbaas, enamus sellest peaks olema moodultestide jaoks, väike osa — teenindustestide jaoks, ning väiksem osa — integreeritud testimise jaoks [7].

2.3 UX

UX *(user experience)* ehk kasutajakogemus — digitaalse toote kasutamisest tulenev lõppkasutaja emotsioonid [8].

Kaasaegses maailmas on väga tihe konkurents, iga ettevõte soovib kasutajaid meelitada ja just oma toodet kasutama ja ostma. Üha rohkem ettevõtteid hakkab järjest enam UX-ile tähelepanu pöörama, sest nad saavad aru, selles võib olla suur mõju nende toote pikaajalisele äriedukusele. UX on oluline, kuna see on otseselt seotud sellega, kuidas kasutajad tajuvad veebilehte.

Kasutajakogemus hõlmab kõiki emotsioone, veendumusi, eelistusi, aistinguid, kasutaja füüsilisi ja psühholoogilisi reaktsioone, käitumist ja saavutusi, mis ilmnevad enne interaktsiooni süsteemiga konkreetsel ajal ja pärast seda. Interaktsiooni all mõistetakse kõik kasutaja veebilehel tehtud klikke, kerimisi ja toiminguid. Need on kasutajakogemuse lahutamatu osa: kui inimesel ei teki veebilehega edukat interaktsiooni, ei saa tema

kogemust nimetada positiivseks ja on ebatõenäoline, et ta sooviks sellist toodet uuesti kasutada või teistele soovitada [9].

Selle jaoks, et UX oleks edukas, on vaja läbi viia UI-testimine.

2.4 UI ja selle testimine

UI ehk kasutajaliides on programmi "nägu", mis on inimese ja digitaalse toote interaktsiooni punktiks. Seega hindab kasutaja kogu süsteemi toimivust ja tõhusust kasutajaliidese põhjal.

Kui kasutaja on jõudnud veebilehele, kuid ei suuda navigeerimisest aru saada, lehed on läinud segamini, ei leia olulist teavet või kui veebileht on kuvatud tema seadmes ebakorrektse paigutusega — see tähendab, et veebiressurss on kaotanud potentsiaalse kliendi. Konversioon väheneb [10].

„Kasutajasõbralik liides" eeldab külastaja lojaalset suhtumist, et veebilehel töötamise mehhanism on arusaadav, kõik teenused on ennetuslikud, kasutajasõbralikud jne.

Mõnikord tõmmatakse paralleel liidese ja veebiprojekti disaini vahel. Kuid on olemas märkimisväärne erinevus. Disain on välimus, aga liides on inimese interaktsioon veebiprojektiga. Täisjõulisel ressursil peab kindlasti olema kasutajasõbralik liides. Rakenduse välimus peaks aitama kaasa toote kasutusmugavusele ja arusaadavusele: nuppude kujundus, paigutus, sisestusväljad, teksti loetavus jne.

Liidese test simuleerib rakenduse kasutaja toiminguid. Testimise eesmärk on kontrollida, kas kõik süsteemi komponendid töötavad omavahel õigesti ja annavad oodatud tulemuse [11].

UI-testimist tuleks läbi viia juba tootearenduse algstaadiumis — prototüübi etapis. See säästab aega ja raha ning suurendab rakenduse usaldusväärsust.

UI-testimine aitab välja selgitada, mis määral liidest on lihtne kasutada ja kas see vastab määratletud nõuetele ja heakskiidetud prototüübile. Samuti kontrollitakse, mil määral programm käitub nagu eeldati ning kuidas liidese elemendid on kuvatud erinevates seadmetes, ka siis kui kasutaja sooritab teatud toiminguid. See võimaldab hinnata, kui efektiivne on kasutaja töö rakendusega.

UI-testimise ülesanne on tuvastada rakenduse liidese struktuursed ja visuaalsed vead, kontrollida liidese kasutatavust navigeerimiseks ja rakenduse funktsionaalsuse täieliku kasutamise võimalust [12].

Testija kontrollib, kuidas liidese elemendid reageerivad kasutaja toimingutele ja kuidas rakendus käsitseb klaviatuuri ja hiire või puutetoiminguid. Niimoodi tuvastab testija palju rohkem kasutajale avalduvaid vigu, kuna ta töötab rakendusega kasutaja vaatepunktist. Nendel eesmärkidel kasutatakse mitut erinevat lähenemist:

- manuaalne testimine, mis võimaldab kontrollida kasutajaliidese rakenduse vastavust disainile ja prototüübile;
- automaatne testimine pärast iga toote loomist liidese vigade tuvastamiseks;
- Fookusgrupi testimine on uurimismeetod, mille puhul kutsutakse kasutajaliidese testimises osalema kasutajate rühm või mitu iseseisvat rühma erineva taustaga (näiteks võivad nad olla nii algajate, vahetarbijate kui ka tehniliste spetsialistide rühmad). Tavaliselt kasutajad on potentsiaalsed kliendid, kellel palutakse hinnata liidese kasutatavust. Fookusgrupi meetodi abil on võimalik saada teavet lõppkasutaja rakenduses kuvatavate omaduste kohta, mida teiste meetoditega ei saa teada. Antud meetod võimaldab paremini mõista kasutajate probleeme ja soove. Meetod on hea, sest nii väikese kasutajate arvu korral (mitte rohkem kui 7–12) võib liideses leida umbes 80% vigadest ja ebatäpsustest [13].

Testimisel võetakse arvesse kvaliteetse UI loomise järgmisi kriteeriume:

- ülesannete täitmisele kuuluv aeg. Parem on selline liides, milles ülesande täitmise aeg on lühem;
- minimaalne vigade arv, mida kasutaja teeb töötades rakendusega;
- kasutajate täielik arusaamine liidesest ja sellega töötamisel ebaselguste puudumine;
- minimaalne kasutajate sisestatud teabe hulk; samu andmeid ei tohiks mitu korda sisestada;
- liidese lihtsus ja visuaalne atraktiivsus.

Kasutajaliidese testimiseks viiakse läbi ka brauserite ja mitme platvormi testimine, mille järel saadakse kvaliteetne rakendus, mis töötab igat tüüpi seadmetes ja kõigis kaasaegsetes brauserites [12].

2.5 UI-testimise põhitehnikad

Praegu on olemas kaks peamist testimise viisi. Digitaalset toodet saab testida käsitsi või automaatselt [1]. Igal lähenemisviisil on omad plussid ja miinused, mida töö autor võrdleb ja analüüsib, et valida optimaalsem variant töös püstitatud eesmärgi saavutamiseks.

2.5.1 Manuaalne testimine

Manuaalne testimine on tarkvara testimine, mille käigus QA spetsialist teeb teste käsitsi. Manuaalse testimise jooksul testija kontrollib toote kõiki põhifunktsioone ja kasutab veebilehte nagu tavakasutaja ning testimine toimub ilma tarkvara automatiseeritud testide tööriistadeta [16].

Manuaalsel testimisel on olemas mitmed eelised, nagu näiteks tagasiside kasutajaliidese kvaliteedi kohta, kuna testija aruanne on esimene tagasiside potentsiaalselt kliendilt, mis annab arusaamise, kui hea või halb kasutajaliides on, või näiteks mis määral toode on mugav lõppkasutaja jaoks. Rakenduse üldise disaini testimine ja puuduste tuvastamine on võimalik ainult manuaalse testimise abil [14]. Samuti kulud manuaalse testimise puhul on madalamad, selle rakendamise kulud väikeste projektide jaoks on alati väiksemad, kui automatiseeritud testimise rakendamise kulud. Lisaks tasub mainida manuaalse testimise paindlikkust, näiteks toote väiksemate muudatuste korral saab testimise läbi viia otsekohe ilma lisakuludeta koodi kirjutamise eest. See on eriti oluline uue funktsionaalsuse kiire rakendamise puhul, kui on vaja olla kindel, et funktsionaalsus töötaks korrektselt.

Manuaalsel testimisel on ka mitmeid puudusi, näiteks väiksem usaldusväärsus, kuna testimisprotsessi viib läbi inimene ning see on alati veaohtlik, ehk inimfaktorit ei saa täielikult välistada.

2.5.2 Automatiseeritud testimine

Automatiseeritud testimisel kasutatakse automatiseeritud tööriistu testjuhtumite käivitamiseks, mida saab taaskasutada ja see aitab säästa aega. Programm määrab, kas skript vastab kasutaja ootustele. See aitab testijal kindlaks teha, kas rakendus toimib

ootuspäraselt või mitte. Automatiseeritud testimine on eriti oluline suure koormusega projektide jaoks, millel on palju funktsionaalsust.

Automatiseeritud testimine toetub täielikult eelvalmistatud skriptile, mis töötab automaatselt tegelike ja eeldatavate tulemuste võrdlemiseks. Vaatamata sellele, et kõik protsessid on automaatsed, automatiseerimine nõuab mõnesid käsitsi tehtud jõupingutusi, et luua esialgse testimise skripte.

Automatiseeritud testides on võimatu ette näha kõiki võimalikke vigu. Automatiseeritud test ei suuda programmi hinnata kasutaja vaatenurgast. Maksimaalselt on võimalik saada ainult vastus küsimusele, kas süsteem töötab või mitte. Isegi kui minna üle automatiseeritud testidele, pole võimalik täielikult loobuda manuaalsest testimisest. Manuaalsete testijate töö suurendab testide katvust lisaks automatiseeritud testidele ja aitab leida vigu, mida automaatselt ei saa tuvastada [1]. Automatiseeritud testimise peamised eelised hõlmavad paremat tootlikkust ja kiirust (automatiseeritud testimine on kiirem ja tõhusam protsess, see aitab leida rohkem vigu võrreldes inimtestijaga), võimalust taaskasutada teste, samuti puudub inimfaktor. Puuduseks on rakendamise suuremad kulud.

Manuaalse ja automatiseeritud testimismeetodite üksikasjalik võrdlus on käsitletud järgmises peatükis.

2.5.3 Manuaalse ja automatiseeritud testimise võrdlus

Selles peatükis teeb autor manuaalse ja automatiseeritud testimismeetodite üksikasjaliku võrdlust, et valida kõige sobivam meetod käesoleva töö eesmärgi täitmiseks, mis seisneb selles, et muuta testimisprotsessi võimalikult efektiivseks ja usaldusväärseks.

Peamine erinevus kahe meetodi vahel on järgmine:

Manuaalset testimist teeb käsitsi QA analüütik, samal ajal kui automatiseeritud testimist teeb testija, kasutades skripte, koodi ja automatiseerimise tööriistu [15].

Manuaalne testimisprotsess on ebatäpne inimfaktorist tingitud vigade võimaluse tõttu, samas kui automatiseerimisprotsess on usaldusväärne, kuna see põhineb koodil ja skriptil.

Manuaalne testimine on aeganõudev ja töömahukas protsess, protsessi pole võimalik salvestada, seetõttu on käsitsi testide korduvkasutamine võimatu, samas kui automaatne

testimine annab võimalust salvestamiseks ja taaskasutamiseks, võimaldades testida tarkvara palju kiiremini.

Manuaalne testimine on võimalik ilma programmeerimise oskuseta, samas kui automatiseeritud testimine pole võimalik ilma programmeerimise oskuseta.

Manuaalne testimine võimaldab juhuslikku testimist, samas kui automaatne testimine ei võimalda juhuslikku testimist [16].

Antud töö eesmärgi saavutamiseks autor peab kõige olulisemaks järgmisi testimiskriteeriume: kiire töötlemisaeg, usaldusväärsus (inimfaktori puudumine vigade vältimiseks), efektiivsus (sisaldab testide taaskasutamist). Kasulikud on ka inimese hinnang (kasutajasõbraliku UI loomiseks) ning testimise aruanne.

Tabel 1. Manuaalse ja automatiseeritud testimise võrdlus.

Kriteerium	**Automatiseeritud testimine**	**Manuaalne testimine**
kiire töötlemisaeg	+	_
usaldusväärsus	+	_
Testide taaskasutamist	+	_
inimese hinnang	_	+
Automaatne testimise aruanne.	+	_

Võrreldes kahte testimismeetodit, autor jõudis järeldusele, et selle töö eesmärkide saavutamiseks sobib paremini automatiseeritud testimismeetod.

