TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Ivo Mäeoja 192859IADB

Hajusrakenduse automaatse ehitus- ja paigaldusprotsessi arendus

Bakalaureusetöö

Juhendaja: Kristiina Hakk
PhD

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: Ivo Mäeoja

15.05.2023

Annotatsioon

Käesoleva töö raames uuritakse, kuidas parandada tiimis kasutusel olevat rakenduse lähtekoodi kompileerimise ja rakenduse paigalduse automatiseerimisprotsessi. Senisel protsessil on mitmeid puudujääke, mis mõjuvad negatiivselt tarkvara arendusprotsessile.

Analüüsitakse ja hinnatakse erinevaid võimalike automatiseerimisplatvorme ning komponente, mis võiksid senist olukorda parandada. Käsitledes erinevaid platvorme ja komponente kirjeldatakse ka nende tööviise ning valitakse arendatava hajussüsteemi ning selle komponentide jaoks sobivaimad.

Analüüsi käigus loodud prototüübi täitmiskiirust võrreldakse senise konveieri omaga ning peale hilisemate täienduste lisamist on võimalik konveier tiimis kasutusele võtta.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 44 leheküljel, 6 peatükki, 26 joonist, 5 tabelit.

Abstract

Improving the Automation of Building and
Deploying Distributed Systems

In the scope of this work, it's investigated how to improve currently in use software application build and deploy automation process. Current process has many deficiencies that have negative effect on the software development process.

Potential automation systems and components that could improve current situation are analysed and evaluated. Addition to examining different systems and components, how they work will be also described with the intent to select most suitable for the distributed system and its components currently in development.

Running speed of prototype pipeline created using accelerated systems analysis will be compared against the current pipeline and presumably can be put to use after applying additions that will improve the pipeline.

The thesis is in Estonian and contains 44 pages of text, 6 chapters, 26 figures, 5 tables.

Lühendite ja mõistete sõnastik

.NET	.NET raamistik, mis on Microsofti poolt loodud erinevate rakenduste arendamiseks
AppCmd	Tööriist IIS veebiserveri haldamiseks
Beetaversioon	Tarkvara või rakenduse proovi- ja katsetamisversioon
Eesrakendus	Rakenduse osa mis tegeleb veebilehe kuvamisega
Git	Lähtekoodi versioonide haldamise tööriist
HTTP	*Hypertext Transfer Protocol*, klient-server protokoll ressursside pärimiseks arvutivõrgus
HTTPS	*Hypertext Transfer Protocol Secure*, laiendus HTTP protokollile mis kasutab seal liikuvate andmete krüpteerimist
IIS	*Internet Information Services*, Windows Server operatsioonisüsteemis olev veebiserver
Konteineriseerimine	Isoleeritud rakendus vajalike sõltuvustega, ilma operatsioonisüsteemita
Konveierfail	Järjestatud programmikäskude fail mis on täitmiseks ette nähtud
Koodirepositoorium	Koodihoidla, mis on hallatav versioonihaldustarkvara abil
Kotlin	Ettevõtte JetBrains poolt loodud programmeerimiskeel
Kvaliteeditagamine	*Quality assurance*, programmikoodi nõuetele vastavuse kontroll
Käitus	*Operations*, rakenduse käigus hoidmine
NuGet	.NET teekide haldustööriist
Operatiivmälu	*Random Access Memory* (*RAM*), suvapöördusmälu arvutis või serveris andmete kirjutamiseks ja lugemiseks
Pidevintegratsioon	*Continuous Integration* (*CI*), tarkvaraarenduse tava kus arendajad lisavad koodirepositooriumisse koodi vähemalt üks kord päevas
PowerShell	Microsofti skriptimiskeel ja käsureaprogramm
Programmifail	(*Executable*) Täitmiskõlblik fail mis paigaldab arvutisse programmi tööks vajalikud failid ja muudatused
Rakendusliides	*Application Programming Interface* (*API*), liides HTTP-päringute abil rakendusele info saatmiseks või vastuvõtmiseks
Saalefail	*Pagefile*, võetakse kasutusele arvutis või serveris kui suvapöördusmälu on täies mahus kasutusel

SCM	*Source Code Management*, lähtekoodi haldamise süsteem selle versioonide abil
SCSS	*Sassy Cascading Style Sheets*, stiilileht veebilehe visuaali muutmiseks
SFC	*Vue Single-File Component*, Vue raamistiku failiformaat mis võimaldab vajalike komponente ühes failis hoida
SQL	*Structured Query Language*, struktureeritud päringukeel andmebaaside käsitlemiseks
Tagarakendus	Rakenduse osa mis tegeleb rakenduse äriloogika ja päringute vastuvõtmise ning vastuste koostamisega
Teams	Microsofti koostöö ja koosoleku rakendus
Tehis	*Artifact*, Lähtekoodi kompileerimisprotsessi tulem ehk täitmisprogramm
Veebikonks	*Webhook*, automaatsed teavitused mida rakendused saadavad teineteisele kui mingi sündmus toimub
Vue	Javascripti raamistik

