TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Eva-Anna Klugman 206644IABB

Kasutajaliidese automaattestimise raamistiku juurutamine infotehnoloogia idufirma näitel

Bakalaureusetöö

Juhendaja: Jekaterina Tšukrejeva
Magistrikraad

Tallinn 2023

Autorideklaratsioon

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

Autor: Eva-Anna Klugman

21.03.2023

Annotatsioon

Töö autor töötab infotehnoloogia ettevõttes Mrpeasy OÜ, kus otsustati juurutada automaattestimise kvaliteeditagamise protsessi. Antud töö eesmärgiks on luua projekt automaattestide jaoks, milles on seadistatud integratsioon sidusarenduskeskkonnaga. Samuti dokumenteerida kõiki juurutamise samme piisavalt põhjalikult, et valminud tööd oleks võimalik kasutada juhendina ka teistes taolistes projektides.

Töö tulemuseks on projekt, milles on konfigureeritud automaattestimise raamistik Cypress koos testide tulemuste raporteerimisriistadega Mochawesome ja cypress-terminal-report. Projektis on seadistatud integratsioon sidusarenduskeskkonnaga, kasutades Dockeri tehnoloogiat ja Bitbucket Pipelines teenust, ning installeeritud arendusprotsessi hõlbustavaid tööriistu Prettier ja Husky. Töö projekt seadistati vastavalt automaattestide arendajate soovidele ning varasemalt loodud kirjandusele.

Lõpptulemust valideeriti vastuvõtukriteeriumite abil, mida koostati, lähtudes ettevõtte soovidest ja vajadustest. Lisaks sellele viidi läbi intervjuud automaattestide arendajatega, et määrata projekti kasutuskõlblikkust ning -mugavust. Projekti kasutus ja sidusarenduskeskkonna seadistus oli kõrgelt hinnatud arendajate poolt.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 29 leheküljel, 6 peatükki, 20 joonist, 1 tabel.

Abstract

Implementation of a user interface automation testing framework on the example of an information technology start-up company

The author is an active employee of Mrpeasy OÜ, a company that decided to implement automation testing into their quality assurance process with the goal of increasing the level of software testing productivity and quality.

Despite the rapidly increasing popularity of automated software testing, configuring the project to perfectly accommodate the needs of developers is a burdensome task that needs a broader knowledge of different technologies to succeed.

The objective of this thesis work is to create and configure the project for automatic tests and, in addition, to document every step of the implementation process with precision that will enable this study to be used as a guide in similar projects.

The result of the study is deployed to the cloud repository project. The project includes the installed test automation framework Cypress and the configured CI/CD deployment using Docker technology and Bitbucket Pipelines service.

The project incorporates multiple additional tools, including test results reporting utilities Mochawesome and cypress-terminal-report. To facilitate the process of automatic test development, the project contains configured packages Prettier and Husky.

The result was validated based on the criteria compiled according to the needs of the company. Additionally, two rounds of interviews were conducted with two quality assurance engineers to analyse the project's usability and value of all the additional packages used. The resulting project was highly praised by the developers.

The thesis is in Estonian language and contains 29 pages of text, 6 chapters, 20 figures, 1 table.

Lühendite ja mõistete sõnastik

Android	Mobiilsete seadmete operatsioonisüsteem, mida kasutatakse näiteks Samsung ja Google ettevõtete toodetes.
API	*Application Programming Interface,* kirjeldus viisist, kuidas kaks ja rohkem seadet saavad üksteisega kommunikeerida.
CI/CD	*Continuous Integration/ Continuous Delivery*, sidusarendus.
CRM	*Customer Relationship Management,* kliendisuhete haldus.
E2E	*End to End*, kasutatakse kasutajaliidese testide abil tarkvara läbivtestimise kontekstis.
ES	*EcmaScript*, JavaScripti standard.
Git	Harutatud versioonihaldus süsteem.
Hook	Viis, kuidas muuta tarkvara käitumist, reageerides kindlaks määratud sündmustele.
IOS	Mobiilsete seadmete operatsioonisüsteem, mida kasutatakse Apple ettevõtte toodetes.
json	Tekstfailide formaat, mis edastab informatsiooni võtmete ja väärtuste paaridena.
LTS	*Long Term Support,* pikaajaline tugi.
npm	Pakettide haldus süsteem, mida vaikimisi kasutatakse Node.js keskkonna jaoks mõeldud pakettide puhul.
Pipeline	Termin, mis sidusarenduse kontekstis tähendab keskkonda, mille abil saab käivitada, muuta või edastada koodi.
Regressioonitestimine	Varasemalt valminud ja testitud tarkvara veel kord kontrollimine pärast uuenduste lisamist, et teha kindlaks, et antud funktsionaalsus pole mõjutatud ning töötab nagu varem.
ssh	*Secure Socket Shell*, turvalisuse protokoll.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Suure konkurentsi tõttu infotehnoloogia valdkonnas, omab ettevõtte maine väga suurt tähtsust turul positsioneerimisel. Sellepärast peab tarkvara kvaliteeditagamise protsess olema väga hästi läbi mõeldud, kuna ebastabiilne programm vähendab teenuse usaldusväärsust ning ettevõte võib kaotada nii olemasolevaid kui ka potentsiaalseid kliente.

Vaatamata sellele pole haruldane juhus, et ettevõtte piiratud ressursid ei võimalda teha piisava regulaarsusega täielikku regressioontestimist. See on tõsi ka antud töö tellija jaoks, kelleks on ettevõte Mrpeasy OÜ. Tegemist on Eestis asuva infotehnoloogia idufirmaga, kes tegeleb pilvepõhise tootmise planeerimise tarkvara arendamisega. Aastatega on pakutava teenuse funktsionaalsus mitu korda suurenenud, mis teeb tarkvara testimise protsessi järjest keerulisemaks.

Protsessi aeglustab ka see, et uue funktsionaalsuse arendamisel peavad manuaalsed testijad sageli vaatama mitu korda läbi sama testimisstsenaariumi. Seda sellepärast, et peale iga uuendust peab testija tegema kindlaks, et varem valmis saadud funktsionaalsus pole mõjutatud ning töötab endiselt korrektselt. Sama töö mitu korda läbitegemine on väsitav, ajakulukas ning pidurdab tarkvarauuenduste väljalaske.

Mainitud probleemide tõttu võeti vastu otsus juurutada automaattestimist tarkvara kvaliteeditagamise protsessi, mis selle abil peab muutuma tõhusamaks, võimaldama kiiremat programmiuuenduste väljaminekut ning looma võimalust teha eelnevalt võimatut pidevat regressioonitestimist, mis aitab kindlustada teenuse katkestamatu töö.

Kasutajaliidese automaattestimise raamistiku installeerimine on lihtne, kuid testide koodi jaoks projekti seadistamine on uurimist ja lisateadmisi nõudev protsess. Põhjalikult läbimõeldud projekti konfiguratsioon võimaldab arendajatel saavutada suuremat produktiivsust ning ennetab mitmeid probleeme, mida on projekti kasvamisega aina keerulisem lahendada.

Antud töö eesmärgiks on valmistada projekt tulevaste automaattestide jaoks, mis on täielikult konfigureeritud ja kasutamiseks valmis. Lisaks sellele dokumenteerida protsessi käigus samme piisava põhjalikkusega, et valmivat tööd saaks kasutada automaattestimise juurutamise juhendina ka teistes taolistes projektides.

Töö tulemuseks on pilvepõhisesse repositooriumisse paigaldatud projekt automaattestide jaoks, kus on juurutatud JavaScriptil põhinev raamistik Cypress. Projekt seadistatakse vastavalt arendajate soovidele ning varasemalt loodud kirjandusele. Projektis on konfigureeritud sidusarenduskeskkonnas automaattestide käivitamine, kasutades sidusarenduse vahendit Bitbucket Pipelines ja Dockeri tõmmist, kuhu komplekteeritakse kõik vajalik testide käivitamiseks masinatel, mis toetavad Dockerit.

Lisaks eelnevalt mainitud projekti seadistusele, võetakse kasutusele mõned testi tulemuste raporteerimisutiliidid ning arendust hõlbustavad tööriistad. Projekti kergemaks kasutuselevõtmiseks ettevõttes luuakse põhjalik dokumentatsioon, mis sisaldab installeerimisjuhendit. Lõppeesmärgiks on saavutada automaattestide arendajate rahulolu projekti kasutamise mugavuse aspektist.

Töö koosneb neljast põhilisest peatükist. Esimeses peatükis antakse ülevaade olemasolevatest akadeemilistest ja praktilistest töödest sarnastel teemadel. Töö teises peatükis kirjeldatakse ettevõtte vajadusi, pannes paika lõpptulemuse vastuvõtukriteeriumid ning põhjendades tehnoloogiate valikut. Kolmas peatükk on pühendatud projekti realisatsiooni kirjeldamisele. Selle käigus tuuakse välja kõik projekti loomise ja seadistamise sammud koos koodi näidete ning seletustega. Töö neljandas peatükis valideeritakse saadud tulemused ning arutletakse antud töö edasiarendamise võimalustest.

2 Ülevaade olemasolevatest töödest

On kirjutatud mitmeid põhjalikke töid, mis keskenduvad automaattestide juurutamisele erinevates ettevõtetes ja projektides. Üheks selliseks tööks on Vaasa ülikoolis lühendiga VAMK kaitstud Malika Tasnim Taky diplomitöö, mis annab põhjaliku ülevaate Cypress testimisraamistiku juurutamisest ühes Nokia ettevõtte projektis [1].

Ulatusliku ülevaate automaattestimise raamistiku juurutamisest annab 2022. aasta jaanuaris Tallinna Tehnikaülikoolis kaitstud Kristo Loidu bakalaureusetöö „Kasutajaliidese automaattestid Valitsusportaali näitel“, mis erinevalt sarnastest töödest pöörab tähelepanu ka automaattestide integreerimisele arendustsüklisse CI/CD keskkonna ja Dockeri abil.

