Tallinna Ülikool

Digitehnoloogiate instituut

Informaatika õppekava

VEEBISAIDI KASUTAJALIIDESE TESTIMISE RAKENDUSE ARENDAMINE

Bakalaureusetöö

Autor: Kardo Jõeleht

Juhendaja: Romil Rõbtšenkov

Autor: ....., ..., 2017

Juhendaja: ....., ..., 2017

Instituudi direktor: ....., ..., 2017

Tallinn 2017

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.....	.....
(kuupäev)	(autor)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina Kardo Jõeleht (sünnikuupäev: 06.10.1987)

1. Annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Veebisaidi kasutajaliidese testimise rakenduse arendamine”, mille juhendaja on Romil Rõbtšenkov, säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.

2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas, _____

(digitaalne) allkiri ja kuupäev

Sisukord

Sissejuhatus

McConnell (2004) on välja toonud, et testimine on kõige populaarsem tarkvara kvaliteedi tõstmise viis – erinevate testimistegevuste tulemuslikkust näitavad mitmed uurimused ja kogemus. Selleks, et arendaja saaks kindel olla, et tema loodud veebilehe kasutajaliides töötab nii nagu ta seda on ette näinud, on vaja liidese vahendusel erinevad kasutajalood läbi proovida. Iga kord pärast uue funktsionaalsuse lisamist oleks hea põhilisi asju, kui mitte kõike, uuesti testida, et olla kindel, et kogu süsteem endiselt töötab. Korduvalt samade tegevuste manuaalselt läbi tegemine muutub aga pikapeale tüütuks.

Käesolev bakalaureusetöö pakub ühe võimaliku lahenduse sellise testimise automatiseerimiseks. Eesmärgiks on luua rakendus, mis on võimeline avama veebilehe, salvestama kasutaja poolt tehtud tegevuste jada ja selle taasesitama, tehes navigatsioonid, tekstide sisestused ja kontrollid täpselt selliselt, nagu kasutaja need ette on näinud. Kui mingil põhjusel, näiteks kasutajaliidese elemendi puudumisel, ei õnnestu testi edukalt taasesitada, annab rakendus sellest teada. Loodava rakenduse sihtgrupina näeb autor eelkõige manuaaltestijaid, kuid ka kõiki teisi huvilisi, kes veebirakenduste kasutajaliidestega kokku puutuvad.

Töös ei keskenduta liialt koodidetailidele, vaid pigem antakse arendusprotsessist, ideedest ja valminud rakenduse osadest üldisem ülevaade. See tähendab, et kirjeldatakse küll olulisemad tarkvara loomisel tehtud otsused, avastused ja loogika, kuid ei kirjeldata koodi kui õpetust arendajale samasuguse lahenduse loomiseks. Eeldatakse, et lugeja on levinumate IT-alaste mõistetega tuttav. Samas aga saab rakenduse lähtekood olema vabalt saadaval avalikus repositooriumis, kus huvilistel on võimalus iga koodidetailiga tutvuda. Lisaks jätab autor soovijatele võimaluse töö osana loodud tarkvara parandada ja oma vajaduste järgi vabalt täiendada.

Töö jaguneb kuueks peatükiks. Esimeses peatükis kirjeldab autor lähemalt, millist rakendust hakatakse looma ja millised funktsionaalsused käesoleva bakalaureusetöö raames valmivad. Teiseks paneb autor paika kasutatavad tehnoloogiad ja annab neist ülevaate, et töö oleks huvilistele hästi mõistetav. Tarkvaraarendaja võib sellest töö osast saada ka inspiratsiooni kasutatud tehnoloogiate abil mõne oma rakenduse loomiseks või otsustamiseks, kas ja millises olukorras on mõistlik autori valitud vahendeid rakendada.

Järgmise sammuna selgitatakse välja, millised on need kasutajaliidese elemendid, missugustele on mõttekas keskenduda ja milliseid kasutades saab olulise osa kasutajalugudest testitud. Töö maht ei võimalda siin kõiki elemente käsitleda.

Edasi kirjeldab autor rakenduse loomist ja sellega seotud otsuseid. Kirjeldatakse rakenduse arhitektuuri, failistruktuuri, kasutatavaid võtteid. Tutvustatakse millised on plaanid, kuidas need muutuvad ja mis jääb lõpptulemusse. Kui rakendus on juba kasutuskõlblik, toimub erinevate testilugude katsetamine.

Viimasena antakse ülevaade avatud lähtekoodiga tarkvara avaldamise võimalustest. Kirjeldatakse, mis on niisugust teed minemise head ja halvad küljed ning kuidas avatud litsentsi kasutamine võib tarkvara edule või ebaedule kaasa aidata.

1 Rakenduse kirjeldus, konkurents ja sihtgrupp

Arendatava rakenduse eesmärgiks on veebilehe kasutajaliidese kaudu tegevuste jadade (stsenaariumite) salvestamine ja nende taasesitamine, kontrollides igal sammul, kas tegevuse esitamine õnnestus korrektselt. Salvestuse osaks on navigatsioonid, sisestused ja kontrollid. Kasutaja saab ükshaaval omas tempos interaktsioonid läbi teha ja jada testina salvestada. Salvestatud stsenaariumit on võimalik uuesti rakendusse laadida ja taasesitada. Pärast salvestuse taasesitamist teatatakse kasutajale, kas ja millises osas stsenaarium edukalt läbiti. Eesmärgiks on olla kindel, et veebilehel on vajalikud elemendid olemas ja navigatsioon on võimalik täpselt selliselt, nagu see oli tegevuste salvestamise hetkel.

Esimese sammuna on kasutajal pärast rakenduse käivitumist võimalik navigeerida veebilehele või valida mõni eelnevalt salvestatud test. Kui kasutaja on soovitud veebilehele navigeerinud, saab ta alustada teststsenaariumi salvestamist. Kui ta aga ei taha alustada uue stsenaariumi salvestamist vaid valib juba salvestatud testi, on võimalik seda esitada.

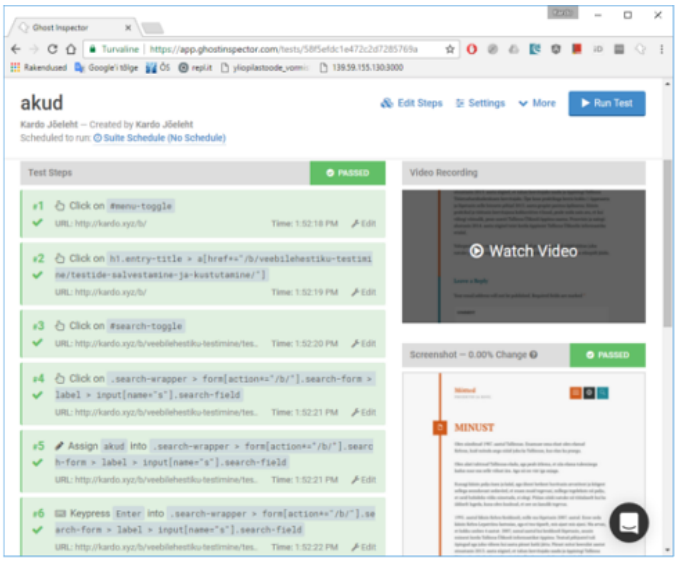
Olles uue stsenaariumi salvestamist alustanud, salvestab rakendus kasutaja interaktsioonid nende toimumise järjekorras. Salvestuse käigus saab igal hetkel lisada elemendi kontrolli. Pärast soovitud elemendi valimist saab märkida, kas kontrollitakse tema olemasolu, tema sisuteksti või ükskõik millist tema atribuuti. Taasesituse ajal kontrollitakse, kas elemendil olid kõik nõutud omadused olemas või kas element ise oli lehel olemas. Salvestuse lõpetamisel küsitakse kasutajalt testi nime. Kui nimi on sisestatud, saab stsenaariumi faili salvestada. Salvestatud faili on võimalik ka teistele jagada.

Kuna kasutajaliidese osas on tegu üsna lihtsa rakendusega, ei pidanud autor vajalikuks kasutajaliidest eraldi prototüüpida ega jooniseid teha. Ekraanitõmmised valminud rakendusest tuuakse välja edaspidi.

Põhiliseks konkurendiks autori loodavale rakendusele on Ghost Inspector[1] (vt Illustratsioon 1), mis pakub pea sarnast funktsionaalsust nagu käesolevas töös loodav rakendus. Ghost Inspectori miinuseks on tasuta plaani väikesed mahud: muuhulgas

1 https://ghostinspector.com/

ainult 100 testi käivitamist ühes kuus (Ghost Inspector, 2017). Kui aga on vajadus Ghost Inspectorit suuremas mahus kasutada, peab igakuiselt juba märgatava rahasumma maksma ja isegi tasulistel plaanidel on testide käivitamise kordade piirangud, kuna lahendus töötab Ghost Inspectori serverites. See, et teenus tootja servereis töötab, on autori arvates miinuseks, kuna proovimise hetkel käivitati test näiteks USAs asuvas serveris. Testi läbimine võttis isegi ilma ühegi sammuta testi puhul märgatava aja. Ghost Inspectori plussideks on suuremad võimalused, viimistletum teenus ja detailsem tagasiside erinevates vormides (näiteks ekraanitõmmised ja testi video). Serveris töötava teenuse puhul on ekraanitõmmised ja video vajalikud tulemuse paremaks tajumiseks, aga nende genereerimine võtab aega. Kohaliku rakenduse puhul saab soovi korral tõmmiseid ise teha või ise videosalvestise luua.