2.6 Automaattestide loomiseks vahendi valimine

Tänapäeva kiiresti muutuvas tarkvaraarenduse maailmas automatiseeritud testimisvahendid mängivad usaldusväärse toote loomisel kriitilist rolli, samas tagades kvaliteeti ja kiirust. Pidevalt muutuva nõudlusega sammu pidamiseks teevad organisatsioonid suurt hüpet pideva integreerimise (CI) ja pideva kasutuselevõtu (CD), Agile ja DevOps metoodikate suunas. Testide automatiseerimine on nende aspektide põhiolemus. Automatiseerimisvahendite eesmärk on testimise aja vähendamine, parem katvus ja testjuhtumite tõhusam kasutamine.

Kvaliteetse toote kiireks valmimiseks on hädavajalik õigel ajal kasutada õigeid tööriistu testide automatiseerimiseks. Need tööriistad tagavad, et automatiseerimise eelised realiseeritakse täiel määral. Seega automatiseeritud tööriistad on tarkvaraarenduse protsessis ülioluline komponent.

2.6.1 Nõuded automatiseerimisvahendile.

Edukaks testimiseks, mille käigus kasutatakse automatiseerimist, on vaja kindlaks määrata nõudeid testimisvahendile, iga ettevõtte jaoks võivad need nõuded olla erinevad.

Tööriista valimisel tuleks arvestada ka testitava rakenduse/toote eripäradega, sest igal rakendusel on oma eripärad, mis piiravad tööks sobivate automatiseerimisvahendite valikut. Näiteks selles töös autor testib veebirakenduse kasutajaliidest, seega tööriist, mida valitakse, peaks olema antud ülesande jaoks sobiv. Samuti on vaja arvestada testijate soovide ja vajadustega, sest just nemad on selle peamised kasutajad.

Tööandja-ettevõtte HireRight eesmärk oli, et tööriist peab olema tasuta, lihtne, efektiivne ja usaldusväärne vigade otsimiseks, millest autor hakkab ka lähtuma ning lisab ka veel teisi vahendeid, mis võivad kasulikud olla.

Veebilehe kasutaja töö imitatsiooni raames peaks valitud tööriist pakkuma mugavaid võimalusi veebilehe elementide analüüsimiseks ja haldamiseks automaatrežiimis. Mida lihtsam on testi kirjutada, seda lihtsam on seda ka ajakohasena hoida ka mitte spetsialistile. Autor võtab need soovid aluseks ja lisab juurde veel mõned, mis on kasulikud. Automatiseerimistööriista jaoks, mida hakatakse kasutama Tudengiportaali testimiseks esitatakse järgmised nõuded:

- Kokkusobivus Java-ga;
- Veebilehe testimise võimalus;
- Automaatsete aruannete genereerimine;
- Elementide otsing teksti järgi/tekstiandmete kontroll;
- Automaatne brauseri seadistamine;
- Mitme sirvija ühilduvus (*Cross-browser compatibility*, võimalus kasutada tööriista enamiku enamkasutatavate brauseritega);
- Tööriist peab olema tasuta;
- Lihtne õppida ja kasutada;
- Laialdaselt kasutatav (usaldusväärsuse tagamine).

Käesoleva lõputöö autor käsitleb mõningaid populaarseid tasuta automatiseeritud testimisvahendeid. Interneti-otsing kõige populaarsemate automatiseerimistööriistade kohta pakub võrdluseks järgmist loendit [17]: SoapUI, Selenium, Cucumber, Selenide. Rohkem detaile igaühe riista karakteristikute ja omaduste kohta on esitatud järgmistes punktides.

2.6.2 SoapUI

SoapUI on avatud lähtekoodiga rakendus teenusele orienteeritud arhitektuuride (SOA) veebiteenuste testimiseks ja olekuülekande (*REST*) vaatamiseks. Selle funktsioonid hõlmavad veebiteenuse testimist, käivitamist, arendamist, modelleerimist ja prototüüpimist, funktsionaalset testimist, koormuse ja vastavuse testimist [18].

SoapUI saab testida veebiteenuseid, nagu *SOAP* ja *REST*, *JMS*-i, samuti teha mistahes *HTTP(S)* ja *JDBC* järelpäringuid.

2.6.3 Selenium

Selenium on avatud lähtekoodiga veebirakenduste automatiseerimise töövahend. Seleniumit võib käivitada mitmes brauserites ja operatsioonisüsteemides, see toetab märkimisväärset arvu programmeerimiskeeli (*Java, .Net* ja palju teisi). Toetab ainult

veebikeskkonda (*web environment*). Aitab luua keerukaid ja uueaegseid automatiseerimisskripte [19].

See on aluseks enamiku muude tarkvara testimise tööriistade jaoks.

2.6.4 Cucumber

Cucumber on avatud lähtekoodiga *Behavior Driven Development* (BDD) tööriist, mis toetab mitut keelt (Ruby, Java, Scala, Groovy) ja toetab ainult veebikeskkonda. See oli loodud arvestades filosoofiaga, et tõsta lõppkasutajate kasutusmugavust [20].

Cucumber ei ole kasutajaliidese testimise tööriist, aga see saab töötada koos Seleniumiga. Enamik IT-ettevõtteid kasutab Seleniumi ja integreerib selle Cucumber'iga.

2.6.5 Selenide

Selenide on raamistik veebirakenduste automatiseeritud testimiseks, mis pakub mitmeid täiendavaid eeliseid, nagu AJAX-i tugi, lihtsam seadistamine. Selenide on Selenium WebDriveri mähis, mis võimaldab seda kiiresti ja lihtsalt kasutada testide kirjutamisel, keskendudes loogikale, mitte pidevale rutiinile brauseriga töötamisel [21].

Selenide pakub täiendavaid meetodeid toiminguteks, mida ei saa teha ühe Selenium WebDriveri käsuga. Need on raadionupu valimine, rippmenüüst elemendi valimine, ekraanipildi tegemine, brauseri vahemälu tühjendamine jne.

2.6.6 Automaattestide loomise vahendi valiku põhjendus

Autor on jõudnud järeldusele, et kõige sobivamad tööriistad on Selenium WebDriver ja Selenide. Ülejäänud tööriistad ei sobi ülesande täitmiseks: API testimiseks on eelistatav SoapUI, Cucumber on enamasti integreerimine Selenium ja Selenide jaoks, mitte kasutajaliidese testimise tööriist.

Pöördudes tagasi peatukis 2.6.1 automatiseerrimistöörista käsitlevate nõuete juurde, saab koostada Selenium и Selenide jaoks järgmine võrdlustaabel:

Tabel 2. Selenium и Selenide võrdlus.

Nõuded	**Selenium**	**Selenide**
Kokkusobivus Java-ga	+	+
Veebirakenduse testimine	+	+

Mitme sirvija ühilduvus	+	+
Tasuta tööriist	+	+
Elementide otsing teksti järgi/tekstiandmete kontroll	+	+
Automaatsete aruannete genereerimine	+	+

On näha, et põhiomadused on sarnased.

2.7 Selenium WebDriver ja Selenide võrdlus.

Antud peatükis teeb lõputöö autor üksikasjalikku võrdlust Selenium WebDriveri ja Selenide toimivate näidete kohta, et määrata, milline tööriist on taltech.ee kodulehe testimiseks kõige tõhusam.

Võrdluse läbiviimisel on olulised sellised aspektid, nagu testskriptide konfigureerimise lihtsus, brauseri juhtimine ja erandite käsitlemine. Olulised omadused on AJAX-i tugi, ekraanipiltide salvestamise võimalus, veebidraiveri automaatne seadistamine brauseri käivitamiseks ja juhtimiseks, võime kirjutada lihtsama ja ülevaatlikuma koodi [22].

Need omadused saavad kirjutamise automatiseerimise jaoks määravateks antud töö kolmandas peatükis.

2.7.1 Automaatne brauseri seadistamine

.Selenium WebDriver:

```
DesiredCapabilities desiredCapabilities =
DesiredCapabilities.htmlUnit();

desiredCapabilities.setCapability(HtmlUnitDriver.INVALIDS
ELECTIONERROR, true);

desiredCapabilities.setCapability(HtmlUnitDriver.INVALIDX
PATHERROR, false);

desiredCapabilities.setJavascriptEnabled(true);

WebDriver driver = new
HtmlUnitDriver(desiredCapabilities);
```

Selenide:

```
open("/my-application/login");
```

Järeldus: Selenium nõuab enne käivitamist täieliku brauseri konfiguratsiooni kirjutamist. Selliseid konfiguratsioone tuleb kirjutada iga testigrupi jaoks, mis toob kaasa suurt hulka samatüüpiliste koodide kirjutamist.

Selenide automaatselt konfigureerib veebidraiveri brauseri käivitamiseks ja juhtimiseks

2.7.2 Brauseri sulgemine pärast testimise lõppu

Selenium WebDriver:

```
if (driver != null) {
    driver.close();
}
```

Selenide: sulgeb automaatselt

Järeldus: Seleniumi testimise lõpus ei saa brauserit pärast testimise lõppu automaatselt sulgeda. Selenide sulgeb brauseri pärast testide lõpetamist automaatselt.

2.7.3 AJAX-i tugi

Selenium WebDriver:

```
FluentWait<By> fluentWait = new
FluentWait<By>(By.tagName("TEXTAREA"));

fluentWait.pollingEvery(100, TimeUnit.MILLISECONDS);

fluentWait.withTimeout(1000, TimeUnit.MILLISECONDS);

fluentWait.until(new Predicate<By>() {

    public boolean apply(By by) {

        try {

            return browser.findElement(by).isDisplayed();

        } catch (NoSuchElementException ex) {

            return false;

        }

    }

});

assertEquals("John",
browser.findElement(By.tagName("TEXTAREA")).getAttribute(
"value"));
```

Selenide:

```
$("TEXTAREA").shouldHave(value("John"));
```

Järeldus: Selenium vajab pikka koodi sündmuste ootamisega töötamiseks. Selenide'il on sisseehitatud ooteaeg elemendi otsimisel või sündmuse ootamisel, lisakoodi kirjutamine pole vajalik.

2.7.4 Elemendi leidmine teksti järgi

Selenium WebDriver: puudub.

Selenide:

```
WebElement customer = $(byText("Customer profile"));
```

Järeldus: Seleniumisse pole sisseehitatud tööriistu elemendi leidmiseks teksti järgi. Selenide'il on olemas sisseehitatud tööriistad elemendi leidmiseks teksti järgi.

2.7.5 Sisu puudumise kontrollimine

Selenium WebDriver:

```
try {

    WebElement element =
driver.findElement(By.id("customerContainer"));

    fail("Element should not exist: " + element);

}

catch (WebDriverException itsOk) {}
```

Selenide:

```
$("#customerContainer").shouldNot(exist);
```

Järeldus: Selenium nõuab erandolukordade töötlemist (*try-catch* plokk), kui elementi ei leita. Selenide omab sisseehitatud tööriistu elemendi puudumise kontrollimisel veebilehel.

2.7.6 Elemendi otsimine indeksi järgi:

Selenium WebDriver:

```
WebElement element =
driver.findElements(By.tagName("li")).get(5);
```

Selenide:

```
$("li", 5);
```

Järeldus: Seleniumil on mahukas liides elemendi otsimisel indeksite järgi. Selenide omab lakoonilist liidest elemendi otsimist indeksi järgi.

2.7.7 Töötamine dialoogaknaga

Selenium WebDriver:

```
    try {

      Alert alert =
checkAlertMessage(expectedConfirmationText);

      alert.accept();

    } catch (UnsupportedOperationException
alertIsNotSupportedInHtmlUnit) {

      return;

    }

    Thread.sleep(200); // sometimes it will fail
```

Selenide:

```
    confirm("Are you sure to delete your profile?");

or

    dismiss("Are you sure to delete your profile?");
```

Järeldus: Seleniumil on dialoogikastiga töötamisel mahukas liides. Selenide'il on dialoogikastiga töötamisel lakoonilised tööriistad.

2.7.8 Ekraanipiltide tegemine:

Selenium WebDriver:

```
    if (driver instanceof TakesScreenshot) {

      File scrFile = ((TakesScreenshot)
webdriver).getScreenshotAs(OutputType.FILE);

      File targetFile = new File("c:\temp\" + fileName +
".png");

      FileUtils.copyFile(scrFile, targetFile);

    }
```

Selenide:

```
    takeScreenShot("my-test-case");
```

või JUnit -iga

```
public class MyTest {

  @Rule // automatically takes screenshot of every
failed test

  public ScreenShooter makeScreenshotOnFailure =
ScreenShooter.failedTests();

}
```

Järeldus: Seleniumil pole ekraanipiltide salvestamiseks valmis tööriistu, arendaja peab ekraanipildi loomiseks otse draiveriga ühendust võtma ja seejärel ekraanipildi faili käsitsi looma. Selenide'il on sisseehitatud tööriist ekraanipilti salvestamiseks, lisaks Junit kasutamisel on võimalik annotatsiooni @Rule kasutamine ekraanipiltide lakooniliseks konfiguratsiooniks.