Mitmed akadeemilised tööd ja artiklid keskenduvad kaasaegsete kasutajaliideste automaattestimise raamistike võrdlemisele. Sellisteks infoallikateks on näiteks 2022. aasta Tallinna Tehnikaülikooli magistritöö „Läbivtestide testimisvahendi valik veebirakenduse näitel“ [2], IEEE Xplore digitaalses kogus leiduv Mediterranean Conference on Embedded Computing (MECO) konverentsi toimetis „A Comparative Study of UI Testing Framework“ [3] ja raamatu „A Frontend Web Developer's Guide to Testing“ [4] kolmas ja neljas peatükk.

Huvipakkuv uurimus on esitatud ka Aalto ülikoolis aastal 2021 valminud magistritöös pealkirjaga „A developer-friendly automated web GUI test strategy“ [5]. Antud töös autor keskendub üldtuntud probleemile, et mõnikord arendajad pole innustatud kirjutama automaatteste, mille tõttu püütakse lõpetada testide kirjutamise etapp võimalikult vähese pingutusega. Töö käsitleb meetmeid, kuidas on võimalik muuta automaattestimise strateegiat arendajasõbralikumaks ja seeläbi suurendada testide kirjutajate produktiivsust.

3 Ettevõtte vajaduste analüüs

Järgnevas peatükis tutvustakse lähemalt testitavat rakendust ja pannakse paika automaattestide juurutamise edukuse mõõdikud, mida kasutatakse hiljem analüüsimisel. Viimases alapeatükis tutvustatakse ja põhjendatakse projekti tööriistade valikut.

3.1 Testitava rakenduse kirjeldus

Tegemist on pilvepõhise toomisplaneerimistarkvaraga, mis on suunatud tootmisettevõtetele suuruses 10 kuni 200 töötajat. Teenus toetab kõiki modernseid laialtkasutatavaid veebilehitsejaid. Samuti on olemas eraldi rakendus mobiilseadmetele, mis toetab nii Androidi kui ka IOS operatsioonisüsteeme.

Programm on jaotatud kaheksaks peamiseks mooduliks [6]:

- *CRM* – müügi ja tellimuste haldus
- *Production Planning* – tootmise planeerimine ja haldus
- *Stock* – laohaldus ja inventuur
- *Procurement* – ostujuhtimine
- *Dashboard* – kiirülevaade põhilistest näidikutest
- *My production plan* ja *Internet-kiosk* – tootmisoperatsioonide raporteerimine reaalajas
- *Settings* – programmi seadistused koos konto, kasutajate ja andmebaasi haldusega
- *Accounting* – raamatupidamine

Ettevõte pakub neli erinevat hinnapaketti. Igale hinnapaketile vastab erinev lisanduv funktsionaalsus, mida saab reguleerida programmi seadistustes.

Tarkvaral on mitu integratsiooni väliste teenustega. Peamisteks integratsioonivaldkondadeks on:

- Raamatupidamistarkvarad, nagu näiteks QuickBooks Online, XERO
- Internetipoode pakuvad teenused, nagu näiteks Shopify, Magento
- Välised CRM tarkvarad, nagu näiteks Pipedrive

- Välised laohaldustarkvarad, nagu näiteks Ware2Go

Samuti toetatakse API integratsiooni, kuid ainult kõige kallimas hinnapaketis.

Tarkvaraarenduseks kasutatakse mitu serverit:

1. Testserver koodnimega *test,* mis on täielik peamise serveri repliik.
2. Testserver koodnimega *dev*, mida kasutatakse peamiselt suurte muudatuste testimiseks, nagu näiteks infrastruktuuri uuendused.

3.2 Töö vastuvõtukriteeriumid

Sissejuhatuses oli mainitud, et tellija ettevõte otsustas sisse viia automaattestimist kvaliteeditagamise protsessi. Antud töö vastuvõtjateks on ettevõtte automaattestide arendajad. Tellija soove arvestades oli koostatud nimekiri kriteeriumitest, mis peavad olema täidetud, et töö lõpuks valmiv projekt oleks vastuvõetav:

1. Automaattestide loomiseks peavad arendajad omama teadmisi ainult JavaScript keelest, Cypressist ning *git*'ist.
2. Automaatteste peab olema võimalik jooksutada sidusarenduskeskkonnas.
3. Automaattestide arendajad peavad kiitma projekti kasutuskõlblikuks ning mugavaks kasutamiseks.

Esimene punkt nimekirjast viitab sellele, et automaattestide arendajatel ei pea olema muid teadmisi, kui teadmised koodi kirjutamisest kasutusel olevatest keeltes, milleks on JavaScript ja TypeScript, kasutades Node.js keskkonda. See tähendab, et projekti vastuvõtvatel arendajatel ei pea olema teadmisi projekti loomisest, Cypress raamistiku juurutamisest, Dockerist ega sidusarenduskeskkonna seadistamisest.

Teise kriteeriumi järgi valmis projekt peab olema integreeritud sidusarenduskeskkonnaga ehk võimaldama jooksutada projektis sisalduvaid automaatteste CI/CD-s. Antud projekti raames luuakse algne sidusarenduskeskkonna konfiguratsioon, mis võimaldab hiljem kergesti lisada *pipeline*'e vajaduspõhiselt.

Lõpptulemuse kvalitatiivseks hindamiseks kasutatakse intervjueerimise meetodit. Ettevõtte automaattestide arendajaid küsitletakse kaks korda: kohe pärast projekti üleandmist ja kaks kuud hiljem, kui on kirjutatud esimesed automaattestid.

3.3 Tööriistade kirjeldus

Tööriistade puhul tuli teha valikuid järgmistes kategooriates:

- Raamistik, mis võimaldaks kergesti luua kasutajaliidese automaatteste
- Programmeerimiskeel, mis sobiks raamistikuga kokku
- Pilvepõhine repositoorium ja sidusarenduskeskkond
- Automaattestide tulemuste raporteerimisutiliidid
- Arendusprotsessi hõlbustavad tööriistad

Järgnevates alapeatükkides nimetatakse projektis kasutatud tööriistu ning põhjendatakse nende valikuid.

3.3.1 Automaattestimise raamistik ja vastav käituskeskkond

Töö autori kolleegilt oli tellitud uuring, mille käigus võrreldi kasutajaliidese automaattestimise raamistikke ning valiti nende seast ettevõttele kõige paremini sobiv.

Uuringu tulemusena selgus, et ettevõtte eesmärkide täitmiseks sobivaimaks lahenduseks on Cypress raamistik, mis omab omapärast kergelt õpitavat süntaksit. Antud tulemust toetavad ka teised uurimused, mis olid nimetatud varasemate tööde ülevaate alapeatükis, mis nimetavad Cypressit üheks parimatest testimisraamistikest, mis on tänapäeval saadavad [2], [3].

Cypress on kirjutatud JavaScripti keeles ning kasutab käituskeskkonda Node.js, mis võimaldab käivitada JavaScripti ilma veebilehitsejata, näiteks arendaja arvutis. Cypressi testid jooksevad veebilehitsejas ning nende käivitamiseks on kaks moodust: koos veebilehitseja graafilise liidesega või ilma ehk *headless* töörežiimis. Testide loomisel on arendajatel mugavam kasutada testide käivitamist veebilehitseja graafilise liidesega, kuna see võimaldab jälgida tarkvara käitumist testimise vältel. Ilma graafilise kasutajaliideseta käivitamist kasutatakse tihedamini näiteks sidusarenduskeskkonnas.

3.3.2 Programmeerimiskeeled

Cypressil on enda süntaks, mis on abstraktsioon JavaScriptist. See võimaldab kasutada selle süntaksi kõrval ka tavalist JavaScripti koodi, näiteks andmete genereerimiseks või arvutuste tegemiseks.

Lisaks eelmainitule on Cypressi dokumentatsioonis kirjas, et raamistik toetab ka TypeScript keele kasutust [7]. TypeScript on JavaScripti põhjal loodud keel, mis toetab JavaScripti funktsionaalsust, kuid lisab sellele tüüpide kontrolli [8].

TypeScripti kasutus projektis polnud algselt kavandatud, kuid arendajad suhtusid positiivselt ettepanekusse võtta antud tehnoloogia kasutusele. Ettepanekut toetas ka iga-aastane küsitlus *State of JavaScript 2022,* milles küsimusele „Mis praegusel hetkel puudub JavaScript keeles?“ valisid 55,7% vastajatest vastuseks staatiliste tüüpide olemasolu [9]. Otsust kinnitas ka uuring „To Type or Not to Type? A Systematic Comparison of the Software Quality of JavaScript and TypeScript Applications on GitHub“, mille tulemusena leiti, et TypeScripti kasutamine parandab tunduvalt koodi loetavust ja kvaliteeti [10].

3.3.3 Pilvepõhine repositoorium ja sidusarendus

Repositooriumi valik oli tehtud tuginedes ülejäänud tehnilise osakonna valikule. Bitbucket on Atlassian ettevõtte poolt pakutav teenus, mis võimaldab paigutada koodi pilvepõhisesse repositooriumisse ja kasutada *git*'i [11]. Seda kasutatakse ka ettevõtte peamise tarkvara jaoks.

Sidusarenduskeskkonnaks valiti Bitbucket Pipelines, mis on samuti Atlassiani poolt pakutav teenus [12]. Põhjuseks oli selle kasutamise mugavus koos Bitbucket teenusega. Seda kasutatakse ka teistes ettevõtte projektides.

Sidusarenduse kasutamiseks rakendati Dockeri tehnoloogiat, mis võimaldab käivitada koodi virtuaalmasinaga sarnases keskkonnas [13]. Cypressil on ettevalmistatud ja saadavad Dockeri tõmmised, mida saab kohandada enda vajaduste põhiselt [14].

3.3.4 Automaattestide tulemuste raporteerimistööriistad

Automaattestide kasu seisneb tulemustes, mille tõttu nende ülevaatamine peab olema võimalikult lihtne ja mugav. Kui test kukub läbi, siis arendajatele on abiks igasugune teadmine, näiteks mis testisisestest käskudest ebaõnnestus või mis hetkel see juhtus. Kui testid käivitatakse ilma veebilehitseja graafilise liideseta, trükitakse testide tulemused standardsesse väljundisse, kus navigeerimise ja eksportimise võimalused on väga piiratud.