Illustratsioon 1. Ghost Inspectori testi kokkuvõte

Kasutajaskonnana näeb autor peamiselt manuaaltestijaid – neid inimesi, kelle tööks on veebirakenduse testimine läbi kasutajalugude käsitsi katsetamise. Rakendus aitaks neil testide korduvat läbimist kiirendada ja kergema vaevaga proovida sisendi varieerumisega kaasnevaid erinevaid vastuseid. Lisaks võiks rakendus huvi pakkuda kasutajaliidese arendajatele. Kui kujundus on paika pandud ja ka rakenduse serveri pool on teatud maani ära realiseeritud, saaks edaspidise arenduse käigus korduvalt teste uuesti käivitades olla kindel, et kasutajaliidesega tihedamalt seotud asjad töötavad. Rakenduse eesmärgiks ei ole asendada arendaja poolt programmeerimise käigus kirjutatavaid automaatteste, vaid pakkuda võimalus lisaks nendele ka kasutajaliidesega tihedamalt seotud osade lihtsamaks testimiseks.

2 Kasutatavad tehnoloogiad

On väga mugav kui rakendust saab kasutada erinevatel operatsioonisüsteemidel ilma selle koodi spetsiaalselt iga ühe jaoks muutmata või täiesti uuesti kirjutamata. Autor tahab oma töös kindlasti kasutada seda võimaldavat lahendust. Järgnevalt on toodud lühikirjeldused valitud tehnoloogiatest ja vahenditest.

2.1 JavaScript, HTML, CSS

Üheks võimaluseks, kuidas oma rakendust lihtsa vaevaga erinevate operatsioonisüsteemide jaoks luua, on viimastel aastatel populaarseks saanud JavaScripti, HTMLi ja CSSi abil hübriidrakenduste loomine. Personaalarvutitele saab seda teha näiteks Electroni[2] abil, nutitelefonidele kasutades Cordovat[3]. Mitmed laialt tuntud ettevõtted nagu Google ja Facebook on kasutanud Electroni ja veebitehnoloogiaid oma rakenduste arendamisel (Finley, 2016). Idee seisneb selles, et rakendus luuakse mainitud tehnoloogiaid kasutades ja pakendatakse operatsioonisüsteemi spetsiifilisse konteinerisse (nt Windowsi jaoks luuakse `.exe` fail). Konteineris on veebibrauser ja selle sees käivitatakse veebilehestik. Rakenduse kasutajaliides on moodustatud HTMLi ja CSSi abil, arvutused teeb ära lehtede küljes olev JavaScript. Kirjeldatut kasutades luuakse mulje, nagu oleks tegu „päris” töölauarakendusega, kuid tegelikult saab rakendus vabalt hoopis ka veebiserveris töötada. Seda küll mõningate eranditega, sest mainitud hübriidrakenduste platvormid pakuvad võimalusi ka operatsioonisüsteemi käskude käivitamiseks: näiteks ligipääs failisüsteemile.

2.2 Node.js

Node.js on JavaScripti käituskeskkond, mis võimaldab kirjutada veebiserveri koodina JavaScripti (Node.js Foundation, 2017). Lisaks on temaga kaasas paketihaldur (laienduste haldur) Node Package Manager (`npm`), mille abil saab lisamoodulite näol oma süsteemi lihtsasti uut funktsionaalsust hankida. Seda nii serveris kui personaalarvutis. Üheks `npm`’i mooduliks on ka Electron. Uue Electroni projekti

2 https://electron.atom.io/

3 https://cordova.apache.org/

tegemiseks ei ole vaja teha muud, kui installeerida oma arvutisse Node.js ja hankida `npm`'i abil Electron. Täpsemad juhendid, kuidas projekti luua ja Electroni kasutada, on leitavad Electroni kodulehelt https://electron.atom.io.

2.3 Electron

Autori jaoks on Electron just õige valik seetõttu, et arendatava rakenduse sisenditeks on veebilehed ja Electroni põhifunktsionaalsus on selleks hästi sobiv – laadida veebileht konteinerrakendusse ja sellega seal suhelda. HTML dokument võib olla pärit kas veebiaadressilt või kohalikult kettalt. Järgnevalt annab autor natuke tehnilisema kirjelduse Electroni toimimisest.

Electroniga loodud rakenduse tehnilise poole võib jagada kaheks. Üheks osaks on *main*-protsess (`main.js`), mis tegeleb akende (`BrowserWindow`) instantside loomisega ja haldamisega. Tehes seda sarnaselt veebirakenduse serveri poolele või tagaosale (ingl *back-end*). Uue akna klassi nimi on `BrowserWindow` seetõttu, et ta sisaldab tüüpiliselt HTML dokumenti, aga sisuliselt on siin tegu töölauarakenduse aknaga. Teiseks põhiliseks komponendiks on *renderer*-protsess, mis käivitatakse akna loomisel ja igal avatud aknal on oma *renderer. Renderer* on aknaga seotud protsess, mis haldab ühe akna sees toimuvat ja teeb koodis kättesaadavaks selles oleva veebilehe (Electron, kuupäev puudub). Samuti võimaldab *renderer* suhtlust rakenduse tagaosaga (*main*).

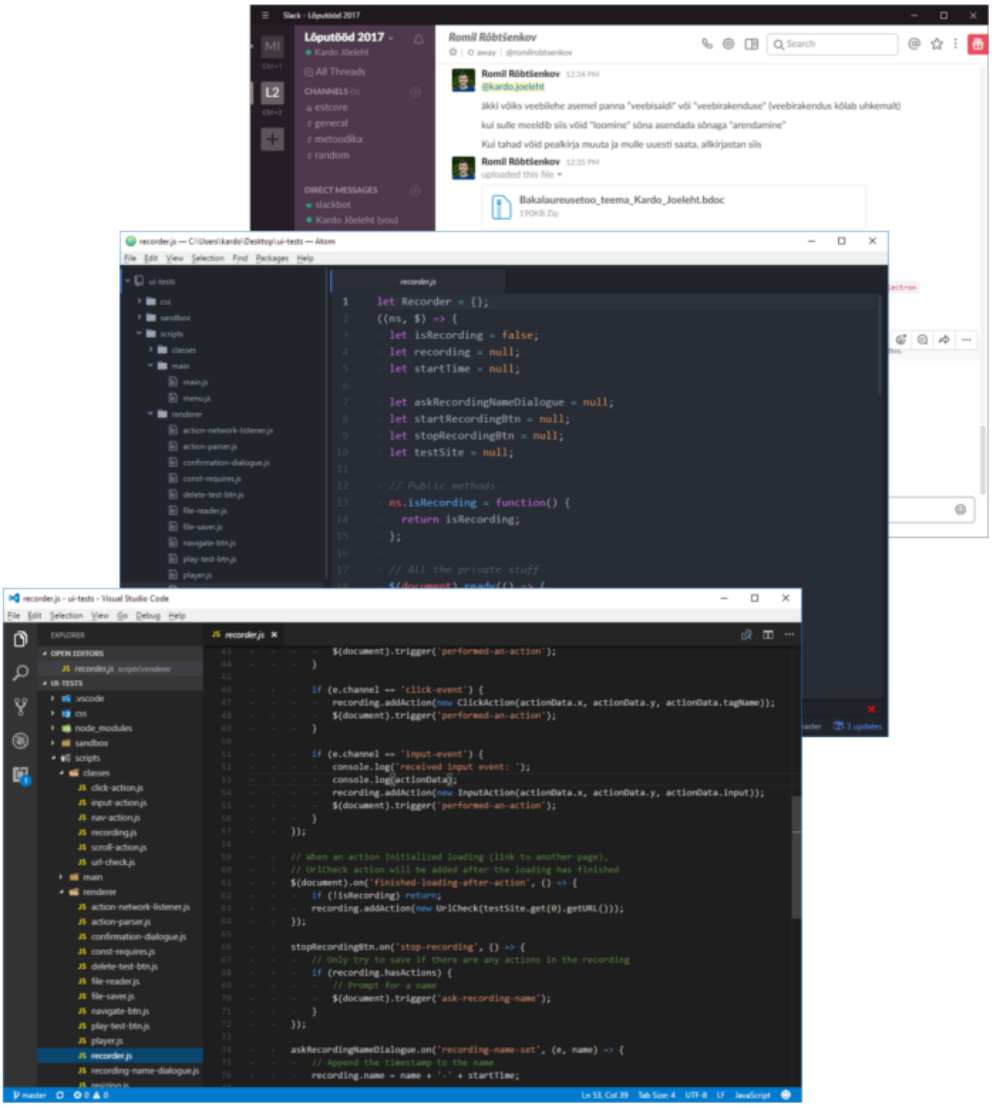
Kui tegu on lihtsama rakendusega, kus on ainult üks `BrowserWindow` instants, toimub kogu rakenduse töö selliselt, et selle ühe akna sisse laetakse veebileht, mille küljes on JavaScripti koodifailid, mis käivitavad lehel erinevaid operatsioone või laevad aknasse kogunisti terve uue lehe.