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tarkvaraarendajana on rakenduse programmikoodi tehtud muudatustele oluline saada kiiret tagasisidet. Vead ning mestimisprobleemid, mis on tekkinud teiste tarkvaraarendajate poolt lisatud koodiga, on kiirem ja odavam lahendada kohe pärast nende avastamist. Probleemide kiire avastamine nõuab aga teatud automaatsete etappide kasutuselevõttu, millele tuginevad ka pidevintegratsiooni praktikad. Antud töös käsitletakse probleeme, mis on tekkinud olemasoleva automatiseerimisprotsessi ja selle komponentide kauaaegsest tähelepanuta jätmisest ning nende mõjust protsessile enesele ning tagasiside kiirusele.

Eesmärk on võtta kasutusele asutuse poolt toetatud pidevintegratsiooni platvorm ning sellega liidestatud serverid rakenduse kompileerimiseks ja paigaldamiseks. Vabaneda või märgatavalt vähendada sõltuvust programmeerimiskeeles F# kirjutatud faili abil tehtavatest toimingutest. Muuta pidevintegratsiooni protsess paremini arusaadavamaks, hallatavamaks ning kiiremaks ja luua eeldused edasisteks täiendusteks.

Puudulik või iganenud automatiseerimissüsteem aeglustab tagasiside saamise kiirust. Probleemide kiireks lahendamiseks on vaja tehtud muudatustele võimalikult kiiresti tagasisidet saada. Vea parandamine arendustsükli alguses on odavam ja kiirem ning hästi toimiv automatiseerimissüsteem võimaldab neid varem leida.

Lõputöö on jaotatud kuueks peatükiks. Töö teises peatükis kirjeldatakse tekkinud probleeme ja eesmärke ning pidevintegratsiooni praktikaid, mida võetakse eeskujuks kiirendamaks arendusprotsessi. Kolmandas peatükis analüüsitakse automatiseerimisplatvormide ja konveierite ning erinevate tööriistade nõuetele vastavust. Valitakse tööriistad mis sobivad kõige paremini arendatava hajusrakendusega. Neljandas peatükis kirjeldatakse töö käiku ning konveieri täitmist uuel automatiseerimisplatvormil. Viiendas peatükis vaadeldakse uuel automatiseerimisplatvormil käivitunud konveieri töö käiku ning kirjeldatakse edasiarenduse võimalusi.

2 Probleemi taust ja projekti eesmärk

Tarkvaraarenduse elutsükkel koosneb mitmest etapist, sealhulgas lisaks tarkvara arendamise etapile tsükli hilisemas faasis ka lähtekoodi kompileerimise ja rakenduse paigaldamise etapp. [1]

Traditsiooniline arendusprotsess algab nõuete kirjeldamisest kliendi poolt, peale mida jagunevad ülesanded kolme osapoole vahel. Esimesena alustavad projekti kallal tööd tarkvaraarendajad. Kui nõue on realiseeritud, siis tehtud muudatused lisatakse koodihoidla põhiharusse. Peale seda algab tarkvara testimise ja kvaliteedi tagamise protsess, kus käivitatakse erinevaid teste veendumaks, et tarkvara toimib korrektselt ning see vastab spetsifitseeritud nõuetele. Testimise protsess võib ka tähendada rakendusest tehiste loomist ning paigaldust testkeskkonda manuaalsete testide jaoks. Kõikidest vigadest mis leitakse, teavitatakse tarkvaraarendajaid. Kolmanda ning viimase sammuna edastatakse rakenduse kood käituse eest vastutavatele, kes tegelevad rakenduse paigaldusega ning jälgimisega. Leitud vigade korral pöördutakse tarkvaraarendajate poole. Traditsiooniline arendusprotsess on olematu või puuduliku automaatika tõttu ajamahukas ning tarkvaraarendajad ei saa piisavalt kiiresti tagasisidet tekkinud vigade kohta. [2], [3]