2.7.9 Praeguse lehe värskendamine

Selenium WebDriver:

```
webdriver.navigate().to(webdriver.getCurrentUrl());
```

Selenide:

```
refresh();
```

Järeldus: Seleniumil on mahukas liides praeguse lehe värskendamisel. Selenide-l on lakooniline liides praeguse lehe värskendamisel.

2.8 Automaattestide loomise vahendi valiku põhjendus

Ülaltoodud omaduste põhjal teeb autor valiku Selenide kasuks. Võrdlev analüüs näitas, et Selenide aitab luua dünaamilise sisuga veebirakendustele stabiilseid teste, lahendades enamiku ajalõppude ja Ajaxiga seotud probleeme. Selenide'is määratakse ebamäärane ooteaeg vaikimisi 4 sekundit (konfiguratsioonis saab hõlpsasti konfigureerida mistahes muu ajalõpu väärtust). Kui leht kohe ei lae, Selenide peatub enne seda, et tunnistada test testi ebaõnnestunuks.

Selenide'il on võrreldes Seleniumiga lihtsam ja lakoonilisem kood, mis ei nõua pika koodi kirjutamist.

Selenide'is on sisseehitatud tööriist ekraanipiltide salvestamiseks ja elemendi otsimiseks teksti järgi.

Selenide konfigureerib veebidraiveri brauseri käivitamiseks ja juhtimiseks ning automaatselt sulgeb brauseri pärast testide lõppu.

Selenide võimaldab kirjutada kvaliteetseid automatiseeritud teste, mis vastab eesmärgile, mis tagab selle, et muuta testimisprotsessi võimalikult efektiivseks ja usaldusväärseks.

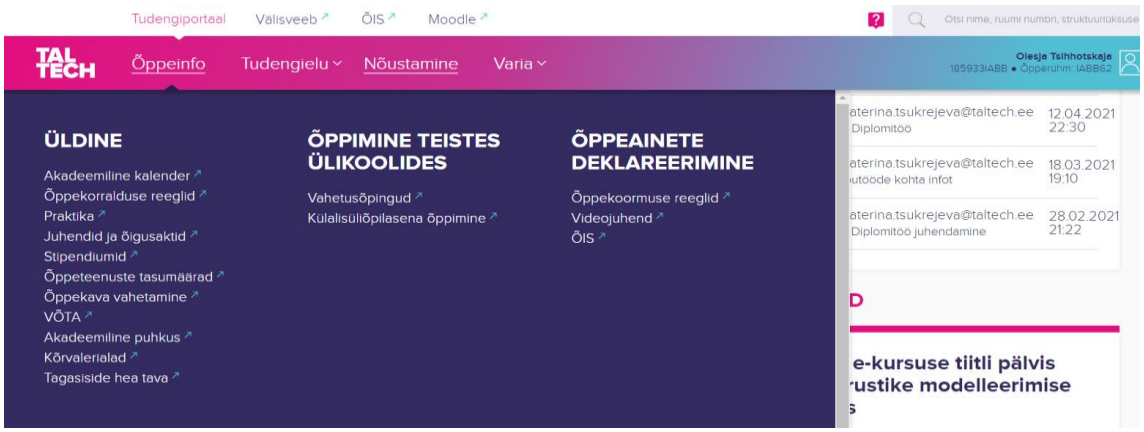
3 Kasutajaliidese testimine

Käesolevas peatükis testitakse veebilehte https://student.taltech.ee/ Selenium ja Selenide tööriistade abil, et tegelikult võrrelda toote testimisrakendusi. Automatiseerimine toimub mõlema tööriista abil.

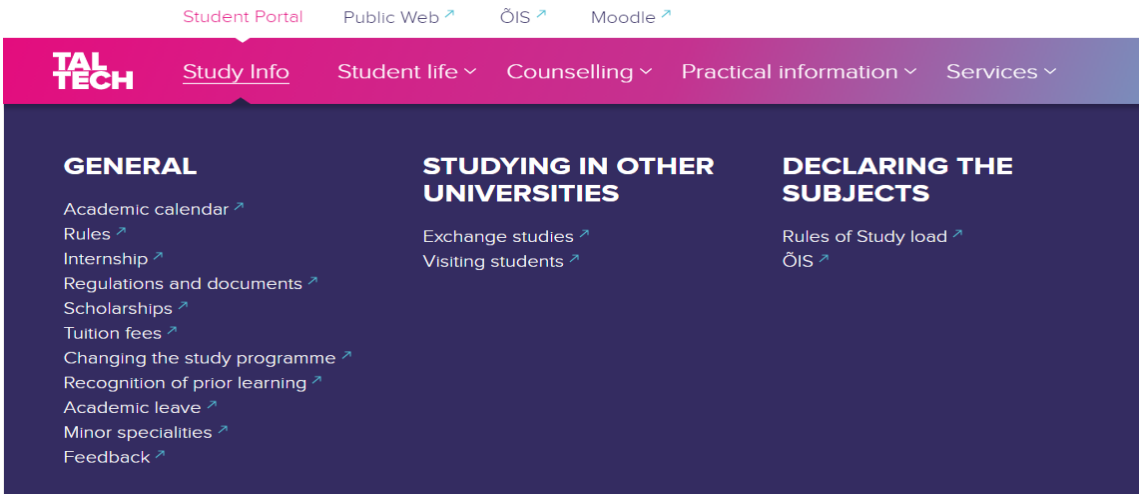
Testitulemuste visuaalse esitamise mugavuse huvides täiendavalt kasutatakse autotesti aruannete genereerimise tööriista Allure raamistikku, mis lihtsustab nende edasist analüüsi. Allure automaatselt genereerib testi teostamise kokkuvõtte.

3.1 Tudengiportaali infosüsteemi funktsionaalsed nõuded ja kasutaja teststsenaarium

Tudengiportaali kasutajal peab olema võimalus otsida teavet veebilehelt, järgida linke, pääseda juurde ÕIS-ile, e-postile, kalendrile, raamatukogule ja muule. Foto all saab näha, kuidas välja näeb Tudengiportaali peamenüü:



Joonis 2. Näide. Eestikeelne menüü



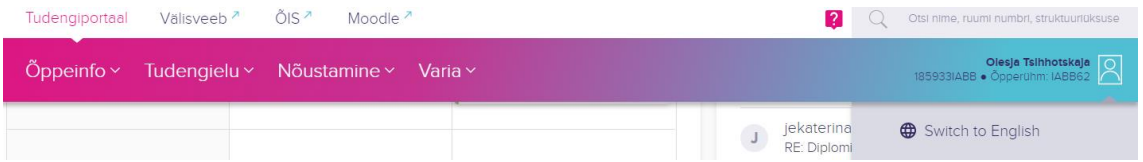
Joonis 3. Näide: Inglisekeelne menüü.

Selles töös testitakse põhimenüüd „Õppeinfo“ veebilehe eestikeelse versiooni jaoks ja Study info menüüd Tudengiportaali infosüsteemi ingliskeelse versiooni jaoks.

Automaattestidega kontrollitavad funktsionaalsed nõuded on järgmised:

Registreeritud kasutaja peaks nägema hüpikakent keele muutmiseks klõpsates kasutaja profiili nuppu;

Kui kasutaja vajutab keele valiku nuppu, tuleks keel muuta vastupidiseks (kui kasutaja on eestikeelses režiimis, muutub keel inglise keeleks ja vastupidi);



Joonis 4. Näide: Keele lülitamise aken.

Kui registreeritud kasutaja on eestikeelses versioonis, klõpsates nuppu „Õppeinfo“ peamenüü näeb ta rippmenüüd, milles on lingid akadeemiline kalender, õppekorralduse reeglid, praktika ja muud vastavad asjakohased lingid (kõigi alammenüü linkide täielik loend on Joonisel. Eestikeelne menüü);

Kui registreeritud kasutaja on inglisekeelses versioonis, klõpsates nuppu „Study Info“ peamenüü näeb ta rippmenüüd, kus on lingid akadeemilisele kalendrile, reeglitele,

praktikale ja nii edasi, õppekorralduse reeglitele, praktikale ja muudele asjakohastele linkidele (kõigi alammenüü linkide täielik loend on Joonisel. Inglisekeelne menüü);

Kui registreeritud kasutaja on „Õppeinfo“ menüü eestikeelses versioonis, siis ta saab klõpsata lingil akadeemiline kalender, mille järel avaneb kalendriga brauseriaken (sarnased nõuded kehtivad kõigi Õppeinfo menüü linkide kohta);

Kui registreeritud kasutaja on „Study Info“ menüü inglisekeelses versioonis, siis ta saab klõpsata lingil academic calender, mille järel avaneb kalendriga brauseriaken (sarnased nõuded kehtivad kõigi „Study Info“ menüü linkide kohta).

Automatiseerimist kirjutatakse ainult „Õppeinfo“ nupule rippmenüüga , kuna ülejäänud peamenüü nuppude automatiseerimine on absoluutselt identne.

Samuti autor ei kavatse automatiseerida sisselogimist, kuna tudengiportaal kasutab kaheastmelist autentimist SMS-i kaudu. Kahefaktorilise autentimise keelamiseks või testkasutaja profiili saamiseks pole töövõtjaga võimalik ühendust võtta. Autor testib süsteemi otse oma isikliku konto abil.

3.2 Testjuhtumite loend

On koostatud testjuhtumite tabel. Iga test saab keelt parameetri kujul, milles seda testitakse, ja teksti, mille alusel otsitakse element „Õppeinfo“ menüüs, et sellele hiljem klõpsata.

Kõigi testide puhul kehtib ühine eeldus: eesti keele režiimis muuta keel inglise keeleks või vastavalt inglise keele režiimis muuta keel eesti keeleks.

Tabel 3. Testjuhtumite teostamise tulemuste tabel.

Testi nr.	**Tegevus**	**Oodatut tulemus**	**Testi tulemus**
1	Klõpsata „Academic Calendar menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK
2	Klõpsata „Akadeemiline kalender menüü elemendil peamenüüst „Õppeinfo	Vaheleht on edukalt avatud	OK

3	Klõpsata „Rules menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK
4	„Õppekorralduse menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK
5	Klõpsata „Internship menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK
6	Klõpsata „Praktika menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK
7	Klõpsata „Regulations and documents“ menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK
8	Klõpsata „Juhendid ja õigusaktid“ menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK
9	Klõpsata „Scholarships „menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK
10	Klõpsata „Stipendiumid“ menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK
11	Klõpsata „Tuition fees“ menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK
12	Klõpsata „Õppeteenuste tasumäärad“ menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK
13	Klõpsata „Changing the study programme” menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK
14	Klõpsata „Õppekava vahetamine“ menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK
15	Klõpsata “Recognition of prior learning” menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK

16	Klõpsata „VÕTA“ menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK
17	Klõpsata „Academic leave menüü“ elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK
18	Klõpsata „Akadeemiline puhkus“ menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK
19	Klõpsata „Minor specialities“ menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK
20	Klõpsata „Kõrvalerialad“ menüü elemendil peamenüüst Õppeinfo	Vaheleht on edukalt avatud	OK
21	Klõpsata „Feedback“ menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK
22	Klõpsata „Tagasiside hea tava“ menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK
23	Klõpsata „General information“ menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	Ebaõnnestunud. „General Information“ nuppu ei leitud, pealkiri on valesti kirjutatud.
24	Klõpsata „Üldinfo“ menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK
25	Klõpsata „Timetable“ menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK
26	Klõpsata Treeningkalender menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK
27	Klõpsata „Exchange studies“ menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK
28	Klõpsata „Vahetusõpingud“ menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK
29	Klõpsata „Visiting students“ menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK

30	Klõpsata „Külalisüliõpilasena õppimine“ menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK
31	Klõpsata „Rules of Study load“ menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	OK
32	Klõpsata : Õppekoormuse reeglid“ menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK
33	Klõpsata „Video guide“ menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	Ebaõnnestnud. Menüü element “Video guide“ puudub
34	Klõpsata „Videojuhend“ menüü elemendil peamenüüst Õppeinfo„Õppeinfo“	Vaheleht on edukalt avatud	OK
35	Klõpsata „Study Information System“ menüü elemendil peamenüüst „Study Info“	Vaheleht on edukalt avatud	Ebaõnnestunud. Veebirakenduses kasutatakse eestikeelset lühendit ÕIS
36	Klõpsata „ÕIS“ menüü elemendil peamenüüst „Õppeinfo“	Vaheleht on edukalt avatud	OK

3.3 Automatiseerimisprotsessi ülevaade. Selenide'i ja Selenium'i realiseerimiste võrdlus

3.3.1 Brauseri konfigureerimine enne testide tööd:

Testbrauseriks oli valitud kõige populaarsem brauser - Chrome. Chrome'i brauseri tuum on paljude teiste brauserite aluseks ning sellel on ka palju tööriistu silumiseks ja arendamiseks.