Rakenduse protsessid on võimelised Node.js laienduste abil käivitama ka operatsioonisüsteemi põhiseid käske, mis muidu veebirakendustele kättesaamatud on. Näiteks on võimalik failisüsteemiga suhtlemine, mis teeb autori loodavas rakenduses võimalikuks testide kohalikesse failidesse salvestamise ja nende lugemise.

Rakenduse tagaosa ehk *main*-protsess ja eesosa ehk *renderer*-protsessid saavad omavahel asünkroonsete ja vajadusel ka sünkroonsete (see on mittesoovitatav kasutajaliidese töö häirimise tõttu) sõnumite kaudu suhelda. Kogu programmikood kirjutatakse kasutades JavaScripti. Kasutajaliides moodustub lehel kasutatavast

HTMList ja CSSist. Ja näiteks Internet Exploreri erisustega ei ole vaja arvestada, kuna tegevus toimub Chrome'i veebibrauseris. Iga töötav Electroni rakendus käivitab sisuliselt uue Chrome'i, mida peetakse miinuseks, sest Chrome kasutab palju süsteemi ressursse (Vilches, 2016).

Mõned populaarsed Electroni abil tehtud rakendused on Slack[4], Atom[5] ja Visual Studio Code[6] (vt Illustratsioon 2). Visual Studio Code'i kasutab autor ka käesolevas töös loodava rakenduse arendamisel.



Illustratsioon 2. Ekraanitõmmised - Slack, Atom, Visual Studio Code

4 https://slack.com/

5 https://atom.io/

6 https://code.visualstudio.com/

3 Toetatavate kasutajaliidese elementide valik

Üks olulisemaid käesoleva tööga seotud teemasid on see, missuguste kasutajaliidese elementidega seotud testimise funktsionaalsused tuleks kindlasti rakenduse esialgsesse varianti kaasata. Vaja oleks vähemalt kolme kõige enam kasutatavat elementi. Lihtne ei ole leida statistikat, milliseid elemente kasutatakse veebilehel kõige enam.

3.1 Teststsenaariumid

Selleks, et jõuda järeldusele, millistele kasutajaliidese elementidele keskenduda, pakub autor välja mõned lood, mida võib erinevatelt veebilehtedelt leida. Iga loo puhul hindab autor, millist funktsionaalsust on nende testimiseks vaja. Lugude juures on välja toodud tegevused ja iga tegevuse juures sellega seotud elemendid ja funktsionaalsus. Tegevused on kirjeldatud ülevalt alla sooritamise järjekorras.

3.1.1 Sisselogimine

Testi alustamise hetkel ollakse soovitud veebisaidi avalehel. Rakendus ei testi aadressiribale aadressi sisestamist ja sellele navigeerimist, kuna mainitu on veebilehitseja mitte testitava veebilehe funktsionaalsus.

Tabel 1. Sisselogimisega seotud tegevused

Jrk	Tegevus	Elemendid	Funktsionaalsus
1	Sisselogimislehele navigeerimine	Hüperlink, nupp	Vajutuse registreerimine
2	Kasutajakonto väljale vajutamine	Tekstiväli	Vajutuse registreerimine
3	Kasutajakonto sisestamine	Tekstiväli	Märkide sisestamise registreerimine
4	Parooli väljale vajutamine	Tekstiväli	Vajutuse registreerimine
5	Parooli sisestamine	Tekstiväli	Märkide sisestamise registreerimine
6	Sisselogimine	Hüperlink, nupp	Vajutuse registreerimine
7	Navigatsioon	-	URLile navigeerimine, validatsioon

Sisselogimisega seotud tegevused loetleb Tabel 1. Selle väga levinud stsenaariumi jaoks on vaja tekstivälja, hüperlingi ja nupu toetust. Sisselogimislehele navigeerimine on tõenäoliselt lahendatud hüperlingina, kuid võib olla ka nupuna. Järgmiseks peab olema võimalik tekstiväljale vajutada ja sinna andmeid sisestada. Viimaseks peab saama

vajutada sisselogimisnupule, mispeale toimub navigatsioon, mida rakendus valideerima peab.

Viimane punkt toob olulise avastuse. Kuna käesolev stsenaarium lõpeb sisselogimisega (nupu vajutusega, navigatsiooniga), peab rakendus olema loo taasesitusel võimeline kontrollima, kas navigeeriti tagasi samale leheküljele (ebaõnnestunud sisselogimine) või uuele leheküljele (õnnestunud sisselogimine). Kasutaja ei sisesta käsitsi pärast sisselogimist enam ühtegi kontrolli, aga rakendus peab navigatsioonide sihtkohtade kohta ise kontrollid looma, kuna muidu ei saa ta kuidagi tuvastada, kas sisenemine õnnestus või mitte.

3.1.2 Otsingutulemuste kontroll

Tabel 2 toob välja tegevused otsingutulemuste kontrolli sooritamiseks.

Tabel 2. Otsingutulemuste kontrolli tegevused

Jrk	Tegevus	Elemendid	Funktsionaalsus
1	Otsinguväljale vajutamine	Tekstiväli	Vajutuse registreerimine
2	Otsingusõna sisestamine	Tekstiväli	Märkide sisestamise registreerimine
3	Otsingu nupu või lingi vajutamine	Hüperlink, nupp	Vajutuse registreerimine
4	Otsingutulemuste kuvamine	Erinevad elemendid	Elemendile kontrolli lisamine

Otsingu puhul vajutab kasutaja otsinguväljale, sisestab otsingusõna ja vajutab midagi, mis alustab navigatsiooni. Selleks võiks tavaliselt olla kas link või nupp. Avanenud lehel peab olema võimalik klõpsata otsingutulemustes mõnel elemendil ja näiteks selle sisuteksti kontrollida. Järelikult peab rakendus võimaldama ühe kontrollina lisada ükskõik missuguse elemendi sisuteksti kontrolli.

3.1.3 Menüülinkide olemasolu kontroll

Kui menüü luuakse näiteks dünaamiliselt vastavalt kasutajarollile, võiks saada läbi testida, kas kasutaja rollile vastavad menüülingid on igal lehel saadaval. Tegevused stsenaariumi läbimiseks toob välja Tabel 3.

Tabel 3. Menüülinkide olemasolu kontrolli tegevused

Jrk	Tegevus	Elemendid	Funktsionaalsus
1	Kontroll, kas lingid on saadaval	Erinevad elemendid	Mitmele elemendile kontrolli lisamine
2	Navigatsioon järgmisele lingile	Hüperlink, nupp	Vajutuse registreerimine
3	Kontroll, kas lingid on saadaval	Erinevad elemendid	Mitmele elemendile kontrolli lisamine

Seda stsenaariumit võib lõputult jätkata, kuid olulised punktid on lehele navigeerimine kasutades kas nuppu või hüperlinki ja elementidele kontrolli lisamine. Kuna alati ei ole vaja kontrollida elemendi sisu, võiks rakendus võimaldada ka lihtsalt elemendi olemasolu kontrollida.

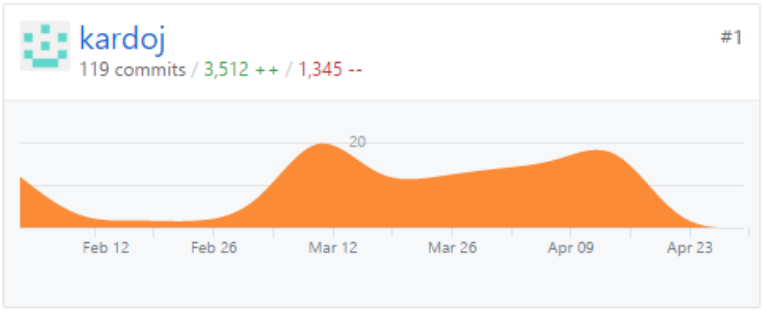
3.2 Järeldused

Valitud stsenaariumite põhjal tegi autor järgnevad järeldused. Katta oleks kindlasti vaja tekstiväljale teksti sisestamine, nupule ja hüperlingile vajutamine ning selle tulemusel aadressile navigeerimine.

Täpsema funktsionaalsuse osas ilmnes teststsenaariumite näidiste loomisel oluline asjaolu, et rakendus peaks iga navigatsiooni juurde automaatselt salvestama ka sihtaadressi kontrolli tegevuse. Seda seetõttu, et kasutaja ei pruugi peale navigatsiooni enam ise ühtegi kontrolli sisestada ja sellisel juhul ei suudaks rakendus ilma sihtaadressi kontrolli lisamata tuvastada, et liikumine õnnestus. Kuid lisaks sellele peab olema kindlasti võimalik iga elemendi puhul kontrollida tema tekstisisu (ja soovitavalt ka atribuute). Samas ei ole iga kord vaja sisu kontrollida, kuna piisab olemasolu kontrollist. Ka selline võimalus võiks olemas olla. Eelnevale lisaks selgus arenduse käigus, et ka Enter klahvi vajutades peaks saama vormi saata, kuna paljudel lehtedel ei olegi otsingu nuppu.

4 Rakenduse arendamine

Käesolevas peatükis on lähemalt kirjeldatud rakenduse arendamise protsess ja kaasatud failid. Väljavõtted koodist on tähistatud `eristiiliga`. Igas alapeatükis on välja toodud esialgsed mõtted ja soovid. Arenduse käigus on neid vastavalt üldpildi selginemisele ja muutumisele täiendatud. Kui lugeja soovib järgnevate alapeatükkide lugemisega paralleelselt kirjeldatud failidega ka lähemalt tutvuda, siis rakenduse lähtekood on saadaval GitHubi[7] koodirepositooriumis aadressil https://github.com/kardoj/ui-tests. Käesolevas töös kirjeldatud funktsionaalsused on arendatud kolme kuu vältel (vt Illustratsioon 3).