Mochawesome on testitulemustest raporti koostav tööriist, mis võimaldab saada paremat ülevaadet testitulemustest tänu filtreerimise võimalusele ja värvikodeeringutele [15]. Antud projektis seda kasutatakse põhiliselt sidusarenduses.

Utiliit Cypress-terminal-report trükib testide jooksmise ajal käivitatud käskude logisid standardsesse väljundisse [16]. See võimaldab testi läbikukkumise korral näha detailset infot testi käigust, isegi kui see oli käivitatud sidusarenduskeskkonnas.

3.3.5 Arendusprotsessi hõlbustavad tööriistad

Aleksi Vuorjoki magistritöö Aalto ülikoolist keskendub arendajate efektiivsuse uurimisele automaattestide loomisel. Uuringu tulemusena selgus, et testide koodi loetavus mõjutab väga tunduvalt arendajate produktiivsust. [5] Tuginedes sellele tööle, tehti otsus kasutada antud projektis arendusprotsessi hõlbustavaid tööriistu Prettier ja Husky.

Prettier on koodi automaatne vormindaja, mis eemaldab esialgse koodi vormingu ning kindlustab ühtse stiili kasutuse terves projektis [17]. Prettier on üks populaarsematest JavaScriptiga koos kasutatavatest arendustööriistadest, mida tõestab 2021. aasta küsitlus *State of JavaScript,* milles 83,2% vastajatest märkisid, et kasutavad Prettieri [18].

Husky on samuti laialt kasutatav tööriist. Vastavalt selle Github leheküljele, millel on 29,3 tuhat tärni, Husky kasutuste arv ületab miljoni [19]. Husky võimaldab kasutada skriptide käivitamiseks *git hooks* tehnoloogiat [20]. Antud projektis kasutatakse seda koodi kontrollimiseks ja vormindamiseks koos *git commit* käsuga. Kontrollimiseks käivitatakse TypeScript keele kompilaatorit [21] ning vormindamiseks jooksutakse Prettieri lisandunud või muudetud koodiridadele. See võimaldab kindlustada, et kogu automaattestide kood, mis jõuab ühisesse repositooriumisse Bitbucket keskkonda, on formateeritud ning kontrollitud.

4 Realisatsioon

Antud töö raames projekti realiseeriti järgnevates etappides:

1. Tehti algne koodiprojekt repositooriumisse ja paigaldati vajalik tarkvara töö autori arvutisse.
2. Seadistati plaanitud projekti sõltuvused *npm*-i abil.
3. Tehti vajalikud seadistused, et automaatteste oleks võimalik jooksutada sidusarenduskeskkonnas.

Järgnevates alapeatükkides kirjeldatakse põhjalikult iga etapi realiseerimise protsessi.

4.1 Projekti loomine ja lokaalse töökeskkonna ettevalmistus

Esimese sammuna tekitati projekt pilvepõhisesse repositooriumisse Bitbucket keskkonnas. Töö autorile olid antud õigused projekti redigeerimiseks. Projekti allalaadimiseks kasutati *git*'i käsku `git clone` [22], mis laadis projekti töö autori arvutisse. Eelnevalt arvutisse oli genereeritud *ssh* võti, mida lisati Bitbucketi kontosse [23].

Projekti loomisel oli Bitbucketi poolt automaatselt genereeritud *.gitignore* fail. Antud failis on võimalik defineerida failid või kaustad, mis ei pea kuuluma projekti versioonikontrollisüsteemi *git* [24] ehk mis peavad jääma ainult lokaalsesse masinasse. Sellesse faili olid lisatud viited kaustadele, mis võivad hiljem tekkida Cypressi automaattestide käivitamise tulemusena ning sisaldama videosid ja ekraanitõmmiseid jooksutatud testidest.

```
# Videos, screenshots, and reports from tests
cypress/videos
cypress/screenshots
cypress/downloads
cypress/results
```

Joonis 1. Failisse „.gitignore“ lisatud read.

Järgmise sammuna laeti alla ja installeeriti Node.js käituskeskkonda koos selle paketihalduriga *npm*, mida kasutatakse hiljem projekti sõltuvuste haldamiseks. Node.js ja

npm-i allalaadimiseks kasutati OpenJs Foundation ametlikku veebilehte. Node.js versioonidest valiti versiooni märgendiga LTS, milleks töö valmimise hetkeks on 18.16.0 koos 9.5.1 *npm*-iga.

Npm-i projektisisese kasutamise jaoks on vajalik jooksutada projekti põhikaustas avatud terminalis käsk `npm init` [25], mis loob faili nimega „package.json“. Antud fail hoiab endas informatsiooni projekti sõltuvustest ning nende kasutatavatest versioonidest.

4.2 Projekti sõltuvuste seadistamine

Projekti sõltuvusteks on kõik tehnoloogiad ja utiliidid, mis on vajalikud projekti töö jaoks. Antud töö raames nendeks on:

- Cypress
- TypeScript
- Mochawesome
- Cypress-terminal-report
- Prettier
- Husky

Järgnevates alapeatükkides kirjeldatakse nende installeerimise ja seadistamise protsess. Kõik sõltuvused laaditakse alla kasutades *npm* paketihaldurit. Kõik projektis kasutatud paketid on kajastatud failis „package.json“ ning selle lõplik sisu on lisatud dokumendile Lisas 2.

4.2.1 Cypressi koos TypeScriptiga installeerimine ja seadistus

Juhised Cypressi installeerimiseks olid võetud ametlikust dokumentatsioonist. Selle juurutamiseks projekti arenduse sõltuvusena kasutati käsku `npm install cypress --save-dev`.

Cypressi käivitamiseks saab kasutada kas käsku `cypress open` või `cypress run`. Esimene käsk avab Cypressi tarkvara graafilise kasutajaliidese ning teine paneb testid jooksma ilma graafilise liideseta ehk *headless* töörežiimis. Esimesel avamisel tarkvara näitab, et on olemas kaks testimise liiki, mida on võimalik projektis seadistada: E2E ehk läbivtestimine või komponenttestimine. Antud projekti raames kasutatakse ainult läbivtestimist. Konfiguratsiooni valiku tegemisel Cypress vaatab, kas projektis on

TypeScript, ning kui ei ole, siis loob konfiguratsiooni JavaScripti põhjal. Kuna antud projektis kavatsetakse kasutada TypeScripti, siis Cypressi esimest avamist on mõistlik teha pärast TypeScripti installeerimist.

TypeScripti installeerimiseks käivitati käsk `npm install --save-dev typescript`. Antud käsk samuti juurutab TypeScripti arenduse sõltuvusena.

Kuna Cypress saab nüüd tuvastada, et projektis on TypeScript kasutusel, käivitati Cypressi tarkvara käsuga `cypress open`. Pärast läbivtestimise variandi valimist, loodi projekti esialgne kaustade struktuur koos põhiliste ja näidisfailidega (Joonis 2).

```
cypress.config.ts
  cypress/
    fixtures/
      example.json
    support/
      e2e.ts
      commands.ts
```

Joonis 2. Cypressi poolt loodud kaustade ja failide struktuur.

Fail „cypress.config.ts“ sisaldab Cypressi seadistusi ning võimaldab neid konfigureerida. See fail on algse sisuga, mida saab hiljem täiendada (Joonis 3).

```
import { defineConfig } from "cypress";

export default defineConfig({
  e2e: {
    setupNodeEvents(on, config) {
      // implement node event listeners here
    },
  },
});
```

Joonis 3. Faili „cypress.config.ts“ algne sisu.

Kaust „cypress“ on põhikaust testide ja nende abikoodi jaoks. See sisaldab kaustasid „fixtures“, mis on kaust ettevalmistatud andmete hoidmiseks, ja „support“, mis on kaust teste toetava koodi hoidmiseks. Kaustas „support“ on loodud failid „e2e.ts“ ja „commands.ts“. Faili „e2e.ts“ käivitatakse enne kõikide testide käivitamist, mis annab võimaluse sisestada sellesse koodi, mida on vaja käivitada enne automaattestide jooksutamist. Faili „commands.ts“ on võimalik kasutada eripäraste Cypress käskude loomiseks, mis on kasulik näiteks mitme koossobiva olemasoleva käsku kombineerimiseks üheks.

Kausta „cypress“ loodi kaust nimega „e2e“, mille eesmärk on hoida automaattestid teistest failidest eraldi. Kaustast „fixtures“ kustutati näidisfail „example.json“.

Cypressil on mitu kasulikku konfiguratsiooni valikut. Antud projektis kasutatakse järgmiseid seadistusi:

- „retries“ – Võimaldab defineerida, mitu korda testi käivitatakse uuesti ebaõnnestumise juhul. On võimalik seadistada erinevalt graafilises ja *headless* keskkondades. Vaikimisi on väärtuseks mõlema puhul null.
- „specPattern“ – Võimaldab defineerida, mis faile tuvastada automaattestide failidena. Antud projektis sellisteks failideks on need, mis on kaustas „e2e“ ning omavad lõppu “cy.ts”. Vaikimisi väärtuseks on “cypress/e2e/**/*.cy.{js,jsx,ts,tsx}”.
- „watchForFileChanges“ – Võimaldab seadistada, kas Cypress jooksutab testi automaatselt selle koodi muutumisel või mitte. Vaikimisi väärtuseks on “true”.

Kuna testitava tarkvaraga võib esineda korralisi käitumuslikke ebapärasusi, näiteks aeglase interneti ühenduse tõttu, oli tehtud otsus lasta kaks korda korrata sama testi olukorras, kui see kukub läbi *headless* režiimis. Arendajad samuti väljendasid soovi lülitada välja automaattestide käivitamise, kui Cypress on töötamas ning arendaja teeb koodis muudatuse.

Tuginedes mainitud soovidele täiendati Cypressi konfiguratsioonifaili „cypress.config.ts“ (Joonis 4).

```
import { defineConfig } from "cypress";

export default defineConfig({
  e2e: {
    setupNodeEvents(on, config) {
      // implement node event listeners here
    },
  specPattern:
      'cypress/e2e/**/*.cy.ts',
  },
  retries: {
    runMode: 2,
    openMode: 0
  },
  watchForFileChanges: false,
});
```

Joonis 4. Faili „cypress.config.ts“ täiendatud sisu.