Selenide ja Selenium'i üldine brauseri seadistamine:

```
@NotNull
private static ChromeOptions getChromeOptions() {
   ChromeOptions options = new ChromeOptions();
   options.addArguments("start-maximized");
   options.addArguments("--allow-profiles-outside-user-
dir");
   options.addArguments("--user-data-
dir=C:\\Autotests\\Selenide\\Students");
   options.addArguments("--profile-
directory=TestProfile");
   return options;
}
```

Joonis 5. Näide: Meetod getChromeOptions().

Meetod. getChromeOptions(): Testimisprotsessi hõlpsaks jälgimiseks kohandub brauser maksimaalse akna suurusega. Testimiseks kasutatakse eraldi Chrome'i brauseri profiili. See oli loodud ja teisaldatud kausta C:\Autotests\Selenide\Students.

Vaikimisi brauser ei saa töötada profiiliga, mis asub väljaspool põhiprofiilide baaskataloogi, seega kasutatakse väliste profiilide kasutamise lubamiseks järgmist argumenti: "--allow-profiles-outside-user-dir“.

Testimise puhtuse jaoks on vaja eraldi profiili, et mitte rikkuda süsteemi peamise kasutajaprofiili andmeid ja ka automaatse testimise iseärasuste tõttu: konkreetse profiili brauser tuleb enne automaatsete testide alustamist sulgeda, muidu testid ei käivitu.

Selenide:

```
public static class TaltechFactory extends
ChromeDriverFactory {
   @Override
   @CheckReturnValue
   @Nonnull
   public MutableCapabilities createCapabilities(Config
config, Browser browser, @Nullable Proxy proxy, File
browserDownloadsFolder) {

      return new
MergeableCapabilities(getChromeOptions(),
createCommonCapabilities(config, browser, proxy));
   }
}
```

```
@BeforeAll
static void beforeAll() {


   // code example: https://javabydeveloper.com/junit-5-
with-allure-reports-example/
   SelenideLogger.addListener("AllureSelenide", new
AllureSelenide().screenshots(true).savePageSource(true));
   Configuration.browser =
TaltechFactory.class.getName();
   Configuration.startMaximized = true;
   Configuration.timeout = 20000;
}
```

Joonis 6. Näide: Draiveri konfiguratsioon. Selenide.

Klass TaltechFactory: brauseri konfiguratsiooni konfigureerimiseks automaatrežiimis.

Meetod createCapabilities(): Kohandame brauserit spetsiaalse testimisprofiili jaoks [23].

Meetod beforeAll(): Seadistab ekraanipilte. Vigade ilmnemisel salvestab Selenide ekraanipilte automaatselt.

Vastavalt antud konfiguratsioonile brauser jääb alati laiendatud versioonis täisekraanil.

Aeg möödub (time out) 20 sekundini, kuna veebilehe laadimiseks ei piisa tavapärasest 4 sekundist [24].

Selenium:

```
@BeforeAll
static void beforeAll() {

   System.setProperty("webdriver.chrome.driver",
"C:\\Autotests\\chromedriver.exe");
   ChromeOptions options = getChromeOptions();
   driver = new ChromeDriver(options);
   driver.manage().window().maximize();
   driver.manage().timeouts().implicitlyWait(20,
TimeUnit.SECONDS);
```

Joonis 7. Näide: Meetod beforeAll(). Draiveri konfiguratsioon. Selenium

.

```
@AfterAll
static void afterAll() {
   driver.quit();
}
```

Joonis 8. Näide: Meetod afterAll(). Browseri kinnipanemine. Selenium

Meetod beforeAll(). Selenium nõuab, et draiveri tee oleks määratud sõnaselgelt. Draiveri konfiguratsiooni tuleb luua käsitsi. Vastavalt antud konfiguratsioonile jääb brauser samuti

täisekraanile ja elementide ajalõpu ooteaeg on seatud 20 sekundiks. Meetod afterAll sulgeb brauserit.s

Järeldus: Selenium'i draiveri konfiguratsiooni tuleb luua käsitsi ja brauseridraiveri tee peab olema selgesõnaliselt määratud ning Selenide teeb seda automaatselt. Samuti peab pärast kõigi testide teostamist sulgema brauserit käsitsi, see ei juhtu automaatselt nagu Selenide'is. Juba draiveri seadistamise etapis võib väita, et võrreldes Selenium'iga on Selenide paindlikum ja mugavam

3.3.2 Veebilehe konkreetse keele konfigureerimise meetod:

Töö algoritm: testi käivitamise ajal avaneb veebilehe https://student.taltech.ee avaleht, valitakse veebilehe esitluse keel. Veebilehel on profiili nuppu ja keele muutmise nuppu tähistavad järgmised HTML-i nimed:

```
static String profileButton = ".tt-profile__img";
static String languageSwitchButton = ".tt-
profile__link:nth-child(1) .btn__text";
```

Joonis 9. Näide: Profiilinuppude ja keele muutmise nuppude üldine konfiguratsioon

.

Keele muutmiseks peab klõpsama nuppu profiil, mille järel ilmub plokk, kus on keele muutmise nupp. Profiilile klõpsates on näha, millisele keelele saab üle minna. (Identne Selenide ja Seleniumi jaoks).

ClickProfileAndGetLanguageSwitchText(). Keele muutmise nupu visualiseerimise meetod.

Selenide:

```
@NotNull
public static String
ClickProfileAndGetLanguageSwitchText() {

    $(profileButton).click();
    return $(languageSwitchButton).getText();
}
```

Joonis 10. Näide: Meetod ClickProfileAndGetLanguageSwitchText. Selenide.

Vajutatakse profiilinuppu ja tagastatakse keele vahetamise nupu tekst. Piisab nupuvalija määramisest. Vaikimisi tehakse tööd CSS-i valijatega.

Selenium:

```
@NotNull
public static String
ClickProfileAndGetLanguageSwitchText(WebDriver driver) {

driver.findElement(By.cssSelector(profileButton)).click()
;
   return
driver.findElement(By.cssSelector(languageSwitchButton)).
getText();
}
```

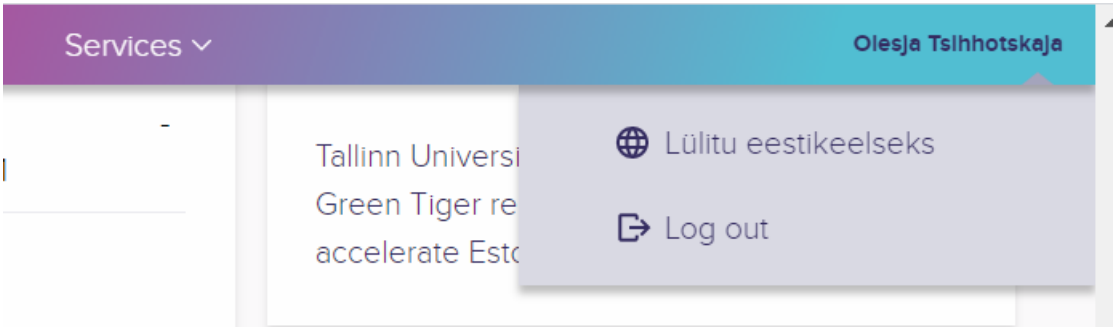
Joonis 11. Näide: Meetod ClickProfileAndGetLanguageSwitchText. Selenium

Sama Selenium'i meetod keele vahetamiseks peab määrama valijatega teed otsast lõpuni. Peab kirjutama palju rohkem koodi.

SwitchLanguageIfCurrentNotCorrece(). Keele vahetus meetod

Kuna veebileht töötab inglise ja eesti keeles, on vaja meetodit veebilehe keele vahetamiseks.

Kui on soov olla inglise keelses režiimis, peaks keelevahetuse aknas kuvama vastupidist keelt.



Joonis 12. Näide Keele vahetus nupp. Tudengiportaal.

Kui kuvatakse, et vahetuskeel langeb kokku eeldatava keelega, peab veebilehe keelt muutma. Meetod SwitchLanguageIfCurrentNotCorrece muudab keelt, kui see ei sobi.

Selenide:

```
public static void
SwitchLanguageIfCurrentNotCorrect(String
shouldBeLanguageSwitchText) {

   String actualLanguageSwitchText =
ClickProfileAndGetLanguageSwitchText();
   if
(!actualLanguageSwitchText.equals(shouldBeLanguageSwitchT
ext))
   {
      $(languageSwitchButton).click();
   }
}
```

Joonis 13. Näide: Keele vahetus meetod.Selenide.

Meetod SwitchLanguageIfCurrentNotCorrece() muudab keelt, kui see pole sobiv. Vajutatakse profiilinuppu ja loetakse keelenupu teksti. Klõpsame profiilil ja loeme nupu teksti, et vahetada keelt. IF teostab keelekontrolli, see tähendab, et kui oodatav tekst ja praegune pole samad, siis peab muutma veebilehe keelt. Lühike käsk valijale, klõpsatakse keele vahetamise nupul.

Selenium:

```
public static void
SwitchLanguageIfCurrentNotCorrect(WebDriver driver,
String shouldBeLanguageSwitchText) {
   String actualLanguageSwitchText =
ClickProfileAndGetLanguageSwitchText(driver);
   if
(!actualLanguageSwitchText.equals(shouldBeLanguageSwitchT
ext))
   {

driver.findElement(By.cssSelector(languageSwitchButton)).
click();
   }
}
```

Joonis 14. Näide: Keele vahetus meetod.Selenium

Sarnane Seleniumi meetod SwitchLanguageIfCurrentNotCorrece(): koos täiendava parameetriga see aktsepteerib draiverit ja jällegi on vaja kogu tee SCC-valija abil kirjutada. IF kontroll on identne IF Selenide'iga kontrollile.

Meetod find(). Elemendi otsing veebilehel

Pärast veebilehe keele konfigureerimist peab avama tudengi teabe menüüd ja valima konkreetse menüüelementi ning seejärel sellel klõpsama eraldi vahelehe avamiseks. Kui nupp ei ole menüüst leitav, loetakse test ebaõnnestunuks.

Selenide:

```
ublic static By find(String language, String text) {

   open("https://student.taltech.ee/");

   switch (language) {
      case "ENGLISH": {
         String shouldBeLanguageSwitchText = "Lülitu
eestikeelseks";

SwitchLanguageIfCurrentNotCorrect(shouldBeLanguageSwitchT
ext);
         $(withText("Study Info")).click();
         break;
      }
      case "ESTONIAN": {
         String shouldBeLanguageSwitchText = "Switch to
English";

SwitchLanguageIfCurrentNotCorrect(shouldBeLanguageSwitchT
ext);
         $(withText("Õppeinfo")).click();
         break;
      }
   }
   return withText(text);
}
```

Joonis 15. Näide. Meetod find(). Selenide.

.

Sisestusmeetod aktsepteerib parameetrina keelt, milles peab asuma leitava elemendi teksti, mida peab leidma. OPEN meetod kuvab konkreetset linki veebilehe avalehele. Switch – case teostab valitud keele põhjal kontrolli, kasutades keele vahetamise nupul eeldatavat teksti. Vajadusel vahetatakse keelt. Seejärel klõpsata peamisel testimenüül Study Info. Pärast Study Info menüü avamist peab elementi selle alammenüüst üles leidma ja tagastama.

Selenium:

```
public static WebElement find(WebDriver driver, String
language, String text) {

   driver.get("https://student.taltech.ee/");

   switch (language) {
      case "ENGLISH": {
         String shouldBeLanguageSwitchText = "Lülitu
eestikeelseks";
         SwitchLanguageIfCurrentNotCorrect(driver,
shouldBeLanguageSwitchText);
         driver.findElement(byXPATH("Study
Info")).click();
         break;
      }
      case "ESTONIAN": {
         String shouldBeLanguageSwitchText = "Switch to
English";
         SwitchLanguageIfCurrentNotCorrect(driver,
shouldBeLanguageSwitchText);
         driver.findElement(byXPATH("Õppeinfo")).click();
         break;
      }
   }
   return driver.findElement(By.linkText(text));
}

 NotNull
static By byXPATH(String text) {
   return By.xpath("//*[text()='" + text + "']");
}
```

Joonis 16. Näide. Meetod find() lisa meetodiga byXPATH(). Selenium.