Illustratsioon 3. Koodi täienduste graafik

Iga muudatuse juures toob autor välja, mille põhjal ta arvas, et niisugune muudatus oli vajalik ja miks just valitud idee lõplikusse lahendusse jäi. Selline kirjeldamine ja põhjendamine on kasulik nii autorile endale järgmiste projektide arendamisel kui ka avardab teiste tarkvarahuviliste vaadet sellest, kuidas ja miks arenduse käigus asjad muutuvad. See juhtub tarkvara loomise protsessi käigus alati, ükskõik kui palju on eeltööd tehtud. Tarkvara planeerimine on „kaval probleem” (ingl *wicked problem*, autori tõlge). Probleemi mõistmiseks vajalik informatsioon sõltub probleemi lahendaja teadmistest selle lahendamise kohta (Rittel & Webber, 1973) ja nägemus lahendusest saabki selgeks alles (osalise) lahendamise käigus.

4.1 Eelduste selgitamine

Esimeseks ja kõige tähtsamaks eelduseks on võimalus JavaScripti abil veebilehel toimuvaid tegevusi „kuulata”, neid salvestada ja taasesitada ehk emuleerida. Seda, et

7 https://github.com/

JavaScripti abil saab veebilehe elementidele lisada tegevuse kuulari (*event listener*) ja tegevuse toimudes soovitud koodi käivitada, teadis autor juba enne. Kuna aga tegevus on vaja salvestada ja hiljem taasesitada, tõstab see protsessi keerukust.

JavaScriptis on enamike kaasaegsete veebilehitsejate poolt toetatud meetod `document.elementFromPoint()`[8], mis võtab argumentidena vastu x ja y koordinaadid ja tagastab nende põhjal kõige täpsema (lehel kõige pealmise) elemendi. Kui kasutaja vajutab veebisaidi elemendil, saab salvestada selle vajutuse x ja y koordinaadid ning taasesituse ajal eelpool mainitud käsuga leida elemendi ja sellel uuesti vajutust emuleerida.

Arenduse käigus sai selgeks, et lisaks vajutuse koordinaatidele on vaja salvestada ka elemendi märgend (ingl *tag*), et oleks võimalik kontrollida, et `document.elementFromPoint()` leiab taasesitusel sama elemendi, mis testi salvestades. Kui märgend ei kattu, saab testi katkestada veateatega. Kui aga peaks juhtuma, et taasesitus leiab sama elemendi, millele vajutades on lähteaadress teine, kukub test läbi järgmisel sammul kuna lähteaadressi võrdlus ei ole edukas.

4.2 Piirangud

Põhiline piirang peab olema see, et testi esitamisel kasutatakse samasugust akna suurust nagu oli testi salvestamisel. Kuna interaktsioone nagu hiirevajutus salvestatakse ja taasesitatakse koordinaatide põhiselt, ei tundu võimalik, et asi saaks teisiti toimida.

Ühe võimalusena võiks proovida paika panna mingisuguse suhte, mille järgi vastavalt akna suurusele vajadusel koordinaate ümber arvutada, kuid tõenäoliselt see ei toimiks väga hästi arvestades, et veebilehe elemendid ei paigutu akna suuruse muutudes alati sarnaselt. Pealegi ei kitsenda sama aknasuuruse piirang oluliselt testi võimalusi.

Teine piirang, mis on autoril algusest peale mõttes olnud, on testide esitamine reaalajas. Et täpselt selliste vahedega nagu testi tegevused esialgselt on salvestatud, mängitakse need ka uuesti. Probleemiks on selle juures muidugi asjaolu, et kui kasutaja ootab testi salvestades kahe vajutuse vahel 5 minutit, tuleb see 5 minutit iga kord ka taasesitusel oodata ja kui testi salvestamisel kasutatakse suuremat interneti kiirust ja tehakse klikke kiiresti, ei pruugi taasesitades näiteks aeglasema kiiruse juures leht järgmise tegevuse

8 http://caniuse.com/#search=elementFromPoint

tegemise ajaks veel laetud olla.

Võimalik oleks esialgseid vahesid muidugi korrigeerida. Umbes selliselt, et kui salvestust luues oodatakse kahe tegevuse vahel rohkem kui 10 sekundit, piiratakse vahe alati tagasi 10 sekundile. Samamoodi võiks teha ka siis kui kasutaja on salvestamise ajal kahe tegevuse vahel oodanud näiteks ainult 2 sekundit. Aja saaks siis tõsta tagasi 10 sekundi peale, et olla kindlam, et leht enne järgmist tegevust laetud on.

Õnneks aga selgus arenduse käigus, et Electroni `<webview>` märgendil, mille sisse rakendus testsaiti laeb, on olemas tegevuste `did-start-loading` ja `did-finish-load` teavitajad. Iga kord kui testleht hakkab üle võrgu ressursse laadima, eraldab `<webview>` element vastava rakenduseülese teate ja kui laadimine lõpeb, samamoodi. Neid võimalusi oma koodi integreerides õnnestub luua lahendus, kus tegevusi esitatakse täpselt sellises tempos, nagu võrgu kiirus seda taasesituse ajal võimaldab. Lisaks efektiivsusele on sellist esitust palju huvitavam jälgida.

4.3 Projekti failistruktuur

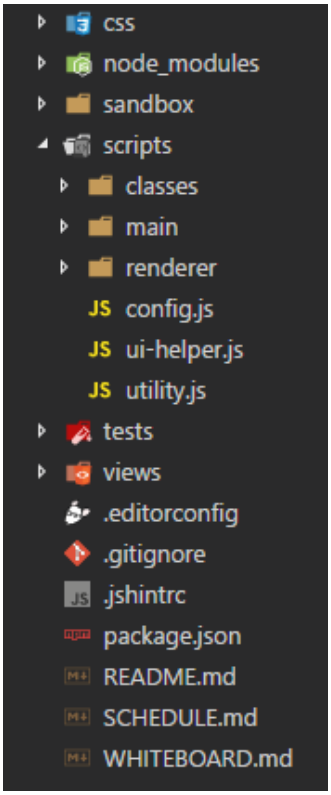
Enne kui algab programmeerimine, oleks hea panna paika mingi kindel projekti failistruktuur, et hiljem oleks lihtne asju üles leida ja iga uue failiga ei peaks liialt kaua mõtlema, kuhu seda paigutada.

Üks projekti struktuur on välja pakutud `electron-vue`[9] poolt. Miskipärast ei õnnestunud autoril seda alguses leida. Hiljem aga uuesti üle vaadates sattus see üsna esimeste tulemuste hulka. Teiseks variandiks on `electron-boilerplate`[10]. Selle abil saab ka mingisuguse aluse, mille pealt oma projekti alustada.

Esimene näidatud variantidest tundub põhjalikum, aga nagu öeldud, leidis autor selle siis kui projekt oli juba tegemises. Ja seal tundub olevat üsna mitmeid kaustu ja faile, mida siinses projektis otseselt vaja ei ole. Ka on need kaks näidet piisavalt erinevad, millest võib järeldada, et standardne projekti struktuur ei ole väga rangelt paika pandud. Oluline on, et failid oleksid programmeerija jaoks loogilistes asukohtades. Autori projekti struktuur on segu koos lisadega pakutud kahest struktuurist ja on näha Illustratsioonil 4.

9 https://simulatedgreg.gitbooks.io/electron-vue/content/en/project_structure.html

10 https://github.com/szwacz/electron-boilerplate



Illustratsioon 4. Projekti failistruktuur

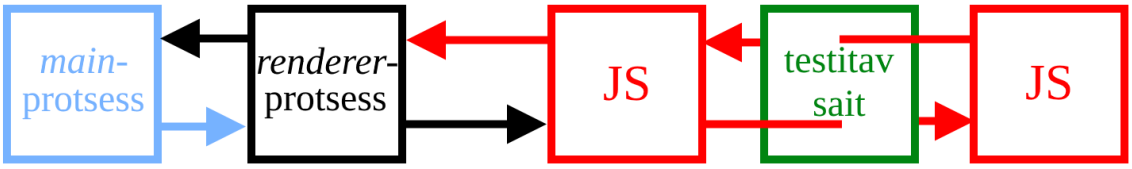
Failide jaotust saab täpsemalt kirjeldada järgmiselt:

- `css` kaustas on stiilifailid.
- `node_modules` kaustas on lisapaketid.
- `sandbox` kaustas on katsetused, mis on projekti käigus erinevate teooriate testimiseks loodud.
- `scripts` on põhiline kaust, kus on koodifailid ja jaguneb omakorda mitmeks:
  - `scripts/classes` kaustas on JavaScript klassid, milledest võib luua programmi töö käigus mitu instantsi.
  - `scripts/main` kaustas on rakenduse tagaosa ehk akna ja failimenüü kood.
  - `scripts/renderer` kaustas on põhiline programmikood.
- `config.js` sisaldab rakenduseüleseid valikuid.
- `ui-helper.js` on kasutajaliidese abifunktsioonide jaoks (nt elemendi keskele joondamine).
- `utility.js` sisaldab üldiseid funktsioone, mida kasutatakse mitmes kohas ja mida on kasulik ülejäänud rakendusest eraldi hoida juhuks kui implementatsioon muutub (nt ajatempli loomine).
- `tests` kausta lähevad loodud testifailid.
- `views` kaustas on HTML failid vaadete jaoks. Kuna põhitöö toimub testlehega, jääb autori vaadete kausta ilmselt kuni lõpuni põhiliselt rakenduse avaleht.