TypeScript on projekti installeeritud, kuid töötamise jaoks vajab see konfiguratsioonifaili „tsconfig.json“. Cypressi dokumentatsioon soovitab paigaldada antud faili „cypress“ kausta, kuna siis ülejäänud projektis on vajadusel võimalik kasutada teist TypeScripti seadistust. Dokumentatsioonis on samuti leitav soovituslik faili sisu, mida kasutati antud projektis (Joonis 5), kuid erineva JavaScripti versiooniga. Originaalses failis oli kasutatud EcmaScripti versioon „ES5“, kuid projektis on kasutatud versioon „ES2021“, mis on uuem ja toetab rohkem funktsionaalsust. Samuti oli faili lisatud seadistus "moduleResolution", mis oli väärtustatud tekstiga "node". Antud säte annab TypeScripti kompilaatorile teada, et tegemist on Node.js käituskeskkonnaga.

```
{
  "compilerOptions": {
    "target": "ES2021",
    "lib": ["ES2021", "dom"],
    "types": ["cypress", "node"],
    "moduleResolution": "node",
  },
  "include": ["**/*.ts"]
}
```

Joonis 5. Faili „tsconfig.json“ sisu.

4.2.2 Mochawesome installeerimine ja seadistus

Testide tulemustest selge ülevaate saamiseks kasutatakse kolme utiliiti koos. Nendeks on „mochawesome“, „mochawesome-report-generator“ ja „mochawesome-merge“. Mochawesome loob eraldi raporti iga testfaili jaoks, mis ei sobi eesmärgiga saada üks põhjalik aruanne testide tulemustest. Mitme raporti kokkupanemiseks on olemas utiliit

„mochawesome-report-generator“ [26], kuid selle puuduseks on vajadus defineerida kõik failid, mis lähevad ühise raporti sisse. Kuna jooksutavate testide arv võib varieeruda, on selle tingimuse täitmine keeruline. Antud olukorras on abiks pakett „mochawesome-merge“ [27], mis moodustab kõikidest Mochawesome raportitest formaadis *json* ühise faili. Seda faili saab edastada tööriistale „mochawesome-report-generator“, mis moodustab sellest täiusliku raporti.

Kõikide nende pakettide korraga installeerimiseks kasutati käsku `npm install mochawesome mochawesome-report-generator mochawesome-merge --save-dev`. TypeScript tüüpide saamiseks kasutati käsku `npm install --save-dev @types/mochawesome`.

Cypressi dokumentatsioonis on leitav soovituslik seadistus [28], mida on vaja lisada Cypressi konfiguratsioonifaili „cypress.config.ts“, et Mochawesome oleks kasutatud vaikimisi raporteerimisriistana (Joonis 6). Antud seadistuse põhjal rapordi failid formaadis *json* luuakse kausta „results“, mida luuakse kausta „cypress“.

```
reporter: 'mochawesome',
reporterOptions: {
  reportDir: 'cypress/results',
  overwrite: false,
  html: false,
  json: true,
}
```

Joonis 6. Koodiread, mida lisati faili „cypress.config.ts“ Mochawesome raportite saamiseks.

Ülejäänud nimetatud utiliidid ei vaja eraldi seadistamist. Kuna nende kasutus on plaanitud sidusarenduskeskkonnas, siis selle näitamine ja seletamine tehakse sidusarendusele pühendatud peatükis.

4.2.3 Cypress-terminal-report installeerimine ja seadistus

Installeerimisjuhend on leitav „cypress-terminal-report“ GitHub leheküljel, kuid paketi seadistamisel töö autor kasutas uuemat JavaScripti süntaksit, kui on kasutatud juhendis, mille tõttu lõplik seadistuse kood on sellest erinev.

Antud utiliidi installeerimiseks projektis kasutati käsku `npm install --save-dev cypress-terminal-report`.

Cypressi konfiguratsioonifaili lisati „installLogsPrinter“ funktsiooni import alla laetud paketist (Joonis 7). Uuema süntaksi kasutus võimaldas vältida TypeScripti vigu, mis muidu oleksid tekkinud projekti ja utiliidi JavaScripti versioonide erinevuse tõttu.

```
import installLogsPrinter = require('cypress-terminal-
report/src/installLogsPrinter');

export default defineConfig({
  e2e: {
    setupNodeEvents(on, config) {
      installLogsPrinter(on, {})
    },
  },
  ...
});
```

Joonis 7. Funktsiooni „installLogsPrinter“ import ja kasutus failis „cypress.config.ts“.

Kohandatud süntaksit kasutati ka „installLogsCollector“ funktsiooni importimisel ja väljakutsumisel failis „e2e.ts“ (Joonis 8).

```
import * as installLogsCollector from 'cypress-terminal-
report/src/installLogsCollector';

installLogsCollector();
```

Joonis 8. Funktsiooni „installLogsCollector“ import ja kasutus failis „e2e.ts“.

Cypress-terminal-report võimaldab saada põhjalikumat logide väljundit. Lisaks logidele standardses väljundis, näiteks terminalis, on võimalik lasta kirjutada logisid failidesse formaatides *txt* ja *json*. Selleks on vaja seadistuses lisada väärtused valikutele „outputTarget“, „outputRoot“ ja „specRoot“. Lisaks saab omistada väärtust valikule „printLogsToFile“ tekstiga „always“, mis annab juhise alati kirjutada logisid failidesse. Logide suure pikkuse vältimiseks standardses väljundis, kasutatakse seadistust „compactLogs“, mis võimaldab defineerida, mitu käsku peab logima enne käsu ebaõnnestunuks lugemise. Koos sellega saab omistada väärtust valikule „outputCompactLogs“, mis võimaldab säilitada kõikide logide kirjutamist failidesse.

Tööriist oli konfigureeritud failis „cypress.config.ts“ funktsiooniga „installLogsPrinter“, võttes mainitud seadistamise võimalusi arvesse (Joonis 9).

```
installLogsPrinter(on, {
        specRoot: 'cypress/e2e',
        outputRoot: config.projectRoot +
'/cypress/results/',
        outputTarget: {
          'cypress-logs-txt|txt': 'txt',
          'cypress-logs-json|json': 'json'
        },
        printLogsToFile: 'always',
        outputCompactLogs: false,
        compactLogs: 10
      })
```

Joonis 9. Utiliidi „cypress-terminal-report“ seaded failis „cypress.config.ts“.

4.2.4 Prettieri installeerimine ja seadistus

Prettieri installeerimise juhendid on leitavad ametlikus dokumentatsioonis. Installeerimiseks kasutati käsku `npm install --save-dev prettier`. Pärast installeerimise lõppu loodi Prettieri konfiguratsioonifail „.prettierrc.json“ käsuga `echo {}> .prettierrc.json`. Failisse lisati konfiguratsiooni seadistusi vastavalt arendajate soovidele (Joonis 10).

```
{
  "useTabs": true,
  "tabWidth": 4
}
```

Joonis 10. Faili „.prettierrc.json“ sisu.

4.2.5 Husky installeerimine ja *pre-commit hook*'i seadistus

Juhendid installeerimiseks ja seadistamiseks on leitavad Prettieri ja Husky dokumentatsioonides. Koos Husky utiliidiga installeritakse ka pakett Lint-staged, mis võimaldab käivitada koos Prettieriga ka TypeScripti kompilaatorit, mis teostab koodi kontrolli. Husky automaatse installeerimise ja seadistamise jaoks kasutati käsku `npx husky-init && npm install`. See tõmbab alla Husky, loob kausta „.husky“, ning paigutab sinna algse *pre-commit hook*'i skriptifaili. Kaust „.husky“ pidi olema automaatselt seadistatud *git*'i *hook*'ide kaustaks. Kontrollimiseks saab kasutada käsu `git config core.hooksPath`, mille vastuseks peab tulema „.husky“.

Järgmise sammuna asendati „pre-commit“ skriptifailis automaatselt lisatud koodi vajaliku käsuga, et käivituks Lint-staged (Joonis 11), mille seadistamist kirjeldatakse järgmises lõigus.

```
#!/usr/bin/env sh
. "$(dirname -- "$0")/_/husky.sh"

npx lint-staged
```

Joonis 11. Faili „pre-commit“ sisu kaustas „.husky“.

Lint-staged paketi alla laadimiseks kasutati käsku `npm install --save-dev lint-staged`. Selle seadistust on vaja lisada faili „package.json“. Antud projekti raames lisatakse sellesse kaks käsku, milleks on TypeScripti kompilaatori käivitamine ja Prettieri jooksutamine (Joonis 12).

```
"lint-staged": {
    "**/*": [
      "bash -c \"tsc -p ./cypress/tsconfig.json --noEmit
--pretty\"",
      "prettier --write --ignore-unknown"
    ]
  },
```

Joonis 12. Lint-staged konfiguratsioon „package.json“ failis.

Husky ja Prettieri seadistamise tulemuste kontrollimiseks tehti *git commit* koos kõikide loodud failidega. *Git*'i konsoolis oli näha, et käsud olid jooksutatud vastavalt ootustele (Joonis 13).

```
[STARTED] Preparing lint-staged...
[SUCCESS] Preparing lint-staged...
[STARTED] Running tasks for staged files...
[STARTED] package.json — 12 files
[STARTED] **/* — 12 files
[STARTED] bash -c "tsc -p ./cypress/tsconfig.json --noEmit --
pretty"
[SUCCESS] bash -c "tsc -p ./cypress/tsconfig.json --noEmit --
pretty"
[STARTED] prettier --write ./cypress --ignore-unknown
[SUCCESS] prettier --write ./cypress --ignore-unknown
[SUCCESS] **/* — 12 files
[SUCCESS] package.json — 12 files
[SUCCESS] Running tasks for staged files...
```

Joonis 13. Osalised logid git konsoolist pärast *git commit*'i käivitamist.