Selenium'i meetod find () on kirjutatud identselt, ainsa erinevusega, et lühikese $ (leia element) asemel peab iga kord brauserile juurde pääsema robustse draiveri driver.findElement(By()) abil. Lisaks peaks kirjutama meetodit byXpath(), mis otsib teksti HTML-koodi sees. Sisestusmeetod aktsepteerib teksti ja tagastab veebilehe elementi.

Kood on palju pikem. Lingi avamiseks pole vaja küsida ainult konkreetset draiverit, vaid ka nupu otsimiseks on uurida vaja kogu HTML-koodi.

Nupu otsimine toimub alamseade otsimisega HTML-teksti plokkidest.

3.3.3 Põhiparameetriline test:

Selle testiga kontrollitakse Õppeinfo peamenüü elementide kättesaadavust ja õigsust. Leitud elemendid on nupud teistele veebilehe jagudele üleminekuks. Antud nupud klõpsatakse automaatselt teisele vahelehele üleminekuks. Testi parameetrid

genereeritakse tabelina CSV-vormingus. Parameetrite abil saab ühe testi põhjal genereerida 36 erinevat testi (iga keele jaoks 18). See võimaldab läbida kõik menüüelemendid mõlemas keeles. Parameetritena aktsepteeritakse menüüelemendi keelt ja teksti. Selle lähenemisviisi abil saab kirjutada vähem koodi testide jaoks. Sisendandmed moodustavad vajaliku arvu teste (testjuhtumite täielik loetelu, tabeli number X). Junit'i parameetriga testi põhiosa on Selenide ja Selenium puhul sama:

```
/ https://www.baeldung.com/parameterized-tests-junit-5
   @ParameterizedTest(name = "{index} {0} : {1}")
   @CsvSource(
         {
               "ENGLISH,Academic calendar",
               "ESTONIAN,Akadeemiline kalender",

               "ENGLISH,Rules",
               "ESTONIAN,Õppekorralduse reeglid",

               "ENGLISH,Internship",
               "ESTONIAN,Praktika",

               "ENGLISH,Regulations and documents",
               "ESTONIAN,Juhendid ja õigusaktid",

               "ENGLISH,Scholarships",
               "ESTONIAN,Stipendiumid",

               "ENGLISH,Tuition fees",
               "ESTONIAN,Õppeteenuste tasumäärad",

               "ENGLISH,Changing the study programme",
               "ESTONIAN,Õppekava vahetamine",

               "ENGLISH,Recognition of prior learning",
               "ESTONIAN,VÕTA",

               "ENGLISH,Academic leave",
               "ESTONIAN,Akadeemiline puhkus",

               "ENGLISH,Minor specialities",
               "ESTONIAN,Kõrvalerialad",

               "ENGLISH,Feedback",
               "ESTONIAN,Tagasiside hea tava",

               "ENGLISH,General information",
               "ESTONIAN,Üldinfo",

               "ENGLISH,Timetable",
               "ESTONIAN,Treeningkalender",

               "ENGLISH,Exchange studies",
               "ESTONIAN,Vahetusõpingud",

               "ENGLISH,Visiting students",
               "ESTONIAN,Külalisüliõpilasena õppimine",

               "ENGLISH,Rules of Study load",
               "ESTONIAN,Õppekoormuse reeglid",

               "ENGLISH,Video guide",
               "ESTONIAN,Videojuhend",

               "ENGLISH,Study Information System",
               "ESTONIAN,ÕIS"
         })
```

Joonis 17. Näide. Junit 5 põhiparameetriline test.

Iga testi pealkiri sisaldab elemendi seerianumbrit, keelt ja teksti. Kõik see liidab testi nime teostamise ajal. Parameetreid kirjeldatakse tabelina CSV-vormis.

Meetod MenuTest(). Testide läbiviimis meetod:

Selenide:

```
public void MenuTest(String language, String text) {

    $(find(language, text)).click();
}
```

Joonis 18. Näide. Meetod MenuTest(). Selenide.

Meetod MenuTest() võtab keelt ja teksti vastu parameetrina. Koodi kogus on minimaalne. Piisab, kui kutsuda eelnevalt ettevalmistatud otsingumeetodi menüü element ja sellel klõpsata.

Selenium:

```
public void MenuTest(String language, String text) throws
IOException {

    try {
        find(driver, language, text).click();
    }
    catch (Exception exception)
    {
        Allure.addAttachment("Element not found!", new
ByteArrayInputStream(((TakesScreenshot)
driver).getScreenshotAs(OutputType.BYTES)));
        throw exception;
    }
```

Joonis 19. Näide: Meetod MenuTest() koos meetodiga afterAll(). Selenium.

.

Kuna vea ajal ekraanipiltide tegemiseks pole vaikimisi sätet, peab täiendavalt kirjutama plokki „try catch“, mis oluliselt suurendab koodi hulka. Plokis “try catch” otsitakse ja klõpsatakse nuppu, tekst ja konkreetne keel määratakse parameetrite järgi. Draiverit edastatakse ka parameetrina, kuna see pole automatiseeritud testide sooritamise ajal globaalselt nähtav (Selenide omakorda ise valib draiverit). Viidates jaole Allure, seatakse käsk lisada testitulemusele ekraanipildi kujul rakendus. Meetodi lõpus uuesti saadame erandit (throw exception), nii et testi ei loetaks ebaõnnestunuks.

Selenium ei oska iseseisvalt sulgeda brauserit pärast enda tööd, seetõttu on vaja täiendavat meetodit afterAll(), mis käivitub pärast kõigi teiste meetodite käivitamist ja sulgeb draiverit.

Meetod tearDown().Testi töö lõpetamine: Testi ajal võivad menüü nuppude vajutamisel avaneda uued brauseri vahelehed. Selleks, et automatiseerimistööriistad jätkaksid töötamist peamisel veebilehel ja mitte ainult avatud vahelehel, peab naasma põhilehele. Põhilehele üleminek peaks toimuma pärast testi.

Selenide:

```
@AfterEach
public void tearDown() {

    Selenide.switchTo().window(0);
}
```

Joonis 20. Näide: Testi töö lõpetamine. Selenide.

On väga lihtne valida konkreetne vaheleht, peab tagasi minema peaaknasse.

Selenium:

```
@AfterEach
public void tearDown() {

    ArrayList<String> tabs = new ArrayList<String>
(driver.getWindowHandles());
    driver.switchTo().window(tabs.get(0));
}
```

Joonis 21. Näide: Meetod tearDown().Testi töö lõpetamine. Selenium.

Peab valima kogu vahelehtede loendit ja leidma vajalikku nime järgi. Kogutakse kõigi brauseri vahelehtede loend. Vahelehtede loendi esimene element on peamine testitav vaheleht ja toimub sellele üleminek.

Järeldus: Selenium nõuab natuke rohkem kodeerimist. Ühe parameetrilise testi raames võivad need erinevused tunduda ebaolulistena, kuid kui peab kirjutama suure hulga unikaalseid teste. Probleemideks võivad olla näiteks erandite käsitsemine järgmise ekraanipildiga, draiveri uuesti seadistamine, draiveri käsitsi otsimine. Kõik see viib koodi koguse suurenemiseni, mis ei ole testi olemuseks.

Sellist vajalikku koodi, mida ei loeta äriloogikaks, nimetatakse “Boilerplate”-koodi fragmendiks, mida korratakse mitmes kohas muudatusteta või minimaalsete

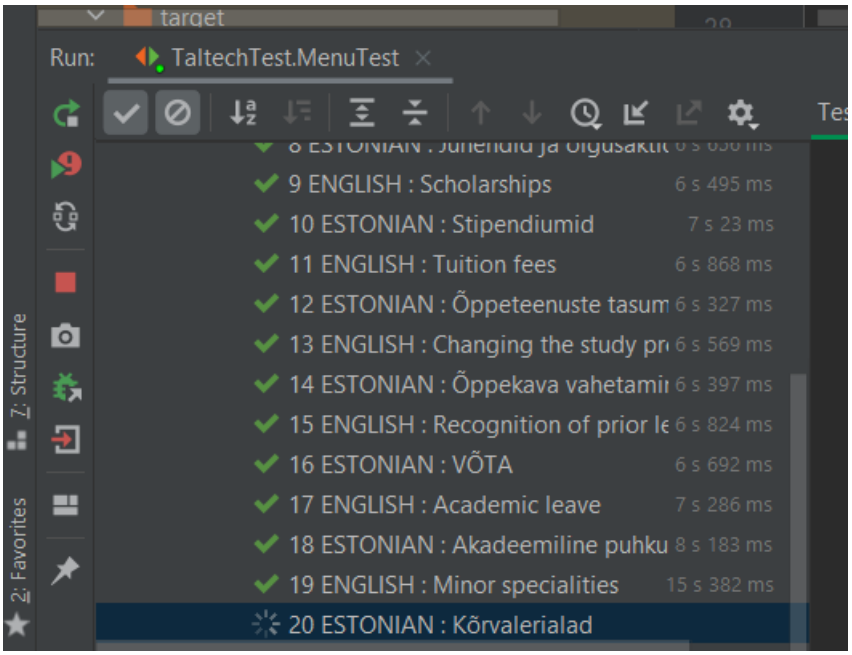
muudatustega, programmeerija peab kirjutama suurt koodi, et täita ainult väiksemaid funktsioone [25]. Äriloogika on testide jaoks oluline ja Boilerplate'i kood segab teste.

Selenide on parem ja pakub rohkem võimalusi, see võimaldab keskenduda äriloogikale.

3.4 Testimisprotsess ja tulemused.

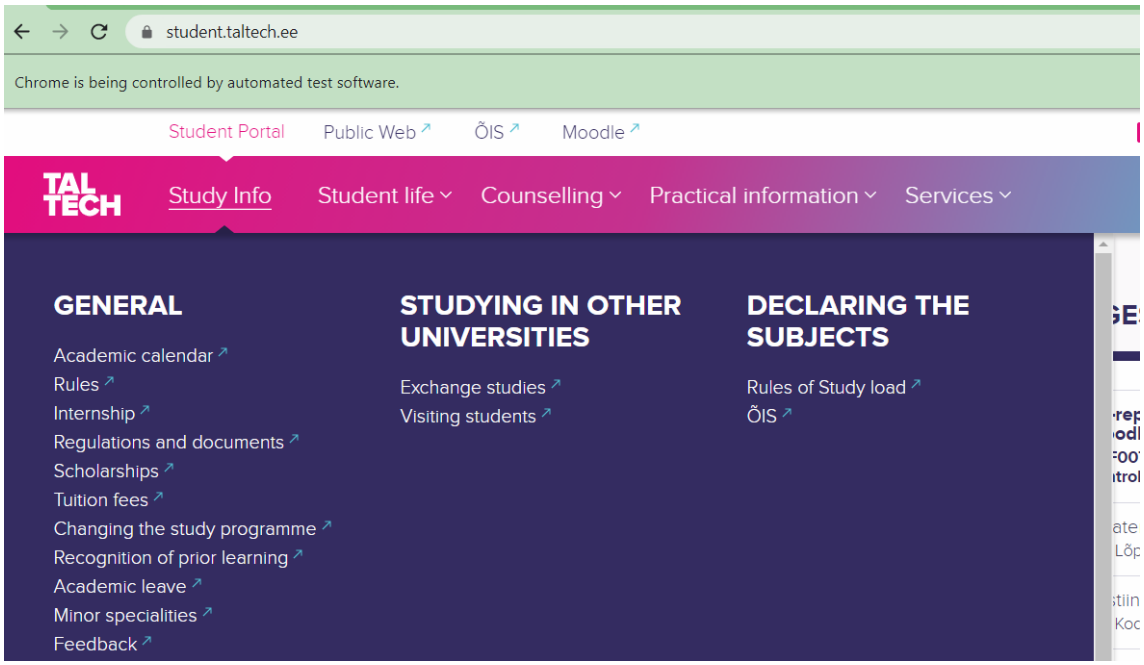
Testimisprotsess on nii Selenide'i kui ka Selenium'i puhul sama.