Lisaks eelpool mainitud kaustadele ja failidele on failipuus näha veel mõned. `.editorconfig` on EditorConfig[11] nimelise laienduse konfiguratsioonifail, mis näiteks annab võimaluse määrata erinevate failitüüpide jaoks erinevad treppimise skeemid ja eemaldab soovi korral ridade lõpust tühikud. `.gitignore` on Git[12] versioonihalduse eiratave failide konfiguratsioonifail, kus loetletud faile ei laeta repositooriumisse. `.jshintrc` on konfiguratsioonifail JSHint[13] nimelise laienduse jaoks, mis jälgib vigu JavaScriptis ja annab nendest reaalajas teada. `package.json` on `npm` paketihalduse selles projektis kasutusel olevate pakettide nimekiri. `README.md` on projekti kirjeldus ja `SCHEDULE.md` on autori tehtud plaan projekti edukaks kulgemiseks. `WHITEBOARD.md` on tahvli eest, kuhu jooksvalt arendusega seotud mõtteid lahti kirjutada. Niisugust faili on hea pidada, et mitte pauside ajal pooleli jäänud asju täiesti ära unustada. Lisaks aitab asjade samm-sammuline lahti kirjutamine oluliselt probleemide lahendamisele kaasa.

4.4 Rakenduse arhitektuuri põhielemendid

Hea visuaalse ülevaate sellest, kuidas projekt saab olema üles ehitatud, annab Illustratsioon 5.



Illustratsioon 5. Projekti arhitektuur

Võib mõelda, et *main*-protsess justkui ümbritseb *renderer*'i. Nooled näitavad, et protsessid saavad omavahel ka suhelda. Järgmise astmena sisaldab *renderer* JavaScripti programmikoodi. Üks osa sellest on testitava saidi sees ja laetakse enne tema enda koodi. Teine osa ümbritseb testitavat veebisaiti, salvestab ja taasesitab teste. Mõlemad koodi pooled suhtlevad üksteisega rakenduse töö käigus pidevalt.

Nagu on juba kirjeldatud eespool, Electroni kirjelduses (vt ptk 2.3), on Electroni projekti üheks põhiosaks *main* protsess ehk tagaosa, mis tegeleb rakenduse akendega ja nendega seotud funktsionaalsusega. Tavaliselt on see protsess esindatud `main.js` failis

11 http://editorconfig.org/
12 https://git-scm.com/
13 http://jshint.com/

(kuid võib muidugi olla ka mõnes teises failis). Autor jättis selleks failiks `main.js` ja asukohaks `scripts/main/main.js`.

Kuna tegemist on Electroni mõistes suhteliselt lihtsa ja olematu akende omavahelise suhtlusega rakendusega, sisaldab `main.js` ainult programmi käivitamisel uue akna loomise koodi ja `views/index.html` sinna sisse laadimist. Edasi läheb töö loodud akna *renderer*-protsessi kätte, mida on samuti juba põgusalt kirjeldatud Electroni peatükis.

Üheks oluliseks funktsioonaalsuseks, mis on aga ka `main.js` külge lingitud, on `menu.js` (`scripts/main/menu.js`), mis haldab akna „Fail" menüüd, kust on võimalik akna suurust määrata või rakendusest väljuda. Kuna menüü ei suhtle otseselt akna sees oleva sisuga ega muuda seda, on mõistlik tema poolt käivitatud tegevusi läbi viia kasutades rakenduse tagaosa. Menüü tegevused oleks võinud ka otse `main.js` faili kirjutada, kuid autor pidas mõistlikumaks eraldada nii konkreetselt piiritletud funktsionaalsus täitsa omaette faili ja linkida see `main.js` külge. Linkimine toimub `main.js` faili ülaosas käsuga `require(path.join(__dirname, 'menu'));`. `menu` on samas kaustas asuva faili nimi ilma laiendita ja `path.join()` hoolitseb reserveeritud muutujat `__dirname` kasutades selle eest, et failide linkimine erinevates operatsioonisüsteemides usaldusväärselt toimiks. Käsk `require()` teeb linkimise ja ei lase programmi tööl jätkuda kui laadimise hetkel peaks juhtuma, et `menu.js` faili ei leitud.

Rakenduse tagaosa ja eesosa saavad omavahel asünkroonseid sõnumeid saates suhelda. Kuigi eelmises lõigus on öeldud, et interaktsioonid menüüga otseselt akna sisu ei muuda, on testimise rakenduses siiski üks väike nüanss, mida on protsesside omavahelise suhtlusega hea realiseerida. Sel hetkel, kui kasutaja menüüst uue resolutsiooni valib, saadab *main* lisaks akna mõõtmete muutmisele *renderer*'ile teate, et akna suurust muudeti. Seda „kuuldes" arvutab eesosa uuesti `<webview>` mõõtmed, et testitav sait kataks alati akna sisu sajaprotsendiliselt. Seda on kasulik nii teha, kuna akna mõõtmete suurendamisel jääks testitav leht vastasel juhul väiksemana vasakusse nurka ja mõõtmete vähendamisel läheks osaliselt akna ääre taha peitu, tekitades kerimisribad.

Protsesside omavahelisest suhtlusest on autor eeskuju võtnud ja kasutab ka ise sündmusjuhitud arhitektuuri (ingl *event-driven architecture*). See tähendab, et kui mõni lehel olev element teeb mingi tegevuse ära ja selle tulemusel peavad teised elemendid

muutuma, ei käivita muutunud element ise otse teiste koodi, vaid saadab välja rakenduseülese teate, et ta muutus. Selle teate põhjal saavad teised elemendid otsustada, kas nad peavad oma olekut kuidagi muutma või mingi funktsiooni käivitama. Niisugust stiili kasutades on hea erinevaid funktsionaalsusi üksteisest lahus hoida. Samuti tundub autorile käesoleva projekti näitel, et niisugust arhitektuuri kasutades ei kulu liialt palju aega programmi osade organiseerimisele. Miinusena võib aga välja tuua, et pikad sündmuste jadad ja failide vahel „hüppamine" võivad pikemas perspektiivis kindlasti ka üsna segaseks minna.

Põhiline tegevus toimub ühe akna eesosa ehk tagaosa poolt aknasse laetud lehe „küljes", millega koos laetakse hulk skriptifaile. `index.html` faili sees on aknasisene menüü: navigeerimine, testi tegevused, taasesitus (vt Illustratsioon 6). Kuna navigeerimine ja testi tegevused muudavad sisu samal lehel, on neid võimalusel mõistlik realiseerida akna sees. Akna *native* menüüd kasutades tuleks kõik tegevused sõnumitega akna sisse saata, mis teeks süsteemi oluliselt keerulisemaks. Lisaks aknasisesele menüüle sisaldab `index.html` siin juba korduvalt mainitud `<webview>` märgendit, mis on midagi HTMLi `<iframe>` sarnast ja sisaldab endas testitavat lehte (või rakenduse käivitamisel tervituslehte `views/welcome.html`). Testlehe mõõtmed hoitakse kogu rakenduse töö aja niisugused, et leht täidaks koos sisese menüüga kogu akna pindala. Lisaks juba mainitule sisaldab `index.html` veel esialgu peidetud elemente dialoogide jaoks, mida vastavalt vajadusele sobiva sisuga täidetakse ja nähtavale tuuakse (nt testi nime sisestamise ja laadimise dialoogid).

Illustratsioon 6. Valmiva rakenduse avaleht koos menüüga

Missugused failid ja klassid täpselt on rakenduse töö tegemisel kaasatud, kirjeldab täpsemalt järgmine alapeatükk.

4.5 Koodi struktuur

Käesolevas alapeatükis kirjeldatakse täpsemalt, kuidas on autor koodi klasside ja failide vahel jaotanud. Eraldi tuuakse välja tähtsamad klassid ja olulisemad koodifailid, kuid mõned vähem olulised või lihtsama sisuga failid kirjeldatakse üldistatult.

Alapeatükis 4.3 on kirjeldatud projekti failistruktuur, kuid siin täpsustatakse vaid `scripts` kausta sisu. Täpsemalt `scripts/classes`, `scripts/main` ja `scripts/renderer` kaustades olevaid faile, sest need on need mis sisaldavad testimise loogikat ja rakenduse põhifunktsionaalsust.

4.5.1 *Main*-protsess ehk tagaosa

Electroni rakenduse akende halduriks on tema *main*- ehk põhiprotsess. Käesolevas rakenduses, nagu ka paljudes teisetes Electroni rakendustes, on see `main.js` failis, mis asub `scripts/main` kaustas.