4.3 Sidusarenduskeskkonna seadistamine

Järgnevates alapeatükkides kirjeldatakse, kuidas loodi Dockeri tõmmist koos Cypressi automaattestidega ning selle abil seadistati sidusarenduskeskkond Bitbucket Pipelines.

4.3.1 Dockeri tõmmise loomine

Cypressil on mitu ettevalmistatud Dockeri tõmmist, mida saab kasutada automaattestimise projektis [14]:

- „cypress/base:<Node.js versioon>“ – sisaldab operatsioonisüsteemi sõltuvusi, mis on vajalikud Cypressi jaoks;
- „cypress/browsers:<tag>“ – sisaldab veebilehitsejaid koos kõige mainituga eelmise tõmmise kohta;
- „cypress/included:<Cypressi versioon>“ – sisaldab Cypressi koos kõige mainituga eelmise tõmmise kohta.

Nimetatud Dockeri tõmmised on abiks projektispetsiifilise tõmmise loomisel, mis on vajalik sidusarenduskeskkonnas kasutamiseks. Kuna antud projektis Cypressi installeeritakse koos kõikide teiste projekti sõltuvustega, on mõistlik kasutada Dockeri tõmmist „cypress/browsers:<tag>“. Docker Hub leheküljel on leitavad kõik selle tõmmise versioonid, mida saab kasutada. Antud projekti käigus kasutatakse kõige uuemat saadaval versiooni, mis töö valmimise hetkeks sisaldab Node.js versiooni 18.16 ning Chrome, Firefox ja Edge veebilehitsejate versioone, mis algavad numbriga 112.

Dockeri tõmmise loomiseks loodi projekti põhikausta fail nimega „Dockerfile“, kuhu paigutati juhised tõmmise moodustamiseks (Joonis 14). Docker peab juhiste täitmiseks omama baastõmmist, mille põhjal neid täidetakse, mida defineeritakse käsuga FROM. Käsk WORKDIR paneb paika kausta, milles on võimalik käivitada järgmised käsud:

- RUN – võimaldab käivitada käske tõmmise loomisel;
- CMD – võimaldab defineerida käske, mis käivituvad vaikimisi konteineris, kuid mida on võimalik konteineri loomisel üle kirjutada;
- ENTRYPOINT – võimaldab defineerida käske, mis käivituvad konteineri loomisel ning mida pole võimalik muuta;
- COPY – võimaldab kopeerida faile ja kaustasid masinast tõmmisesse;
- ADD – võimaldab lisada faile ja kaustasid masinast või veebist tõmmisesse.

```
FROM cypress/browsers:node-18.16.0-chrome-112.0.5615.121-
1-ff-112.0.1-edge-112.0.1722.48-1

WORKDIR /project-base

COPY ["package.json", "package-lock.json*", "./"]

RUN npm pkg delete scripts.prepare && npm ci

COPY ["/cypress.config.ts", "./"]
COPY ["/cypress", "./cypress"]

ENTRYPOINT [ "npx", "cypress", "run" ]
CMD ["-b", "chrome"]
```

Joonis 14. Faili „Dockerfile“ sisu.

Fail „Dockerfile“ on vajalik tõmmise loomiseks. Antud projekti faili järgi tõmmise baaskaustaks on seatud kaust nimega „project-base“. Sinna kopeeritakse failid „package.json“ ja „package-lock.json“, mille põhjal teostatakse kasutatavate pakettide alla laadimist. Selle jaoks käivitatakse käsku `npm pkg delete scripts.prepare && npm ci`, mis esimese sammuna eemaldab „prepare“ skripti, mis muidu jooksutaks Husky utiliidi, ning installeerib sõltuvusi.

Pärast sõltuvuste installeerimist kopeeritakse tõmmisesse Cypressi konfiguratsioonfail ja kaust „cypress“. Käsuga ENTRYPOINT defineeritakse, mis käivitub konteineri loomise hetkel. Selleks on käsk, mis jooksutab automaatteste Cypressi abil. Käsuga CMD defineeritakse, et vaikimisi käivitatakse teste Chrome veebibrauseris.

4.3.2 Bitbucket Pipelines seadistus

Sidusarenduskeskkonna seadistamiseks on vaja luua põhikausta konfiguratsioonifail, milleks Bitbucket Pipelines teenuse puhul on fail nimega „bitbucket-pipelines.yml“. Selle faili täissisu on lisatud dokumendile Lisas 3.

Seadistamisel lähtutakse sellest, et tellija on avaldanud soovi tulevikus konfigureerida mitu sidusarenduse *pipeline*’i, näiteks erinevate tarkvara keskkondade jaoks või erinevate testide kogumite jooksutamise jaoks. Selle tõttu failis kasutatakse YAML *anchors* tehnoloogiat, mis võimaldab defineerida korduvkasutatavat koodi üks kord ning hiljem viidata sellele. Ühiste ressursside defineerimiseks kasutatakse märksõna `definitions`, mille alla saab paigutada korduvkasutatav kood, mis on näidatud Joonisel 15.

Igal sidusarenduse *pipeline*'il on kohustuslik defineerida samme, mida selle käigus täidetakse. Sammudel on kohustuslik väli `script`, mille alla kirjutatakse sammu käivitumisel jooksutatavad käsud. Korduvaks kasutamiseks on võimalik defineerida nii eraldiseisvaid käske kui ka terviklikke *pipeline*'i samme.

Antud projektis iga *pipeline*'i esimeseks sammuks on Dockeri tõmmise moodustamine järgmiste sammude jaoks (Joonis 15). Tõmmise edasiandmiseks järgmistele sammudele kasutatakse Bitbucket Pipelines artefakte ehk hoidlat, mida saab kasutada käskude tulemuste sammuväliseks salvestamiseks.

```
definitions:
  steps:
    - step: &build-image
        name: Build docker image
        script:
          - docker build -t e2e-tests .
          - docker save --output tmp-image.docker e2e-tests
        artifacts:
          - tmp-image.docker
        services:
          - docker
```

Joonis 15. Dockeri tõmmise moodustamise sammu skript *pipeline*'i konfiguratsioonis.

Igas *pipeline*'is on samm testide jooksutamiseks. Käsud võivad erineda parameetrite poolest, kuid iga selline samm omab samasuguseid artefakte ning *after-script*'i käske. Artefaktidesse kuuluvad kõik failid, mis tekivad testide jooksutamise tulemusena. *After-script*'i moodustavad käsud, mis jooksevad pärast sammu töö lõpetamist. Lisaks mainitule, iga sellise sammu esimeseks käsuks on Dockeri tõmmise alla laadimine ja kontrollimine. Seda on samuti võimalik defineerida korduvaks kasutamiseks (Joonis 16).

```
definitions:
  commonItems: &common docker load --input ./tmp-image.docker &&
    docker images
```

Joonis 16. Dockeri tõmmise alla laadimise ja kontrollimise skript.

Kasutades *YAML anchor*'eid, automaattestide *pipeline*'e saab defineerida nii, et neil erineb ainult üks rida (Joonis 17). Selleks reaks on Dockeri tõmmise käivitamine.

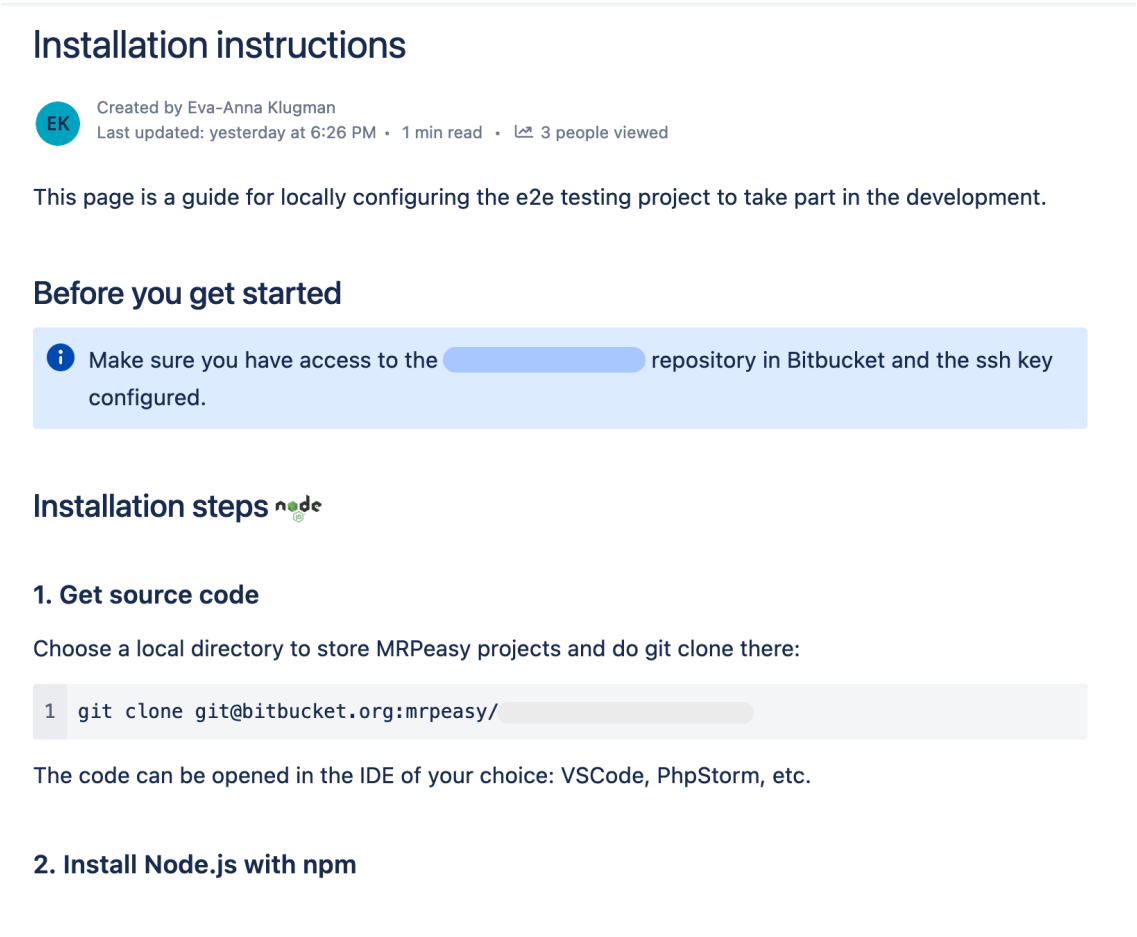
```
pipelines:
  default:
    - step: *build-image
    - step:
        <<: *e2e-run-setup
        script:
          - *common
          - docker run -i -v
$BITBUCKET_CLONE_DIR/cypress:/project-base/cypress e2e-tests
```

Joonis 17. Ühe automaattestide *pipeline*'i skript.