Junit genereerib testi käivitamist:



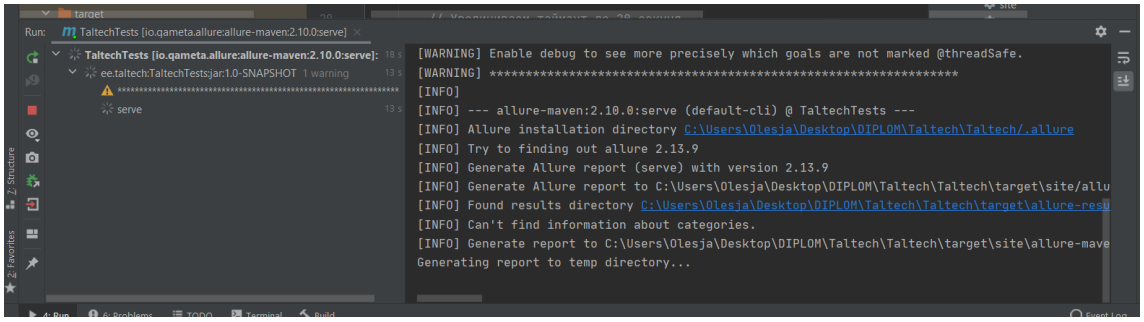
Joonis 22. Näide: Jooksvad testid, IntelliJ.

Kasutajaliidese (UI) testimine viiakse läbi automaatrežiimis. Õppeinfo peamenüü nuppe (alammenüü vahelehed) vajutatakse kordamööda. Iga alammenüü kontrollimisel muutub versiooni keel vaheldumisi eesti keelest inglise keelde, läbides kogu menüü vahelehtede loendit (see tähendab järgmises järjekorras, kontrollides vahelehte Õppeteave, Õppeinfo, Reeglid, Õppekorralduse reeglid, Praktika, Praktika ja nii edasi kuni vahelehe Õppeinfo loendi lõpuni):



Joonis 23. Näide: Jooksvad testid browseris.

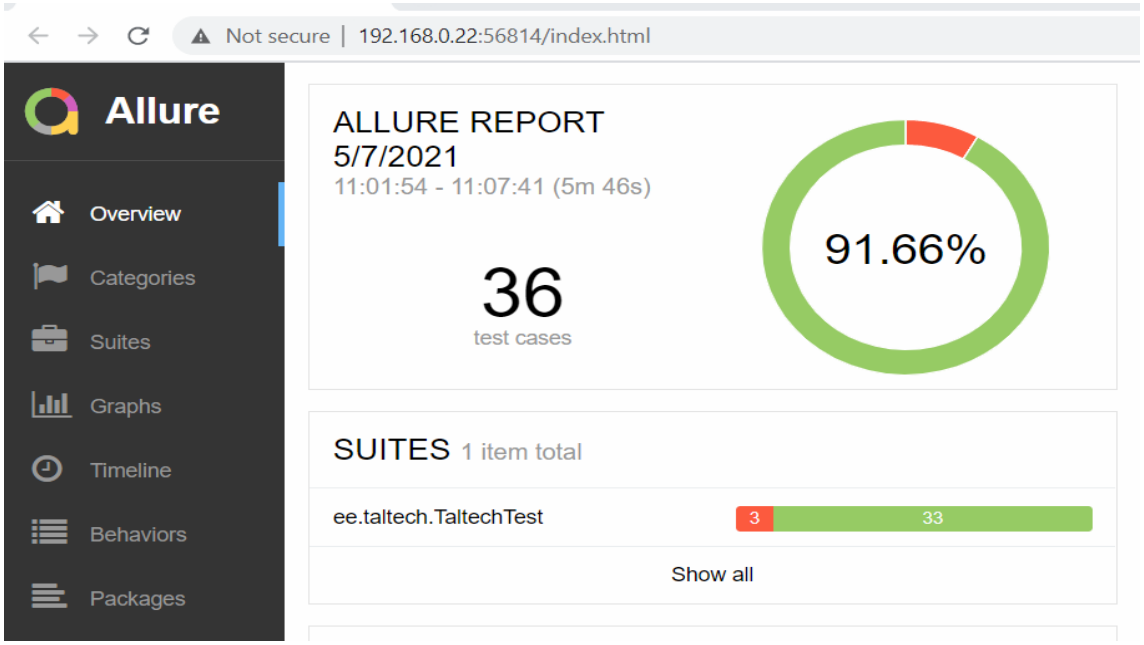
Testiaruannete genereerimist koostatakse Allure'i põhjal:



.Joonis 24. Näide: Allure Testiaruannete genereerimine.

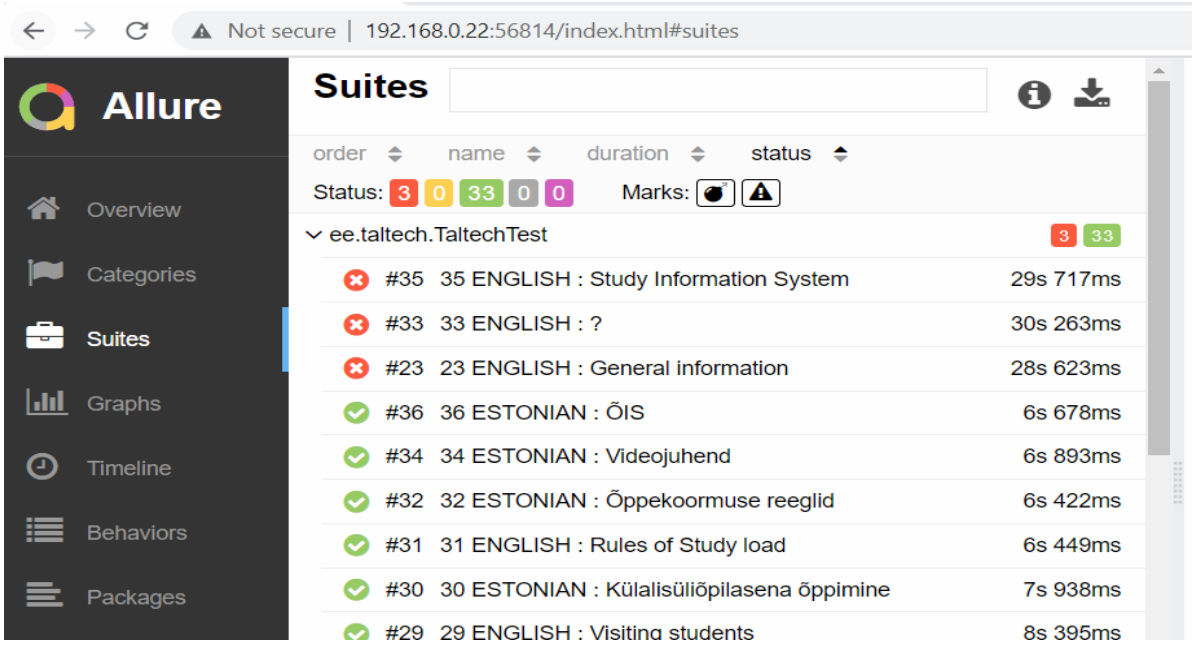
Pärast aruande genereerimist näitab Allure tehtud testide kohta üldist teavet, näidates edukalt läbitud ja ebaõnnestunud testjuhtumite arvu.

Aruandes (Overview) saab näha läbitud testide koguarvu, milleks on 36, edukate testide arvu (33) ja ebaõnnestunud testide arvu (3). Testid on sorteeritud kukkunud-läbitud staatuse järgi:

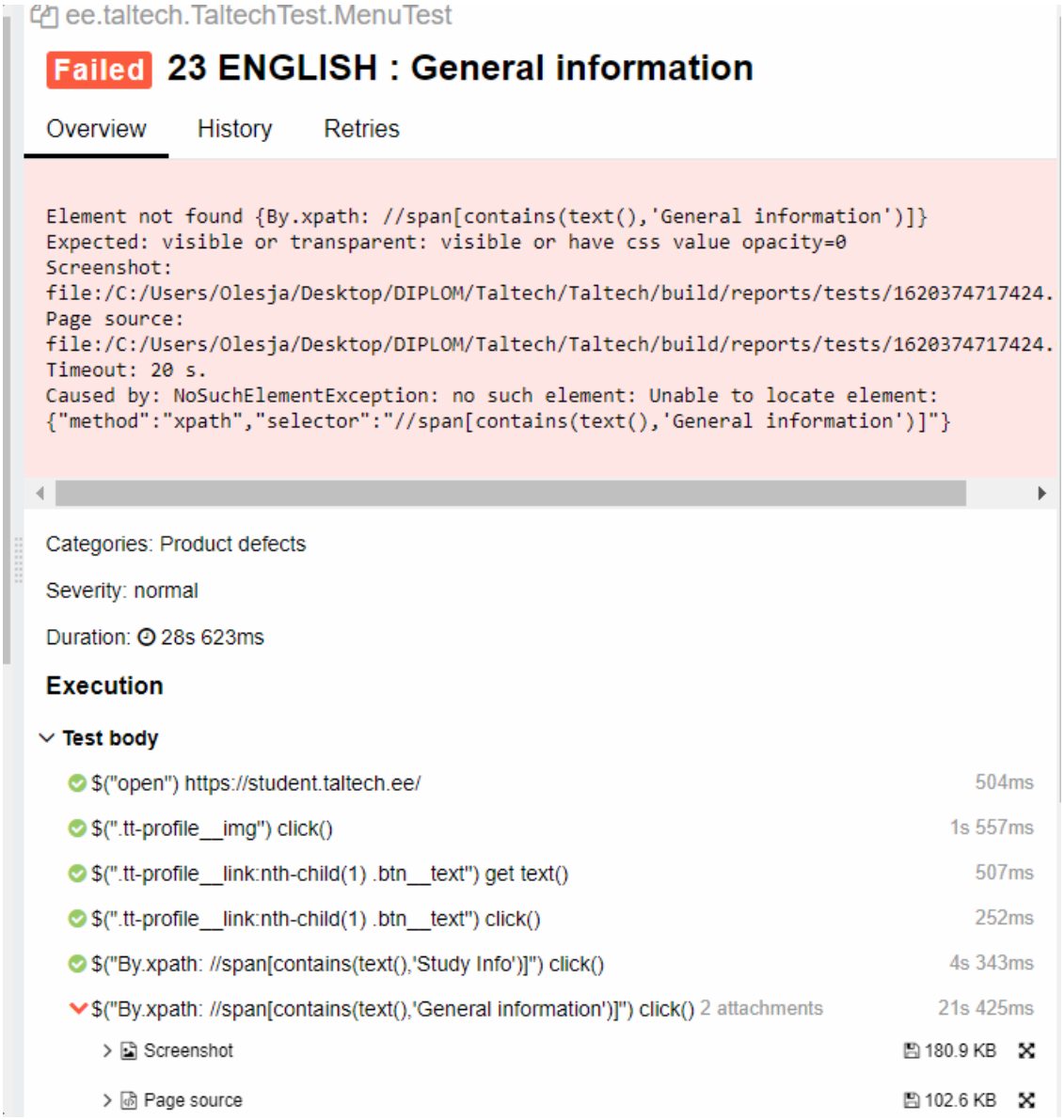


Joonis 25. Näide: Allure Testide üldaruanne.

Vahekaardil Suites saab vaadata eraldi konkreetse testi jaoks üksikasjalikku teavet:



Joonis 26. Näide: Allure Test Suite üldine vaade.



Joonis 27. Näide: Kukkunud testi nr. 23 detailne informatsioon. Allure.

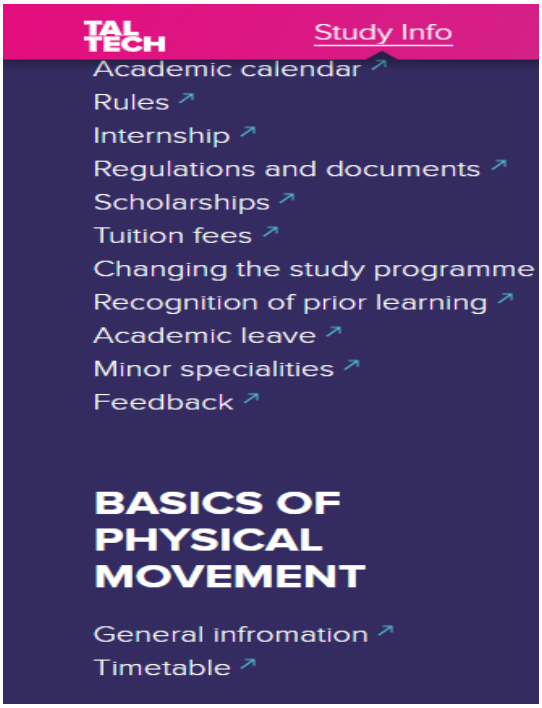
Antud test kirjeldab kõiki samme ja sellest tulenevat viga. Saadud vea asemel luuakse automaatne ekraanipilt ja lehe koopia

Nagu aruandest (Joonis. Allure Testide üldine aruanne) nähtub, et 33 testi 36st lõppesid edukalt, tulemus vastas ootustele. Allpool autor üksikasjalikumalt kaalub ja kirjeldab ebaõnnestunud teste.