`main.js` sisaldab programmi käivitamisel akna loomise loogikat. Lisaks määrab ta ära erinevad atribuudid nagu näiteks missugune fail aknasse laetakse ja kas akent on võimalik hiirega laiemaks venitada. Arenduse käigus liikusid siia ka funktsioonid testifailide salvestamiseks ja lugemiseks, kuna vastasel juhul oleks pidanud *renderer* küsima tagaosa käest õige kausta, mispeale *main*-protsess oleks pidanud selle tagastama ja siis oleks *renderer* selle läbi failisüsteemi käsu salvestanud. Mõistlikum oli saata testifaili nimi ja sisu tagaossa ning lasta see seal salvestada ilma mitmeid sõnumeid saatmata.

`menu.js` failis on defineeritud menüü mall (ingl *template*), mis on massiiv JavaScripti objektidest. Malliga määratakse, millised menüüjaotused aknale tekivad, millised alajaotused olemas on ja millised funktsioonid menüülinkide vajutamisel käivituvad. Lisaks autori poolt määratud funktsioonidele on menüü jaoks Electronis olemas ka hulk eelnevalt valmis programmeeritud levinud käske nagu „Edasi” ja „Tagasi”. Need teevad tavapäraste funktsioonide implementeerimise väga kiireks ja lihtsaks.

Lisaks menüü mallile sisaldab `scripts/main/menu.js` ka menüüdega suhtlemisel tehtavate funktsioonide definitsioone. Samas määratakse ka ära, et rakendus kasutab seal kirjeldatud menüü malli.

4.5.2 *Renderer*-protsessi ehk eesosa klassid

Rakendus on üles ehitatud niisuguselt, et enamik koodifaile sisaldavad funktsioonide kogumeid, kuna nendest ei ole vaja mitut eksemplari luua. Mõned osad on aga sellised, mida on mõistlik kirjeldada klassidena, et nendest saaks programmi töö käigus kergesti mitmeid koopiaid tekitada. Sellisteks on näiteks `Recording`, mis koondab endasse hulga tegevusi. Teiseks näiteks on tegevused ise, nagu `ClickAction`. Klassid paigutas autor `scripts/classes` kausta. Kuna klasse kasutatakse ainult *renderer* protsessi poolt, oleks võinud `classes` ka `scripts/renderer` kaustas olla, aga see ei ole väga oluline.

`click-action.js` defineerib, nagu nimigi ütleb, hiirevajutuse tegevuse (`ClickAction`). Ta sisaldab vajutuse *x* ja *y* koordinaate, vajutatud elemendi märgendi nime (*tagName*) ja tüüpi (*type*), mis on vajalik failist salvestuse tagasi lugemisel õiget tüüpi objekti loomiseks. Lisaks sisaldab ta tegevust `perform()`, mida kutsudes esitatakse tegevus testitaval lehel.

`el-check.js` sisaldab `ElCheck` klassi definitsiooni, mille instants hoiab rakenduse töö käigus andmeid ühe kontrolli tegevuse kohta. Ta sisaldab kontrollitava elemendi *x* ja *y* koordinaate, märgendi nime ja tüüpi. Lisaks nendele omab `ElCheck`'i eksemplar *checks* massiivi, milles on üks või rohkem objekte kujul `{ name: 'atribuudi_nimi', value: 'atribuudi_väärtus' }`, milliseid taasesitusel kontrollitakse. Üldjuhul on *name* väljas mõni elemendi HTML märgendi atribuutidest (nt *class*), aga kaks erijuhtu on väärtused *exists* ja *contents*, millede olemasolul saab `Player` aru, et tuleb teha teistsugune kontroll. Kontroll nimega *exists* kontrollib, kas salvestatud *x* ja *y* koordinaatidel on esituse ajal sama märgendiga element. *contents* viitab kontrollile, kas elemendi `textContent` on sama, mis salvestuse hetkel.

`input-action.js` defineerib `InputAction` klassi, mille eksemplar tekitatakse `Recording`'usse siis, kui kasutaja teksti sisestab. Tal on olemas elemendi *x* ja *y*, *input* (*string*, mis valitud elementi sisestatakse) ja tüüp. Samuti omab ta `perform()` meetodit.

`nav-action.js` kirjeldab navigeerimise tegevuse `NavAction`. See ei ole otseselt kasutaja poolt sisestatud, aga sisestatakse uue testi salvestuse alguses esimese tegevusena, et testi käivitamisel esimese sammuna soovitud lehele navigeeritaks. Vajaliku infona sisaldab ta aadressi (*url*) ja tüüpi. Tegevusena nagu eelnevadki, `perform()`'i. Hiljem lisandus arenduse käigus, et see tegevus lisatakse automaatselt ka

Enter klahviga vormi saates.

`scroll-action.js` kirjeldab `ScrollAction` objekti, mille kasutaja saab testi luues kaudselt sisestada. Kaudselt sisestamise all peetakse silmas seda, et iga kord kui kasutaja teeb mõne tegevuse, kontrollitakse, kas lehte on võrreldes viimase asukohaga keritud. Kui on, sisestab salvestaja enne kasutaja tehtud tegevust uue `ScrollAction`'i. Kerimistegevuse atribuutideks on *x* ja *y*, mis selle tegevuse puhul näitavad, millisele positsioonile tuleks avatud veebileht kerida. Samuti sisaldab ta tüüpi ja `perform()`'i.

`url-check.js` defineerib aadressi kontrollimise tegevuse `UrlCheck`, mis sisestatakse testile automaatselt iga kord kui pärast kasutaja tehtud tegevust hakkas testsait laadima ja lõpetas laadimise. Niimoodi saab enne järgmise tegevuse tegemist olla kindel, et navigeeriti testi salvestamisel soovitud kohta. Atribuutidena on tal sarnaselt `NavAction`'ile aadress ja tüüp. Meetodina on olemas `perform()`.

`recording.js` on definitsioon `Recording` objektile. See on objekt mis koosneb salvestuse nimest ja salvestatud tegevustest. Tal on olemas meetodid uue tegevuse lisamiseks, kõikide tegevuste korraga küsimiseks, lisatud tegevuste arvu küsimiseks ja ühe tegevuse järjekorranumbri järgi küsimiseks (vajalik taasesitamisel). Lisaks meetod nime muutmiseks ja meetod `toJSON()`, mis teeb objektist faili salvestamiseks sobiva objekti ja tagastab selle.

Lisaks eelnevale lisandus arenduse käigus kõigile tegevustele omadus *humanName*, mis ütleb iga tegevuse puhul, milline on tema nimi kasutaja jaoks. Selle asemel, et näidata kasutajale tagasiside ekraanil näiteks vastavalt nime „NavAction", näidatakse „Navigeerimine".

4.5.3 Rakenduse eesosa põhilised koodifailid

Käesolevas alapeatükis tutvustatakse faile, milledest mõnede sisu on vormistatud *singleton*'ide laadsetena (nt `Player` või `Recorder`). *Singleton* on klass, millest luuakse programmi töö käigus ainult üks eksemplar. Tavaliselt on tema konstruktor peidetud ja eksemplari küsimiseks on loodud spetsiaalne avalik meetod (Osmani, 2015). Teiste koodifailide sisu on aga vormistatud veebilehe laadimise lõppemise kuulariga ümbritsetud funktsioonide kogumitena. Kuna enamus funktsioone on tegevuste kuularitega (ingl *event handler*) veebilehe kasutajaliidese elementide külge ühendatud,

tundus loogiline, et suur osa koodist on lihtsalt loogiliselt erinevatesse failidesse jaotatud funktsioonide kogumid. Samas aga, kuna salvestada saab korraga ainult ühte testi ja esitada saab korraga samuti ainult ühte testi, paistab olevat igati loogiline luua mängija ja salvestaja *singleton*'idena.

Järgnevalt kirjeldatud failid asuvad `scripts/renderer` kaustas. Autor ei kirjelda neid siin tähestikulises järjekorras vaid proovib nad esitada tähtsuse järjekorras.

`test-site-listeners.js` on fail, mis laetakse enne testsaidi laadimist tema külge. Electron võimaldab `<webview>` märgendite vahele laetavale veebilehele enne tema enda skriptide laadimist välise skripti külge panna, mis siis koos külastatava saidi koodiga käivitub. Siin on kirjeldatud erinevate tegevuste „kuulamine", rakendusele saatmine ja rakenduselt vastu võetavate tegevuste esitamine. Selles failis on mõnes mõttes rakenduse kõige olulisem osa. Ilma siinse funktsionaalsuseta ei oleks saanud kas üldse või vähemalt mitte nii edukalt soovitud programmi luua.

`recorder.js` sisaldab koodi uue testi salvestamiseks. `Recorder` „kuulab" pärast salvestuse alustamise nupu vajutamist testsaidilt tulevat infot ja loob vastavalt sisendile tegevused. Lisaks sisaldab ta koodi salvestuse lõppedes salvestuse nime küsimiseks ja algatab ka testi faili salvestamise. Kui test on salvestatud, lülitub ta tagasi algseisu järgmist salvestuse algust ootama.

`action-network-listener.js` sees olev kood töötab koos `Recorder`'i või `Player`'iga. Kui kumbki nendest on töös, kuulatakse siin, kas pärast tegevuse salvestamist või esitamist hakkas testsait laadima. Salvestaja ja mängija suhtlevad siin oleva koodiga vahetult. Kui salvesti salvestab uue tegevuse, saadab ta välja rakenduseülese teate, et salvestati uus interaktsioon. Selle peale hakkab `action-network-listener.js` „kuulama", kas leht hakkas laadima. Kui hakkas, ootab ta ära, kuni leht on laadimise lõpetanud ja saadab välja teate, et lehe laadimine on lõppenud. Kui testitav leht ei hakanud üle võrgu ressursse laadima, saadab siinne kood välja teate, et laadimist ei järgnenud. Nende teadete peale saavad kas mängija või salvestaja järgmise tegevusega edasi minna. Niimoodi käib `Recorder`'i või `Player`'i ja `action-network-listener.js` vaheline suhtlus terve salvestamise või esitamise aja.