4.4 Projekti dokumentatsioon

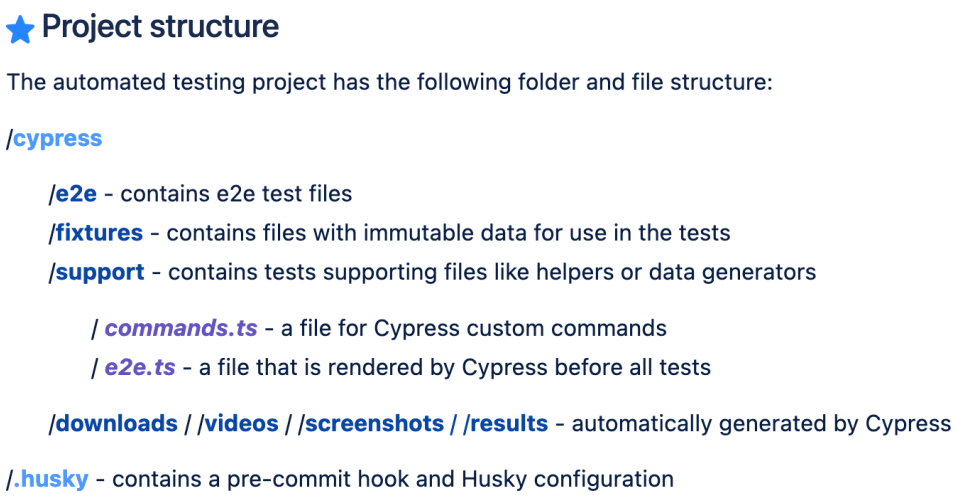
Töö käigus sai loodud inglisekeelne projekti dokumentatsioon, mis on paigutud pilvepõhisesse teenusesse Confluence ettevõttesisesesse keskkonda. Eraldi lehekülgedel on kirjeldatud installeerimisjuhend, projekti struktuur ja arendamise reeglid.

Installeerimisjuhend (Joonis 18) koosneb viiest punktist alates koodi allalaadimisest kuni Cypressi käivitamiseni. Varasemalt oli ettevõttesiseselt loodud juhend, mis käsitleb arenduskeskkonda loomist teiste ettevõtte projektide jaoks, mille tõttu puudus vajadus kirjutada lathi *git*'i ja Bitbucketi konto seadistamist. Automaattestide projekti installeerimisjuhendis on toodud sammud Node.js käituskeskkonna alla laadimiseks, projekti sõltuvuste installeerimiseks ning Cypressi abil testide jooksutamiseks.



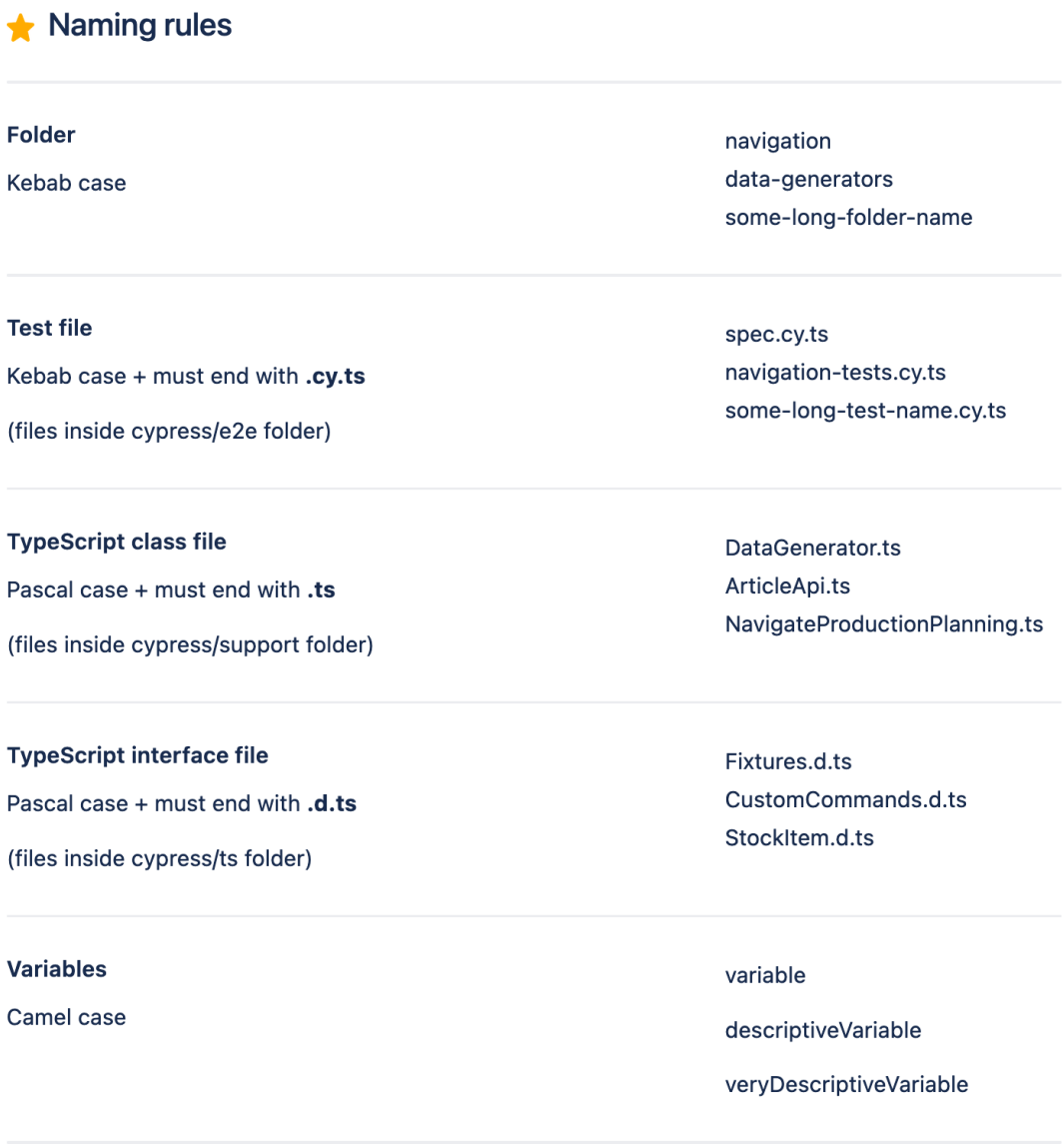
Joonis 18. Installeerimisjuhendi algus Confluence keskkonnas.

Eraldi dokumenteeriti projekti struktuur, milles on seletatud lahti põhiliste kaustade ja failide olemasolu eesmärgid (Joonis 19). Kaustade kohta on lühidalt kirjeldatud, mis failide jaoks nad on olemas. Failide kohta on seletatud nende kasutus.



Joonis 19. Projekti struktuuri kirjeldus dokumentatsioonis.

Antud töö raames on rõhutatud projekti kasutamise mugavust, kuna järeldades varasemalt mainitud Aleks Vuorjoki diplomitööst [5], automaattestide projekti kasutamismugavusel on tugev mõju arendajate produktiivsusele testide kirjutamisel. Seetõttu projektis on installeeritud automaatne koodivormindus tööriist Prettier. Tuginedes sellele, töö autor otsustas dokumenteerida nimetamise reeglid (Joonis 20), eesmärgiga kindlustada koodi kõrgetasemelist loetavust. Nimetamise reeglitesse kuuluvad kaustade, test- ja abifailide ning muutujate nimed.



Joonis 20. Nimetamise reeglid dokumentatsioonis.

4.5 Projekti kasutus

Ettevõttes on kaks kasutajaliidese automaattestide arendajat, kes on loodud projekti lõppkasutajateks. Projekti kasutatakse automaattestide ja nende jaoks vajaliku tugikoodi kirjutamiseks, ning sidusarenduskeskkonnas valminud testide jooksutamiseks.

Töö valmimise hetkeks on kirjutatud mõned regressioonitestid, mis käivituvad automaatselt seadistatud regulaarsusega sidusarenduskeskkonnas. Ettevõtte kvaliteeditagamise osakonnal on väljatöötatud plaan tarkvara kasutajaliidese katmiseks automaattestidega, kuid selle realiseerimine on jäetud tahaplaanile seoses tulevaste suurte programmiuuendustega.

Vaatamata sellele, ettevõttel on kavatsus suurendada kvaliteeditagamise osakonda, ning kasutades suuremat saadavat töötundide arvu, pühendada rohkem tähelepanu testimise automatiseerimisele.

5 Tulemuse analüüs ja järeldused

Antud töö tulemuseks on pilvepõhises repositooriumis olev projekt, mis on mõeldud kasutajaliidese automaattestide jaoks ettevõttes Mrpeasy OÜ. Järgnevates alapeatükkides hinnatakse töö tulemuse vastavust vastuvõtukriteeriumitele ning antakse ülevaade ettevõtte automaattestide arendajatega läbiviidud intervjuude vastustest.

5.1 Lõpptulemuse vastavus kriteeriumitele

Antud projektile oli rakendatud kolm peamist vastuvõtukriteeriumit, mis on toodud ka alapeatükis 3.2 „Töö vastuvõtukriteeriumid“:

1. Automaattestide loomiseks peavad arendajad omama teadmisi ainult JavaScript keelest, Cypressist ning *git*’ist.
2. Automaatteste peab olema võimalik jooksutada sidusarenduskeskkonnas.
3. Automaattestide arendajad peavad kiitma projekti kasutuskõlblikuks ning mugavaks kasutamiseks.