3.5 Ebaõnnestunud testide ülevaade

3.5.1 Test 23. Õigekirjavead.

Eeldatakse, et vaheleht „General Information“ on avatud ingliskeelses peamenüüs „Study info“. Kuid test ebaõnnestub veebilehe õigekirjavea tõttu, oodatud „General Information“ asemel öeldakse „General Infromation“, tähtede järjekord pööratakse ümber ja vastavalt sellele tehakse test veaga.

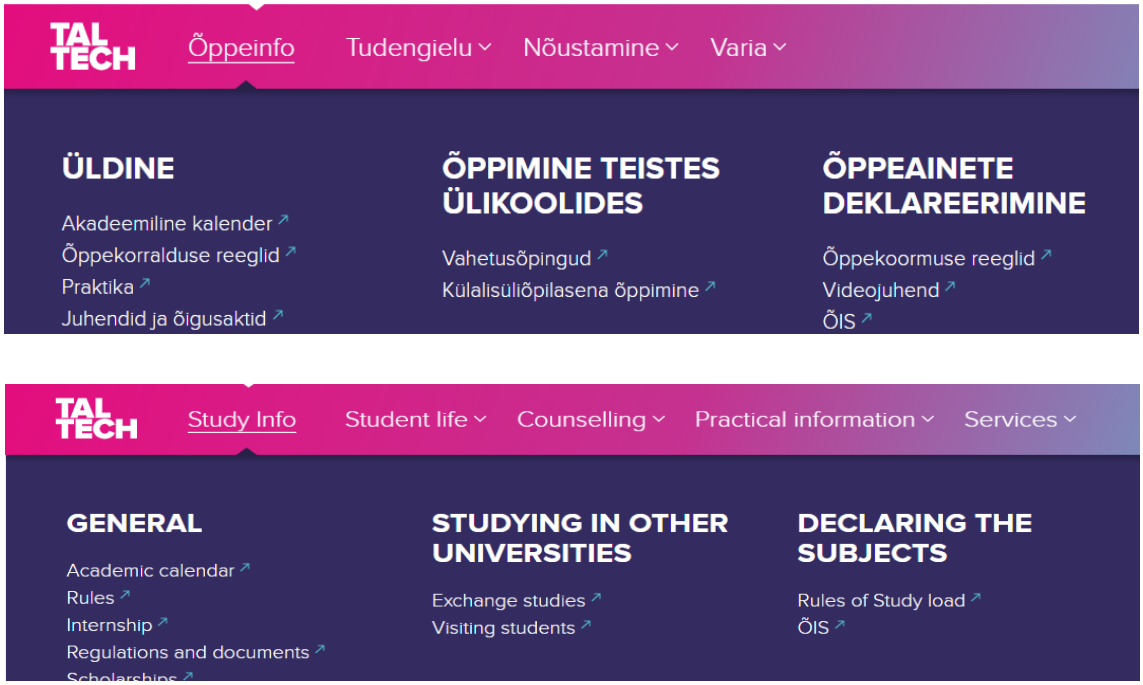


Joonis 28. Näide: Läbikukkunud test n.23. Allure aruanne.

.

3.5.2 Test 33. Eestikeelse ja inglisekeelse versiooni mittevastavus.

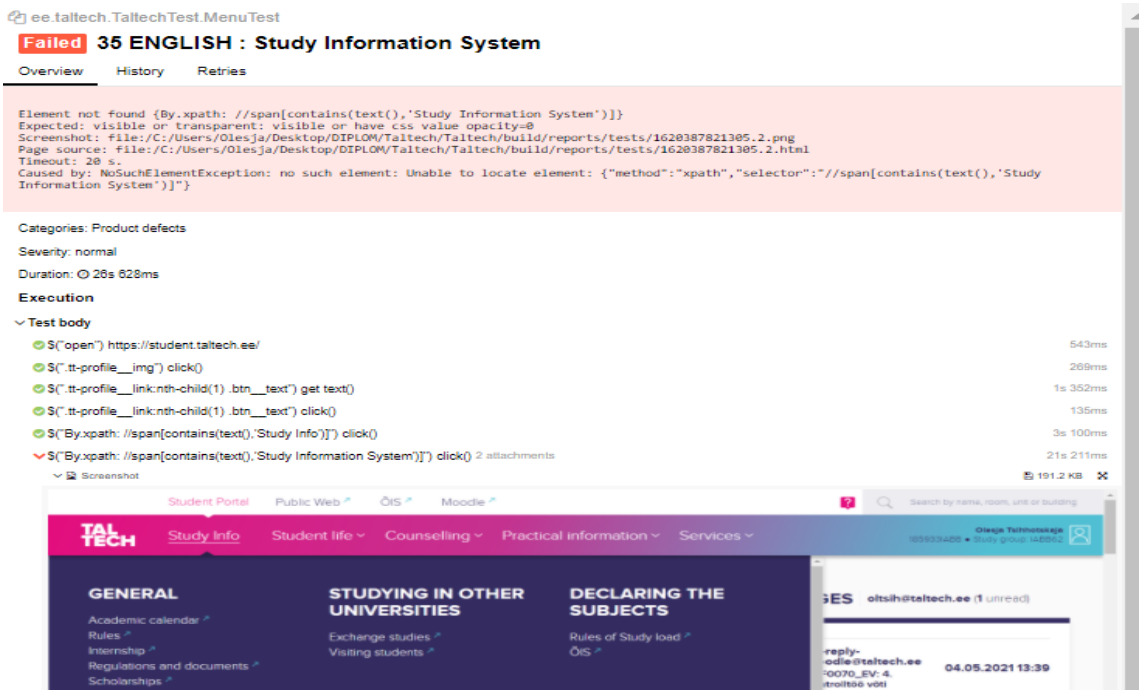
Eestikeelses versioonis on menüül kolm vahelehte, ingliskeelses versioonis on neid ainult kaks, seega vaheleht “Videojuhend” puudub. Kuna autor eeldab, et mõlema keele versioonid peaksid olema identsed, võib selle põhjuseks olla pigem veebilehe viga, kui ilmne viga.



Joonis 29. Näide: Keele versiooni mittevastavus. https://student.taltech.ee//.

3.5.3 Test 35. Eesti tähemärkide kasutamine veebilehe ingliskeelses versioonis.

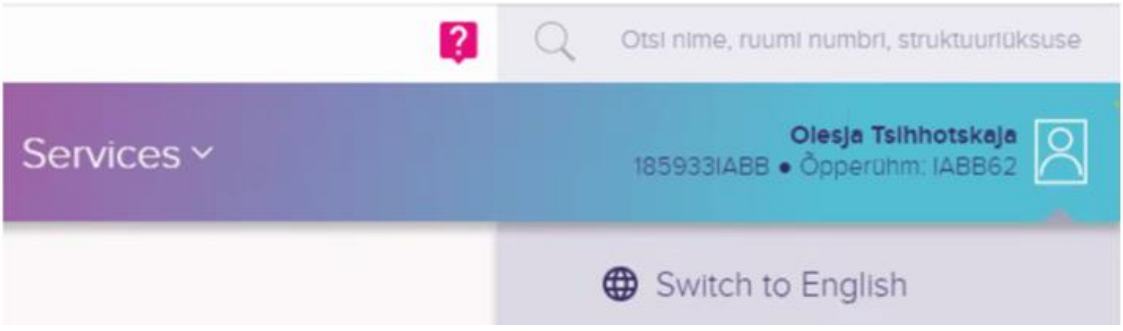
Autor usub, et ingliskeelses versioonis oli õigem kasutada lühendit ÕIS (õppeinfosüsteem) asemel midagi sellist nagu SIS või Study Information System. Selle nüansi võib omistada ka väiksematele vigadele, mitte suurtele vigadele. Kui vahelehe nimi muudetakse ÕIS-ist õppetöö infosüsteemiks, siis järgmiste testide jooksul just see test on töötest.



Joonis 30. Näide: Läbikukkunud test n.35. Allure aruanne.

3.6 Täiendav testimine. Erinevatelt seanssidelt andmete salvestamisel tekkiv viga.

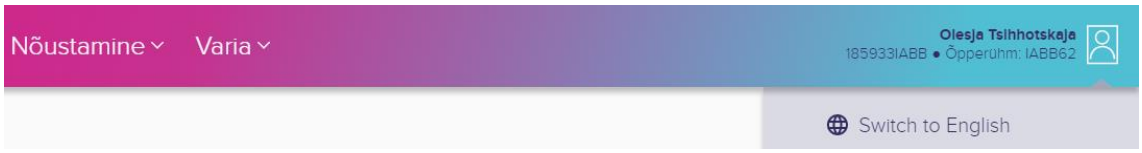
Autor esitab järgmist eeldust: kui minna veebilehele ühes brauseris ja sisse lülitada ingliskeelset versiooni ning teises brauseris minna samale veebilehele ja sisse lülitada eesti keelt, siis kui naasta veebilehele esimesena brauseris võib keele vahetamisel ilmneda viga. Ingliskeelne versioon näitab üleminekut inglise keelele ja eestikeelne versioon näitab üleminekut eesti keelele:



Joonis 31. Näide: Viga, mis ilmub kahe brauseri korraga kasutamisel.

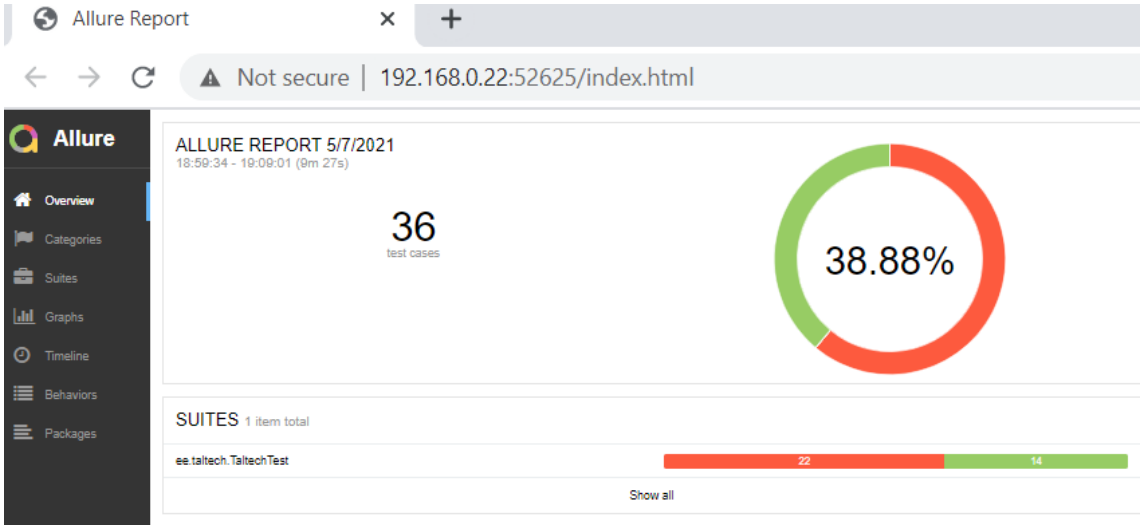
Kuid see peaks olema nii, et ingliskeelne versioon peaks näitama üleminekut eesti keelele ja vastavalt vastupidi eestikeelsele versioonile:

Joonis 32. Näide: Korrektne vaade - lülitamine eestikeelseks. https://student.taltech.ee/