`player.js` on nagu nimigi ütleb, `Player` ehk mängija. Üldine mulje on üsna sarnane `Recorder`'ile, kuid suhtlemine testitava lehega käib teistpidi. Mängija saadab testlehele

pärast esituse algust ükshaaval esitamiseks tegevusi ja „kuulab” vastuseid, kas esitamine õnnestus või mitte. Vastavalt vastustele genereerib ta ka väljundi ja katkestab ebaõnnestumise puhul testi esitamise. Kui esitus on lõppenud, pöördub ta algseisu, oodates uue testi esitamise algust.

`action-parser.js` failis sisaldub `ActionParser`, milles on defineeritud erinevad tegevuste ja kontrollide tüübid. Kasutusel on see kood selleks, et failist välja lugedes erinevad tegevused ja kontrollid tagasi õigeteks programmi objektideks teha. Lisaks erinevatele tegevuste tüüpidele sisaldab ta meetodit `parse()`, mis ootab parameetrina failist tulnud JavaScripti objekti ja loob selle põhjal vastava tegevuse (näiteks `ClickAction` või `UrlCheck`) ning tagastab selle.

Lisaks siin kirjeldatud failidele on `scripts/renderer` kaustas veel kümmekond faili, mis sisaldavad põhiliselt nuppude vajutuste kuulareid ja dialoogide nähtavaks toomist või uuesti peitmist. Autori arvates ei ole need piisavalt huvitavad ega olulised rakenduse üldise idee seisukohalt, et siin neid eraldi välja tuua. Huvilised saavad koodi repositooriumis nendega ise tutvuda.

4.5.4 Täiendavad koodifailid

Faile, mis on otse `scripts` kaustas, `classes`, `main` ja `renderer` kaustade kõrval, võiks pidada justkui autori loodud laiendusteks. Üks fail on siin muidugi päris rakenduse spetsiifiline (`config.js`), kuid teisi võiks vabalt tõsta mõnda teise projekti ja nendes olevat koodi ka seal kasutada.

`config.js` sisaldab rakenduse parameetreid nagu kahe tegevuse taasesitamise vahel oodatav aeg. Mingisuguse variatsiooni sellisest failist leiab pea igast rakendusest, sest numbrilisi konstante ei ole mõtet programmikoodi (mitmesse kohta) kirjutada. Üheks põhjuseks on asjaolu, et kui väärtust on kasutatud rohkem kui ühes kohas, on tema muutmiseks vaja kõik kohad üles leida. Teisena, isegi kui väärtust on kasutatud ainult ühe korra, tuleks ta ilma konfiguratsioonifaili kasutamata muutmiseks koodi seest üles leida, mis võib muutmise tülikaks teha. Konfiguratsioonifaili pidamine teeb aga niisugused muudatused ja erinevate seadete katsetamise triviaalseks.

`ui-helper.js` koondab funktsioonid, mis aitavad kasutajaliidese elementide kuvamisel. Näiteks pannakse testi nime dialoog iga kord enne selle nähtavale toomist

akna raami suhtes nii vertikaalselt kui horisontaalselt keskele. Iga kord on seda mõistlik teha selleks, et dialoogi kõrgus võib sisu tõttu muutuda ja kui tema asukohta alati uuesti ei arvutataks, võiks ta sisu muutumisel keskkohast kõrvale jääda. Selline funktsioon on aga mõttekas kirjutada ühte kohta ja kasutada teda kui välist laiendust, sest miski tema sisus ei seo teda otseselt just selle rakendusega. Niimoodi saab näidiseks toodud funktsiooni abil akna või veebilehe keskele liigutada ükskõik milliseid elemente ja väga lihtsasti on võimalik seda koodi ka teistes veebiprojektides või Electroni projektides kasutada.

`utility.js` on koht umbes sarnaste asjade jaoks nagu `ui-helper.js`, aga siin olev kood ei ole seotud kasutajaliidesega vaid pigem programmi sisemise loogikaga. Hetkel sisaldab see ainult funktsiooni ajatempli (ingl *timestamp*) saamiseks. Seda võib samuti ilma probleemideta erinevates projektides kasutada. Üheks põhjuseks niisuguste funktsioonide koondamiseks ja ülejäänud koodist eraldamiseks ongi lihtne projektide vaheline jagamine. Lisaks ka võimalus ühe muudatusega kõikides kasutuskohtades implementatsiooni muuta. Kolmandaks ei ole selles failis sisalduvad funktsioonid samuti kuidagi otseselt käesoleva rakendusega seotud.

5 Valminud rakendus

Arendustöö tulemusena valmis selline rakendus, nagu esialgsed ideed ja soovid seda ette nägid. Ületamatut raskust või ootamatut tehnilist piirangut ei esinenud. Järgnevalt annab autor koos illustratsioonidega väikese ülevaate valminud rakenduse kasutamisest. Kui rakendus on käivitunud, on võimalik vasakule navigatsiooni välja sisestada aadress, navigeerida soovitud saidile ja alustada salvestust (vt Illustratsioon 7). Teise peamise variandina on võimalik paremalt rippmenüüst juba salvestatud test valida ja seda esitada või see näiteks kustutada. Kustutamisel küsitakse ka üle, kas tõesti sooviti valitud test kustutada.

Illustratsioon 7. Kasutajaliidese testimise rakenduse menüü

Salvestust alustades muutuvad kontrolli lisamise nupp ja salvestuse lõpetamise nupp aktiivseks ning elementidel klikkides, tekstiväljadesse teksti sisestades ja lehte kerides salvestatakse mainitud tegevused ükshaaval testi sammudena. Uue kontrolli sisestamisel tuleb pärast vastava nupu vajutust valida mõni kasutajaliidese element, mispeale avaneb hüpikaken, kust saab määrata, millised kontrollid läbi viiakse. Ülejäänud kuva on sel ajal deaktiveeritud (vt Illustratsioon 8). Kui kontroll on valitud ja lisatud, jätkub testi salvestamine.

Illustratsioon 8. Pealkirja elemendile kontrollide valimine

Testi salvestamisel on igal ajal võimalik vastava nupuga see lõpetada mispeale palub rakendus sisestada testi nime. Numbrid, mis testi nime taha salvestatakse, on praeguse ajahetke tunnustempel ja see on lisatud ainult selleks, et ei oleks vaja implementeerida kogu „Kas tahad olemasoleva faili üle kirjutada?” loogikat. Kuna ajatempel ei kordu kunagi, on programmi jaoks failinimed alati erinevad, isegi kui kasutaja olemasoleva nime sisestab. Nime sisestamise ajal on ülejäänud vaade deaktiveeritud.

Kui test on salvestatud, saab seda esitada. Testi esitamise ajal on ekraani keskel poolläbipaistev paneel infoga, mitu sammu on läbitud, millist hetkel läbitakse ja mitu on veel läbida (vt Illustratsioon 9).

Illustratsioon 9. Testi esitamine

Pärast testi edukat läbimist kuvatakse samas aknas vastav roheline tulemus. Kui aga üht tegevustest ei õnnestu edukalt läbida, peatatakse test ja näidatakse punast tulemust (vt Illustratsioon 10).

Illustratsioon 10. Vasakul testi õnnestumine ja paremal ebaõnnestumine

Selleks et näidata, et arendatud rakendus on võimeline teste salvestama ja taasesitama, salvestas autor kõik alapeatükis 3.1 Teststsenaariumid kirjeldatud testilood. Kõik kolm lugu said ka esitatud ja edukalt läbitud. Videosalvestised käsitletud testilugudest on YouTube'i esitusloendis, mis on kättesaadav aadressil: https://www.youtube.com/playlist?list=PLHqj_enz1cG-eZugesW_vYviHBOpN1i7Z. Videod näitavad lisaks esitusele ka testide samm sammulist salvestamist. Kõike alates rakenduse käivitumisest kuni testitulemuseni välja. Kolm videot kestavad kokku umbes 11 minutit ja annavad rakenduse kasutamisest ning testi esitamisest väga hea ülevaate.

6 Avatud lähtekoodiga tarkvara

Käesolevas töös valminud rakenduse kood avaldatakse avatud lähtekoodi (ingl *open source*) litsentsi alusel koodirepositooriumis aadressil https://github.com/kardoj/ui-tests. Selleks, et repositooriumis olev kood oleks tõesti avaldatud avatud lähtekoodi alusel, peab lisama litsentsi, mis ütleb et tarkvara võib vabalt kasutada, muuta ja avaldada (GitHub Inc, 2017). Autor valis oma rakendusele MIT litsentsi[14], kuna selle sisu tundus olevat piisav, arusaadav ja konkreetne. Selleks, et oma GitHubi repositooriumile sama litsents külge panna, ei ole vaja teha muud, kui lisada *LICENSE* või *LICENSE.txt* fail, mille sisuks kopeerida MIT litsentsi tekst ja asendada aastaarv ning autori nimi. Rohkem infot erinevate litsentside kohta GitHub keskkonnas ja mujal kasutamiseks saab GitHubi enda avaldatud selleteemalisest juhendist[15].