Valminud projektis automaattestide loomiseks ning käivitamiseks nii lokaalsest kui ka sidusarenduskeskkonnas peab arendaja suutma kirjutada automaatteste Cypress süntaksi ja TypeScripti abil. TypeScripti kasutus polnud algselt planeeritud, kuid võttes arvesse, et see on JavaScript keele laiendus, ning et keele muudatus oli kooskõlastatud projekti vastuvõtjatega, on see erinevus algsest kriteeriumist aktsepteeritav. Kuna projekti seadistamiseks kasutatakse *git*’i, siis selle peamiste käskude teadmine on arendaja jaoks vajalik, mis läheb kriteeriumiga kokku. Seega võib järeldada, et lõpptulemus vastab esimesele vastuvõtukriteeriumile.

Pilvepõhises repositooriumis on seadistatud sidusarenduskeskkond, mille jaoks on loodud vastav Dockeri tõmmis ja konfiguratsioonifail. Projektis olev „Dockerfile“ on täielik ning arendajatel ei teki vajadust seda hiljem muuta, välja arvatud baastõmmise väljavahetamise eesmärgil. Erinevalt sellest, sidusarenduskeskkonna konfiguratsioonifaili on vaja tulevikus kohandada, kui tellijal tekib soov käivitada automaatteste erinevates keskkondades või erinevates testide skoopides. Vaatamata

sellele on automaattestid integreeritud sidusarenduskeskkonnaga, mille põhjal saab järeldada, et lõpptulemus vastab teisele vastuvõtukriteeriumile.

Valminud projekti vastavust kolmandale vastuvõtukriteeriumile hinnati intervjuude abil. Kõik küsitletud automaattestide arendajad on kiitnud töö lõpptulemust kasutuskõlblikuks ja kasutamiselt mugavaks, mille põhjal saab väita, et lõpptulemus vastab kolmandale vastuvõtukriteeriumile. Läbiviidud intervjuud on detailselt kirjeldatud järgmises alapeatükis.

5.2 Arendajate tagasiside

Kokku viidi läbi neli intervjuud. Kahte automaattestide arendajat küsitleti kaks korda: kohe pärast projekti tutvustamist ja kaks kuud hiljem. Arendajatel paluti anda hinnang mitmele küsimusele viiepunktilises skaalas ning lühidalt põhjendada igat vastust. Hinnang „5“ on maksimaalne positiivne väärtus, millega sai vastata, ning „1“ on maksimaalne negatiivne väärtus. Väärtus „0“ tähendab, et vastaja ei osanud antud hetkel vastata küsimusele. Tabel 1 esitab intervjuude numbrilisi tulemusi. Tulemused on arvutatud aritmeetiliste keskmistena mõlema arendaja vastustest.

Tabel 1. Intervjuude küsimused ning nende vastustest arvutatud aritmeetilised keskmised.

Esitatud küsimus	**Algne hinnang**	**Hinnang kahe kuu pärast**
Kui lihtne on projekti installeerida lokaalsesse masinasse?	5	5
Kui lihtne on käivitada Cypressit?	3,5	4,5
Kui selge on kaustade struktuur projektis?	4,5	5
Kui selge on olemasolevate failide kasutus?	3,5	4,5
Kui lihtne on käivitada testid sidusarenduskeskkonnas?	5	5
Kui kasulikud on installeeritud raporteerimisutiliidid nagu Mochawesome ja cypress-terminal-report?	0	5
Kui kasulikud on installeeritud paketid Prettier ja Husky?	3,5	5
Mis hinnangut te annaksite projekti kasutuskõlblikusele?	5	5
Mis hinnangut te annaksite projekti kasutamise mugavusele?	5	5

Mõlemad arendajad kiitsid projekti installeerimist väga mugavaks. Installeerimise käigus kasutati kirjalikku juhendit dokumentatsioonist ning kõik arendajad said installeerimisega

hakkama. Takistusi ega probleeme ei esinenud. Cypressi käivitamise mugavuse puhul vastused erinesid. Kuna Cypressi käivitamiseks on vaja kasutada terminali, siis sellega harjuda oli lihtsam arendajal, kellel oli suurem eelnev kogemus konsooli kasutamisega. Samuti arendajad pidid tutvuma käskude `cypress run` ja `cypress open` erinevustega ning õppima kasutama neist sobiva vastavalt olukorrale. Vaatamata sellele, kahe kuu pärast mõlemad arendajad andsid kõrgema hinnangu Cypressi käivitamise lihtsusele, millest võib järeldada, et antud raskus vajas toimetulekuks ära harjumist.

Kaustade struktuur oli arendajatele arusaadav. Üks vastajatest kommenteeris, et ainukeseks küsimuseks on mõnikord see, kuhu paigutada testide tugifailid. Kuna kaust „support“ on nii laiapõhiline, siis üks ettepanekutest oli töötada välja juhendid, et määrata, mis olukordades tasub luua uusi kaustasid ning kuhu täpselt failid paigutada.

Projekti oli tekitatud mitu konfiguratsioonifaili. Arendajad olid sarnasel arvamusel, et algul oli kõikide failide eesmärkidest keeruline aru saada, kuid hiljem nende olemasoluga harjuti ära ning ei kardetud vajadusel teha vajalikke muudatusi.

Kõige rohkem oli kiidetud automaattestide integratsiooni sidusarenduskeskkonnaga. Arendajad olid ühisel arvamusel, et käivitada teste Bitbucketis on väga mugav. Kasutades selle graafilist liidest oli kergesti seadistatud automaatne regulaarne testide käivitus.

Raporteerimisutiliitide kasulikkust said arendajad hinnata ainult teises intervjuus kaks kuud hiljem, kui mõned testid olid valmis kirjutatud. Mõlemad vastajad olid arvamusel, et raportite ja logide olemasolu teeb testide jälgimist sidusarenduskeskkonnas palju kergemaks.

Prettier ja Husky pakettide puhul arendajate arvamus aja möödudes muutus kõige rohkem. Esmasel tutvumisel projektiga oli arendajate seas erimeelsus ühise koodi vormindamise küsimuses. Automaattestide arendajad omavad ettevõttes erinevat programmeerimistausta, mille tõttu on harjunud erineva koodi kirjapildiga. Vaatamata sellele, kahe kuu pärast mõlemad vastajad märkisid, et ühise koodi vormindamise kokkuleppimine algusest peale oli väga mõistlik otsus, kuna see aitab teha koodi ülevaatust efektiivsemaks. Eraldi märgiti, et sellele annavad palju juurde ka nimetamise reeglid, mis on toodud projekti dokumentatsioonis.

Mõlemad küsitletud arendajad hindasid projekti algusest peale kasutuskõlblikuks ja kasutamiselt mugavaks. See arvamus pole ajaga muutunud, millest saab järeldada, et antud töö lõpptulemus vastab ka viimasele vastuvõtukriteeriumile.

5.3 Töö edasiarenduse võimalused

Üks antud töö eesmärkidest oli luua automaattestide juurutamise juhend. Kuigi said dokumenteeritud ning põhjendatud mitmed juurutamise etapid, antud töö puhul jääb ruumi teiste praktikate uurimiseks ning nende abil veel põhjalikkuma juhendi loomiseks.

Intervjuu käigus oli mainitud, et arendajate jaoks oleks kasulik dokumentatsioon kaustade kasutamisest. Tuginedes nimetatud ettepanekule, üks võimalik töö täiendus oleks põhjalik juhend sellest, mis olukordades tasub luua uus kaust ning kuhu peab paigutama erinevaid projektiga seotuid faile. Antud töö raames polnud sellega tegeletud, kuna kirjeldatud juhendi loomiseks on vajalik selge ettekujutus kõikidest failikategooriatest, mis võivad projektis olla. Failikategooriad võivad varieeruda sõltuvalt arendusmetoodikate valikutest, mille tõttu oli tehtud otsus mitte koostada taolist juhendit.

Samuti antud töös ei käsitleta võimalust käivitada erinevad automaattestide kogumid sõltuvalt vajadusest. Sagedaseks praktikaks kvaliteedi tagamises on testide liigitamine kategooriate järgi ning nende eraldiseisev käivitamine. Kirjeldatud lähenemine võib olla kasutatud näiteks pideva regressioonitestimise läbiviimiseks. Kirjeldatud strateegia tehniliseks realiseerimiseks on tarvis teada, mis automaatteste täpselt on plaanis koostada ning mis skoopides arendajad soovivad neid käivitada. Kuna töö valmimise hetkeks ettevõtte automaattestide arendajad polnud võtnud selliseid otsuseid vastu, töö autor otsustas mitte sisse viia selleks vajalikke seadistusi. Kui tulevikus ettevõttel tekib vastav vajadus, on töö autor valmis tegema vastavat konfiguratsiooni.

6 Kokkuvõte

Tänapäeval on järjest rohkem infotehnoloogia valdkonnas tegutsevaid ettevõtteid, kes saavad aru, et ilma automatiseeritud testimiseta ei ole võimalik saavutada piisavat kvaliteedi tagamise taset, mida nõuab aina tihedamaks muutuv konkurents turul. Kuigi on loodud erinevaid automaattestimist hõlbustavaid raamistikke, võib arendamisprojekti seadistus olla vastavalt IT-osakonna soovidele keeruline ja lisateadmisi nõudev.

Antud töö eesmärgiks oli ette valmistada ja seadistada projekt automaattestide jaoks ettevõttes Mrpeasy OÜ kasutamiseks, ning samuti dokumenteerida automaattestimise juurutamise samme piisava põhjalikkusega, et töö lõpptulemust oleks võimalik kasutada juhendina taoliste projektide loomisel. Projekti valideerimise jaoks koostati nimekiri vastuvõtukriteeriumitest, millele töö lõpptulemus pidi vastama, et seda saaks hinnata ettevõtte eesmärkide täitvaks. Kõik kasutuselevõetud tehnoloogiate valikud olid põhjendatud ning lahti seletatud. Projekti üleüldist kvaliteeti ja kasutusmugavust hinnati intervjuude abil, mis viidi läbi ettevõtte automaattestide arendajate seas kaks korda kahe kuu vahega.