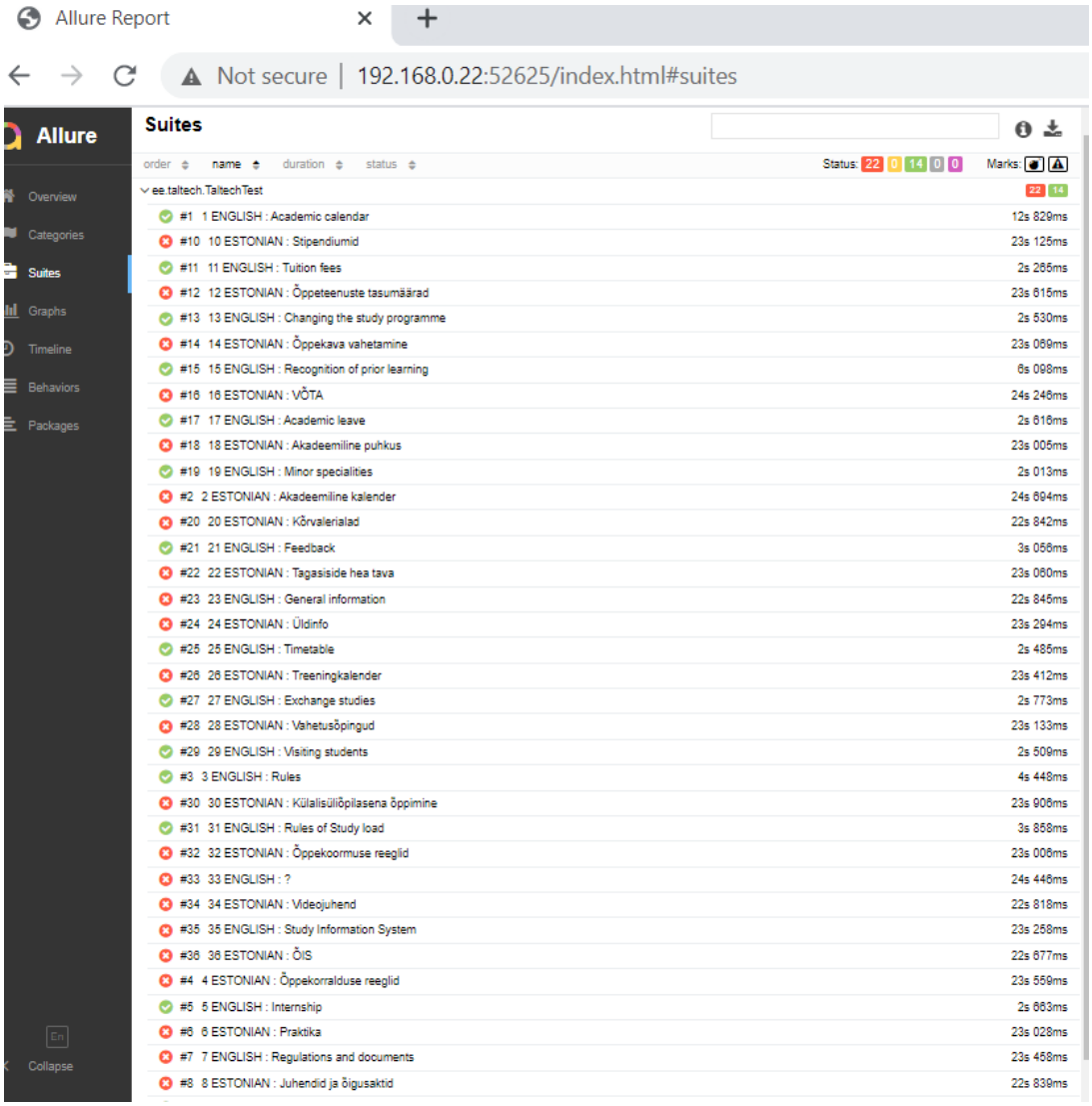


Joonis 33. Näide: Korrektne vaade, lülitamine inglisekeelseks https://student.taltech.ee/

Antud tõrget põhjustas kasutaja tegevus, kes üritas ühe korraga sisse logida kahte erinevasse seanssi. Eeldatavasti ilmneb serveris viga, kui proovida salvestada seansside andmeid erinevates keeltes. Sel hetkel olid käivitatud automatiseeritud testid, millest 22 katset kukkusid läbi. Need olid kõik veebilehe eestikeelse versiooni testid. Nagu on aruandest näha, ei läbinud kõik veebilehe eestikeelse versiooni teste, kuna viga tõi kaasa asjaolu, et keel ei vahetunud.



Joonis 34. Näide: Kahe browseri kasutamisel kukkunud testide koonvaade.



Joonis 35. Näide: Kahe brauseri kasutamisel kukkunud testide koonvaade.

.

Kahjuks on seda eeldust veebilehe serveri toimimise suhtes võimatu kontrollida, kuna veebilehe arendaja oli alltöövõtja ettevõte ja praegu pole temaga võimalik ühendust saada ning antud teooriat kinnitada või ümber lükata, kuid kasutaja ise saab selle probleemi lahendada, kui logib praegusest seansist välja ja siseneb uuesti, siis veebilehe andmed normaliseeruvad.

3.7 Testide ülevaate kokkuvõtte.

Üldiselt võib tõdeda, et Tudengiportaali infosüsteem (mida testib peamenüü vaadet) töötab hästi, 33 testist 36 olid edukalt läbitud. Testimise käigus oli leitud kolm ülalnimetatud puudust (keeleversioonide vasturääkivus, õigekirjaviga ja eestikeelse lühendi kasutamine ingliskeelses versioonis), mis ei ole kriitilised vead.

Selenide'i ja Selenium'i automatiseeritud testid näitasid samu tulemusi. Selenium nõuab natuke rohkem kodeerimist. Ühe parameetrilise testi raames pole need erinevused nii olulised.

Kasutajaliidese automaatse testimisega projekti, mis selgus bakalaureusetöö kirjutamise käigus, saab kasutada veebilehe töökorras töötamise ja arendamise reaalsetes tingimustes.

Autor usub, et pärast leitud vigade parandamist saavad tudengid (ehk veebilehe peamised kasutajad) saada ajakohast ja usaldusväärset teavet, kuna veebilehega töötamisel ei esine segadust.

Kokkuvõte

Käesoleva bakalaureusetöö põhieesmärk oli testide kirjutamise automatiseerimise tööriista võrdlemine ja valimine uue TTU veebilehe funktsionaalsuse automatiseerimiseks veebilehe vigade uurimiseks. Autor läbis praktika ettevõttes HireRightis testija rollis ja praktika käigus oli antud ülesanne luua ettevõtte veebilehele jaoks automatiseeritud testid. Kahjuks ei olnud autoril võimalik töö raames kasutada tööandja digitaalset toodet selle tõttu, et antud ettevõte töötab klientide isikuandmetega. Seetõttu funktsionaalsuse automatiseerimiseks oli kasutatud Taltechi veebilehte, kuna mõlemal veebirakendusel on sarnane struktuur ja funktsionaalsus.

Antud töö esimene osa on pühendatud praegu kasutatavate testimisviiside ja -meetodite teooria ülevaatele ning nende meetodite analüüsile, võttes arvesse nende plusse ja miinuseid. Analüüsiti erinevaid automatiseeritud testimisvahendeid, arvestati nende eeliste ja puudustega ning valiti välja kaks kõige sobivamat tööriista kasutajaliidese testide, mida õnnestus edukalt lõpule viia.

Automatiseerimistööriista valimise peamisteks nõueteks olid: hind, õppimise ja kasutamise lihtsus, tõhusus ja usaldusväärsus vigade leidmisel.

Töö kolmanda osa teoreetilise teabe ja järelduste põhjal kirjutati Tudengiportaali peamenüü jaoks automaatsed testid, kasutades kahte automatiseerimisvahendit - Selenide ja Selenium, ning oli tehtud nende võrdlus, võttes arvesse toodud nõudeid ja kasutusmugavust.

Testid kontrollivad kasutajaliidese elementide vastavust, kontrollivad kasutajaliideses järgmiseid vigu: veebilehe elementides puudub või on vale teave, veebilehe osade ligipääsmatus, veebilehe struktuuri erinevused erinevates keeleversioonides.

Kasutajaliidese kirjalik automatiseerimine ei tuvastanud tõsiseid vigu, olid leitud vaid üksikud väikesed vead ja kokkuvõtteks saab autor väita , et uus Tudengiportaali veebirakendus töötab hästi.

Autori sõnul praktiline võrdlus kinnitas, et Selenide sobib rohkem automatiseerimise kirjutamise jaoks, kuna see tööriist on koodi kirjutamise seisukohalt väga lakooniline, aitab kaasa kõige tõhusamate automaatsete testide loomisele, võimaldab kiiresti ja mugavalt elemente leida veebilehe elementide kättesaadavuse ja toimivuse kontrollimiseks.

Eeltoodust järeldub, et püstitatud eesmärgid on saavutatud ja bakalaureusetööga on saadud tulemused, mis vastavad ootustele lõputöö tulemusele.

Kasutatud kirjandus

[1] Зачем нужно тестирование сайта, [WWW], https://www.dalee.ru/blog/test_site_2.html. Kasutatud 07.03.2021

[2] QA and UX, [WWW], https://www.nngroup.com/articles/quality-assurance-ux/. Kasutatud 15.03.2021

[3] M. Cohn, „Succeeding With Agile. Software Development Using Scrum“, [WWW], http://1.droppdf.com/files/AFvTS/succeeding-with-agile-mike-cohn.pdf. Kasutatud 07.03.2021

[4] Прирамида тестов на практике, [WWW], https://habr.com/ru/post/358950/. Kasutatud 08.03.2021

[5] The Test Automation Pyramid, [WWW], https://www.ontestautomation.com/the-test-automation-pyramid/. Kasutatud 07.03.2021

[6] Прирамида тестов к колесу автоматизации, [WWW], https://www.software-testing.ru/library/testing/testing-for-beginners/3379-rethinking-pyramid-automation-test-wheel. Kasutatud 08.03.2021

[7] Автоматизация и пирамида тестов, [WWW], https://lebedev.of.by/piramida-tests/. Kasutatud 07.03.2021

[8] UI/UX, [WWW], https://on.net.ua/index.php/2019-08-25-12-16-48/ui-ux/2-others. Kasitatud 15.03.2021

[9] UI/UX, [WWW], https://lpgenerator.ru/blog/2016/07/09/vse-chto-vam-nuzhno-znat-o-user-experience/. Kasutatud 15.03.2021

[10] Как UI влияет на успешность бизнеса, [www], https://semantica.in/blog/chto-takoe-polzovatelskij-interfejs-i-kak-on-vliyaet-na-uspeshnost-biznesa-v-czelom.html. Kasutatud 15.03.2021

[11] Тестирование UI (пользовательского интерфейса), [WWW], https://woxapp.com/ru/our-blog/testing-the-ui-user-interface// Kasutatud 15.03.2021

[12] Тестирование пользовательского интерфейса, [WWW] , https://xbsoftware.ru/testirovanie-po/polnij-tsykl/testirovanie-ui/. Kasutatud 15.03.2021

[13] Методы оценки качественного пользовательского интерфейса, [WWW], http://it-claim.ru/Library/Books/ITS/wwwbook/ist6/ponomarev2/ponomarev2.htm. Kasutatud 15.03.2021

[14] Автоматизированное VS ручнное тестирование, плюсы и минусы подходов, [WWW]. https://qa-academy.by/qaacademy/news/ruchnoe-i-avtomatizirovannoe-testirovanie-plyusy-i-minusy-podxodov/. Kasutatud 17.04.2021

[15] Automated vs Manual Testing Comparison [WWW]. https://www.guru99.com/difference-automated-vs-manual-testing.html. Kasutatud 17.04.2021

[16] Автоматизированное VS ручнное тестирование. [WWW]. https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/ruchnoe-testirovanie/avtomatizirovannyi-vs-ruchnoe-testirovanie. Kasutatud 17.04.2021

[17] Top 10 automation testing tools, [WWW], https://www.netsolutions.com/insights/top-10-automation-testing-tools/. Kasutatud19.04.2021

[18] SoapUi. The Industries's #1 API Testing Tool. [WWW] https://www.soapui.org., Kasutatud 20.04.2021

[19] Selenide. Quick Start. [WWW], https://www.selenium.dev. Kasutatud 20.04.2021

[20] Cucumber Framework, [WWW], https://cucumber.io/. Kasutatud 20.04.2021

[21] Codeborne. Selenide and Selenium comparison, [WWW], https://selenide.org/documentation/selenide-vs-selenium.html. Kasutatud 22.04.2021

[22] Selenide and Selenium Comparison. [WWW] https://sergeypirogov.gitbooks.io/pro-selenide/content/selenide-vs-selenium.html. Kasutatud 22.04.2021

[23] Selenide. Create a Custom WebDriver, [WWW],https://mbbaig.blog/selenide-webdriverfactory/. Kasutatud 01.05.2021

[24] Junit 5 with Allure Report Example, [WWW], https://javabydeveloper.com/junit-5-with-allure-reports-example/. Kasutatud 01.05.2021

[25] Boilerplate code, [WWW], https://en.wikipedia.org/wiki/Boilerplate_code. Kasutatud 13.05.2021

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

Mina, Olesja Tsihhotskaja

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

18.05.2021

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.

Lisa 2 – Github link

https://github.com/Olesjat/TaltechTests - Tudendiportaali automaat testide projekt.