Projekti avatud lähtekoodiga avaldamisel on mitmeid eeliseid. Teised programmeerijad saavad sellisel juhul koodi täiendada, mis viib parema üldise koodi kvaliteedini. Samuti on võimalik, et keegi kasutab koodi mõne oma rakenduse laiendamiseks. Kui programm on teatud tüüpi, saab teistes projektides teda sõltuvusena (ingl *dependency*) kasutada. Vastavalt oma soovidele saab valida sobiva litsentsi, mis ühte või teist asja soodustab (Open Source Guides, kuupäev puudub). Lisaks on avatud lähtekoodiga rakenduse kasutajal piisava huvi või teadmiste korral võimalus ise veenduda, et rakenduse kood on üles seatud nii nagu lubatud. Näiteks kui Wire suhtlusprogrammi loojad lubavad, et sõnumid liiguvad krüpteeritult (Wire, kuupäev puudub), saab repositooriumis koodi uurida ja selle lubaduse õigsuses ise veenduda. Kõik eelnev võib projekti edule kaasa aidata ja oluline on märkida, et tarkvara avatud lähtekoodiga avaldamine ei eemalda võimalust sellega raha teenida.

Negatiivset poolt oma hobiprojektide avatud lähtekoodiga avaldamisel autori arvates ei ole. Pigem on võimalus selle kaudu ainult kuulsust võita. Küll aga võib tavakasutajate jaoks avatud lähtekoodiga tarkvara kasutamisel negatiivseks lugeda põhjaliku kasutajatoe puudumise. Suurettevõtted, kes oma tarkvara müüvad, pakuvad tavaliselt oluliselt paremat kasutajatuge ja abi, kui leiab keskmine kasutaja omal jõul veebist. Ja kui tegemist on väga vähe kasutatud projektiga, ei pruugigi sellele kasutusjuhist ega abi leiduda, mis välistab paljude inimeste kokkupuute võib olla hea tarkvaraga.

14 https://opensource.org/licenses/MIT

15 https://help.github.com/articles/licensing-a-repository/

Kokkuvõte

Käesolev bakalaureusetöö pakub manuaaltestijale vahendi, millega oma teste automatiseerida. Töö peamiseks eesmärgiks oli arendada rakendus, mille abil oleks võimalik testida veebisaidi kasutajaliidest, salvestades käigud ja taasesitades testi soovitud ajahetkel, lastes rakendusel kontrollida kas kõik toimub selliselt nagu testi salvestamise hetkel.

Töö käigus selgitati välja olulised kasutajaliidese elemendid ja funktsioonid, milledele keskenduda (hiireklikk, teksti sisestus, navigatsioon) ja nende põhjal arendati valmis rakendus, mille lähtekood on vabalt saadaval aadressil https://github.com/kardoj/ui-tests. Valideerimaks, et rakendus on tõesti esmasteks testideks kasutusvalmis, testiti töös välja toodud kolm stsenaariumit sisselogimine, otsingutulemuste kontroll ja menüülinkide olemasolu kontroll kasutades valminud rakendust.

Kuigi rakendus vajab kindlasti veel edasiarendusi (nt testi tagasiside detailide täiendamine ja kujunduse lisamine), õnnestus autori arvates projekt hästi. Tegeleti ainult nende tegevustega, mis märgataval kiirusel soovitud funktsionaalsuseni viisid ning seetõttu jäi visuaalne pool esialgu tahaplaanile.

Töö käigus selgus ühe olulise avastusena, et Electroni ja veebitehnoloogiaid kasutades on võimalik hämmastavalt kiirelt toimiva lahenduseni jõuda. Asjaolu, et info liigub töölaualt veebi ja märkimisväärne osa tänastest programmeerijatest tegeleb igapäevaselt veebitehnoloogiatega, kiirendab ja soodustab hübriidrakenduste arendust veelgi. Kui eelnevale lisada võimalus oma rakendus põhimõtteliselt 100% koodi taaskasutusega kolmele põhilisele operatsioonisüsteemile väljastada, ei oskagi palju rohkemat tahta. Kindlasti tasub uut projekti alustades mõelda, kas see võiks olla kandidaat Electronile.

Kuigi autoril on plaan käesoleva töö käigus valminud rakendust ka edaspidi vabadel hetkedel täiendada, oleks siin hea teha tagasivaade, mida võiks teha teisiti, kui täna samasugust projekti uuesti otsast alustaks. Esimesena tuleb mõttesse koodi struktuur. Koodi on proovitud hoida loogilise ja lihtsana, kuid tähtaja lähenedes on mõned osad pea märkamatult koledamaks muutunud. Osa segadust tuleneb kindlasti autori vähesest kogemusest, kuid osa ka natukesest ajapuudusest või lohakusest (nt kasutajaliidese elementide tihe sidumine koodiga). Kohati oli raske koodi kirjutamise käigus aru saada,

kus tuleks hoog maha võtta, natuke mõelda ja alles siis jätkata.

Kokkuvõttes on autor siiski tulemustega rahul ja loeb eesmärgi luua kasutuskõlblik veebisaidi kasutajaliidese testimise rakendus täidetuks. Projekti käigus kogunes palju uusi teadmisi ja valmis rakendus, mida portfoolios eksponeerida. Tulevikuplaanide hulka kuuluvad koodi ümberstruktureerimine (ingl *refactoring*), kujunduse loomine ja inglise keele lisamine. Lisaks on kõik huvilised oodatud repositooriumisse leitud vigasid postitama.

Kasutatud kirjandus

McConnell, S. (2004). Code Complete: A Practical Handbook of Software Construction. Washington: Microsoft Press

Ghost Inspector. (2017). *Pricing & Plans - Ghost Inspector.* Loetud addressil https://ghostinspector.com/pricing/

Finley, K. (2016). *JavaScript Conquered the Web. Now It's Taking Over the Desktop*. Loetud addressil https://www.wired.com/2016/05/javascript-conquered-web-now-taking-desktop/

Node.js Foundation. (2017). *Node.js*. Loetud addressil https://nodejs.org/en/

Electron. (kuupäev puudub). *Quick Start | Electron*. Loetud addressil https://electron.atom.io/docs/tutorial/quick-start/

Vilches, J. (2016). *Make Chrome Run Faster and Keep RAM Usage Under Control*. Loetud addressil http://www.techspot.com/article/1193-chrome-performance-memory-tweaks/

Rittel, H., Webber, M. (1973). Dilemmas in a General Theory of Planning. *Policy Sciences 4*, 161.

Osmani, A. (2015). *Learning JavaScript Design Patterns*. Loetud addressil https://addyosmani.com/resources/essentialjsdesignpatterns/book/#singletonpatternjavascript

GitHub Inc. (2017). *Licensing a repository - User documentation*. Loetud addressil https://help.github.com/articles/licensing-a-repository/

Open Source Guides. (kuupäev puudub). *The Legal Side of Open Source - Open Source Guides*. Loetud addressil https://opensource.guide/legal/

Wire. (kuupäev puudub). *Simple, secure, private—always end-to-end*. Loetud addressil https://wire.com/en/privacy/

Summary

Development of Website User Interface Testing Software

This bachelor’s thesis gives manual software testers a tool to automate their tests with. The main goal of the project was to develop an application with which it would be possible to test webpage user interface elements and their functionality. The user would be able to record a series of interactions, save them as a test and be able to play it back whenever he or she so chooses. On playback the application would check whether all the steps carried out as they did when the test was recorded.

An important part of the project was to find out which user interface elements and functionalities were the most important to focus on. They were mouse click, text input and navigation. The program was developed while keeping those elements in mind. Source code of the application is available to everyone under MIT open source license here: https://github.com/kardoj/ui-tests. To make sure that the program was capable of recording and replaying tests, a few of the test cases (signing in, search results validation and menu links availability on different pages) were described early on in the project and tested later using the application.

Although the first version of the software product needs more work (e.g. more detailed test feedback and better user interface), the author is sure that the project was a success. The author was mainly focused on working on parts of the software which would bring the result closer to MVP (minimal viable product) as fast as possible, which is the reason why user interface was not worked on as much.

One key discovery that was made during the project is the fact that Electron and web technologies provide a very fast way of development. All the information is moving from the desktop to the web and a lot of programmers are very familiar with web technologies which makes the development process even faster. And the fact that with Electron it is possible to build an application from the same source code for all three main operating systems reduces development time even more. It is definitely worth considering Electron and web technologies when starting a new software project.

When looking back and thinking about what should have been done differently one

thing comes to mind – the structure of the code. Although the autor tried to keep everything nice, clean and easily understandable, some parts of the program degraded seemingly mysteriously when the deadline came closer. Author's limited experience is surely responsible for some of that mess but some of it is also due to laziness (tight coupling of user interface elements with the code for example) and running out of time. It was sometimes hard to decide when it would have been better to think for a moment instead of rushing with the implementation.

In conclusion the author is still pleased with the result and counts the goal to develop a usable website user interface testing application as achieved. A lot of new things were learned during the development and the program is definitely a nice addition to author's portfolio.

Future plans include refactoring the code, the design of the user interface and English language support. Everyone is welcome to file an issue in the repository with any bugs found in the application.