Antud töö tulemusena valmis pilvepõhisesse repositooriumisse paigutatud projekt automaattestide jaoks, milles on konfigureeritud automaattestimise raamistik Cypress ning integratsioon sidusarenduskeskkonnaga Dockeri tõmmise ja Bitbucket Pipelines konfiguratsioonifailide abil. Projekti on installeeritud kaks automaattestide raporteerimisutiliiti Mochawesome ja Cypress-terminal-report, mis võimaldavad arendajatel saada paremat ülevaadet ka *headless* veebilehitsejas jooksnud testide tulemustest. Lisaks mainitule, projekt sisaldab arendust hõlbustavaid tööriistu Prettier ja Husky.

Lõpptulemusena saadud projekt vastas kõikidele vastuvõtukriteeriumitele ja oli kiidetud kasutuskõlblikuks ning mugavaks kasutamiseks kõikide küsitletud automaattestide arendajate poolt. Saadud intervjuude tulemused annavad põhjust järeldada, et projekti seadistamisel tehtud otsused olid piisavalt põhjendatud ning osutusid automaattestide arendamises ja kasutamises kasulikkeks.

Töö autor hindab projekti korralikult seadistatuks ning mugavaks automaattestide arendamiseks. Suur rõhk antud töös oli projektist kergel arusaamisel ja koodi loetavusel. Selle eesmärgi saavutamise üheks näiteks on sidusarenduskeskkonna konfiguratsioonifail, mis on koostatud jälgides, et *pipeline*'ide lisamisel ei tuleks koodi korduseid. Nimetatud intervjuud projekti lõppkasutajate ehk automaattestide arendajatega kinnitavad, et antud eesmärk oli täidetud.

Kasutatud kirjandus

[1] M. T. Taky, “Automated Testing with Cypress,” Vaasan ammattikorkeakoulu VAMK, University of Applied Sciences, Vaasa, 2021.

[2] T. Pae, „Läbivtestide testimisvahendi valik veebirakenduse näitel,“ Tallinna Tehnikaülikool, Tallinn, 2022.

[3] E. Pelivani, A. Besimi ja B. Cico, „A Comparative Study of UI Testing Framework,“ %1 *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, Budva, Montenegro, 2022.

[4] E. Kinsbruner and G. Bahmutov, A Frontend Web Developer's Guide to Testing, Packt Publishing, 2022, p. 304.

[5] A. Vuorjoki, „A developer-friendly automated web GUI test strategy,“ Aalto University, Espoo, 2021.

[6] Mrpeasy OÜ, „Mrpeasy User Manual: Overview of MRPeasy,“ [Võrgumaterjal]. Available: https://www.mrpeasy.com/resources/user-manual/#overview. [Kasutatud 19 aprill 2023].

[7] Cypress, „TypeScript | Cypress Documentation,“ Cypress, 2023. [Võrgumaterjal]. Available: https://docs.cypress.io/guides/tooling/typescript-support. [Kasutatud 26 aprill 2023].

[8] Microsoft, „TypeScript: JavaScript With Syntax For Types,“ Microsoft, 2023. [Võrgumaterjal]. Available: https://www.typescriptlang.org/. [Kasutatud 26 aprill 2023].

[9] Devographics, „State of JavaScript 2022: Opinions,“ 2022. [Võrgumaterjal]. Available: https://2022.stateofjs.com/en-US/opinions/. [Kasutatud 26 aprill 2023].

[10] J. Bogner ja M. Merkel, „To Type or Not to Type? A Systematic Comparison of the Software Quality of JavaScript and TypeScript Applications on GitHub,“ %1 *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, Pittsburgh, PA, USA, 2022.

[11] Atlassian Cloud, „Bitbucket | Git solution for teams using Jira,“ 2023. [Võrgumaterjal]. Available: https://bitbucket.org/product/. [Kasutatud 26 aprill 2023].

[12] Atlassian Cloud, „Bitbucket Pipelines - Continuous Delivery | Bitbucket,“ Atlassian Cloud, 2023. [Võrgumaterjal]. Available: https://bitbucket.org/product/features/pipelines. [Kasutatud 26 aprill 2023].

[13] Docker Inc., „Docker overview | Docker Documentation,“ Docker Inc., 2023. [Võrgumaterjal]. Available: https://docs.docker.com/get-started/overview/. [Kasutatud 27 aprill 2023].

[14] Cypress, „Docker | Cypress Documentation,“ Cypress, [Võrgumaterjal]. Available: https://docs.cypress.io/examples/docker. [Kasutatud 26 aprill 2023].

[15] A. Gruber, „GitHub - adamgruber/mochawesome: A Gorgeous HTML/CSS Reporter for Mocha.js,“ GitHub, [Võrgumaterjal]. Available: https://github.com/adamgruber/mochawesome#readme. [Kasutatud 28 aprill 2023].

[16] F. Zoltan, „cypress-terminal-report - npm,“ [Võrgumaterjal]. Available: https://www.npmjs.com/package/cypress-terminal-report. [Kasutatud 28 aprill 2023].

[17] Prettier, „What is Prettier? · Prettier,“ Prettier, [Võrgumaterjal]. Available: https://prettier.io/docs/en/index.html. [Kasutatud 26 aprill 2023].

[18] Devographics, „The State of JS 2021: Other Tools,“ Devographics, 2021. [Võrgumaterjal]. Available: https://2021.stateofjs.com/en-US/other-tools/#utilities. [Kasutatud 26 aprill 2023].

[19] typicode, „GitHub - typicode/husky: Git hooks made easy,“ GitHub Inc., 2023. [Võrgumaterjal]. Available: https://github.com/typicode/husky. [Kasutatud 26 aprill 2023].

[20] typicode, „Husky - Git hooks,“ 2023. [Võrgumaterjal]. Available: https://typicode.github.io/husky/#/. [Kasutatud 26 aprill 2023].

[21] Microsoft, „TypeScript: Documentation - The Basics,“ Microsoft, [Võrgumaterjal]. Available: https://www.typescriptlang.org/docs/handbook/2/basic-types.html#tsc-the-typescript-compiler. [Kasutatud 26 aprill 2023].

[22] Git, „Git - git-clone Documentation,“ Git, [Võrgumaterjal]. Available: https://git-scm.com/docs/git-clone. [Kasutatud 29 aprill 2023].

[23] Atlassian, „Set up personal SSH keys on macOS | Bitbucket Cloud | Atlassian Support,“ Atlassian, [Võrgumaterjal]. Available: https://support.atlassian.com/bitbucket-cloud/docs/set-up-personal-ssh-keys-on-macos/. [Kasutatud 29 aprill 2023].

[24] Git, „Git - gitignore Documentation,“ Git, [Võrgumaterjal]. Available: https://git-scm.com/docs/gitignore. [Kasutatud 29 aprill 2023].

[25] GitHub, „npm-init | npm Docs,“ GitHub, [Võrgumaterjal]. Available: https://docs.npmjs.com/cli/v9/commands/npm-init. [Kasutatud 29 aprill 2023].

[26] A. Gruber, „GitHub - adamgruber/mochawesome-report-generator: Standalone mochawesome report generator. Just add test data.,“ GitHub, [Võrgumaterjal]. Available: https://github.com/adamgruber/mochawesome-report-generator. [Kasutatud 30 aprill 2023].

[27] A. Telesh, „GitHub - Antontelesh/mochawesome-merge: Merge several Mochawesome JSON reports,“ GitHub, [Võrgumaterjal]. Available: https://github.com/Antontelesh/mochawesome-merge. [Kasutatud 30 aprill 2023].

[28] Cypress, „Reporters | Cypress Documentation,“ Cypress, [Võrgumaterjal]. Available: https://docs.cypress.io/guides/tooling/reporters#Spec-to-STDOUT-produce-a-combined-Mochawesome-JSON-file. [Kasutatud 30 aprill 2023].

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

Mina, Eva-Anna Klugman

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Kasutajaliidese automaattestimise raamistiku juurutamine infotehnoloogia idufirma 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.

16.05.2023

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

Lisa 2 – Projekti faili „package.json“ sisu

```
{
	"author": "Eva-Anna Klugman <evklug@taltech.ee>",
	"scripts": {
		"prepare": "husky install"
	},
	"lint-staged": {
		"**/*": [
			"bash -c \"tsc -p ./cypress/tsconfig.json --noEmit --
pretty\"",
			"prettier --write ./cypress --ignore-unknown"
		]
	},
	"devDependencies": {
		"@types/mochawesome": "^6.2.1",
		"cypress": "^12.11.0",
		"cypress-terminal-report": "^5.1.1",
		"husky": "^8.0.0",
		"lint-staged": "^13.2.2",
		"mochawesome": "^7.1.3",
		"mochawesome-merge": "^4.3.0",
		"mochawesome-report-generator": "^6.2.0",
		"prettier": "^2.8.8",
		"typescript": "^5.0.4"
	}
}
```

Lisa 3 – Projekti faili „bitbucket-pipelines.yml“ sisu

```
definitions:
  commonItems: &common docker load --input ./tmp-image.docker &&
    docker images
  steps:
    - step: &build-image
        name: Build docker image
        script:
          - docker build -t e2e-tests .
          - docker save --output tmp-image.docker e2e-tests
        artifacts:
          - tmp-image.docker
        services:
          - docker
    - step: &e2e-run-setup
        name: "Run e2e tests"
        image: node:16
        size: 2x
        script:
          - *common
        services:
          - docker
        caches:
          - docker
          - node
        artifacts:
          - cypress/screenshots/**
          - cypress/videos/**
          - cypress/downloads/**
          - cypress/results/**
        after-script:
          - npm install --save-dev mocha mochawesome mochawesome-merge
mochawesome-report-generator
          - npx mochawesome-merge "cypress/results/*.json" > mochawesome.json
          - npx marge --reportDir cypress/results/common-report
mochawesome.json
pipelines:
  default:
    - step: *build-image
    - step:
        <<: *e2e-run-setup
        script:
          - *common
          - docker run -i -v $BITBUCKET_CLONE_DIR/cypress:/project-
base/cypress e2e-tests
```