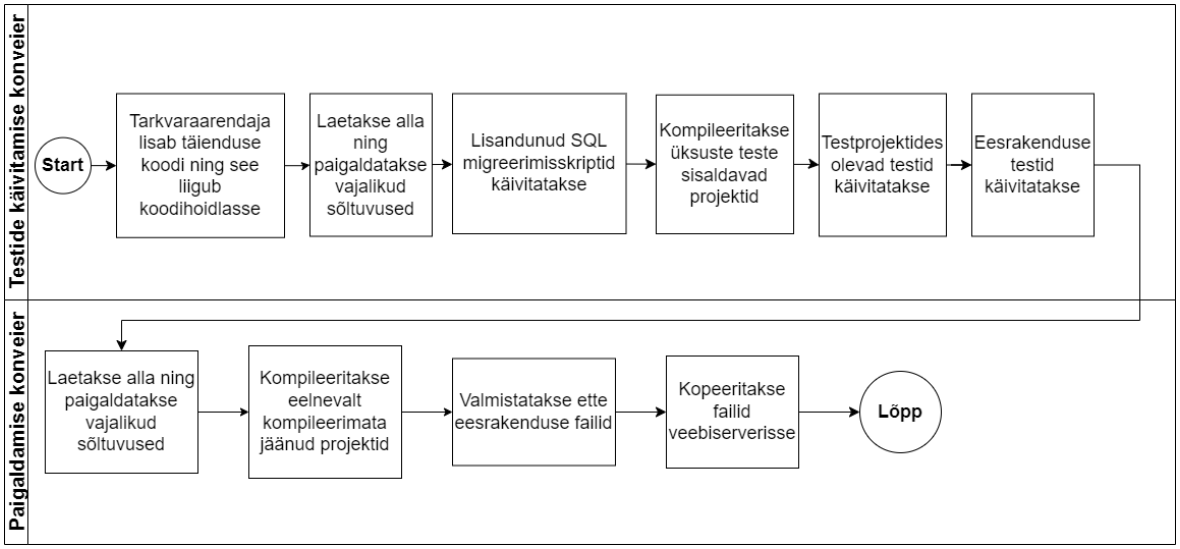
Traditsioonilises tarkvaraarenduse elutsüklis on etapid ajakulukad. Eelneva protsessi automatiseerimine võimaldaks tarkvaraarendajatel saada kiiremat tagasisidet tehtud vigade kohta ning rakendust kiiremini testimiseks paigaldada. Lisaks, eemaldades tarkvaraarendajate poolt manuaalselt tehtavad sammud, jääb neil rohkem aega tegelemiseks tarkvara arendamisega. Protsessi automatiseerimiseks on vaja eelnevad sammud kirjeldada konveieris, mis on sisult skript erinevate sammudega, mis täidavad tarkvaraarenduse elutsükli etappe.

2.1 Taust

Autor töötab asutuses kus tarkvara arendustsükli, rakenduse kompileerimise ja tehiste loomise kiirendamiseks on kasutatud automaatikat, kuid selle rakendamine on erinevatel

põhjusel jäänud poolikuks ning hetkelahendus ei ole erinevate probleemide tõttu jätkusuutlik. Autor kuulub tiimi, kus koos kolme teise tarkvaraarendajaga luuakse hajussüsteemi millel on .NET (*.NET Framework*) tagarakendus, Vue 2-s [4] tehtud eesrakendus ning Microsofti SQL (*Structured Query Language*) andmebaas. Süsteemi arendusel on pidevas muutumises nii rakenduse kood kui ka andmebaasi struktuur.

Tiimi arendajad lisavad täiendusi koodirepositooriumi peaharusse peale mida käivitub automaatselt TeamCity platvormil rakenduse kompileerimise konveier. Peale vajalike sõltuvuste leidmist ning paigaldamist testitakse projekti üksuste testide käivitamise abil lisatud muudatusi ning kui kõik on edukas, siis kuvatakse TeamCity platvormil sellekohane teavitus. Kui see etapp oli edukas, siis on käsitsi võimalik käivitada rakenduse paigaldamise protsess, mis hõlmab endas vajalike sõltuvuste paigaldamist, programmikoodi kompileerimist ning täitmisprogrammi paigaldamist (vt. joonis 1).



Joonis 1. Kompileerimis- ja paigaldusetapid praegu.

Esmane täitmisprogrammi paigaldus toimub arenduskeskkonda, mida uuendatakse kohe peale muudatuse tegemist peaharus juhul kui kompileerimise- ja paigalduskonveier ilma vigadeta lõpetavad. Rakenduse arenduskeskkonda paigaldamine on oluline testijate jaoks, kes teevad koostööd tarkvaraarendajatega loodavate muudatuste testimiseks. Tiimi tarkvaraarendajad vastutavad kompileeritud täitmisprogrammi paigalduse eest arendus- ja arendusjärgsesse keskkonda. Kokku on erinevaid keskkondi viis. Test-, koolitus- ja toodangukeskkonda paigaldamise eest vastutavad süsteemiadministraatorid. Paigaldusel tekkinud probleeme lahendatakse koostöös tiimi liikmetega, kuid peaasjalikult tarkvaraarendajatega kes on paremini kursis tehtud muudatustega.

Kui on vaja mõne keskkonna jaoks paigaldusvalmis failid luua, siis valitakse täiendused koodirepositooriumi peaharust ning mestitakse vastava keskkonna jaoks mõeldud koodirepositooriumi harus oleva koodiga. Hiljem kompileeritakse harus olev lähtekood, luuakse täitmisprogramm kasutades TeamCity platvormi ning paigaldatakse testimiseks vajalikku keskkonda.

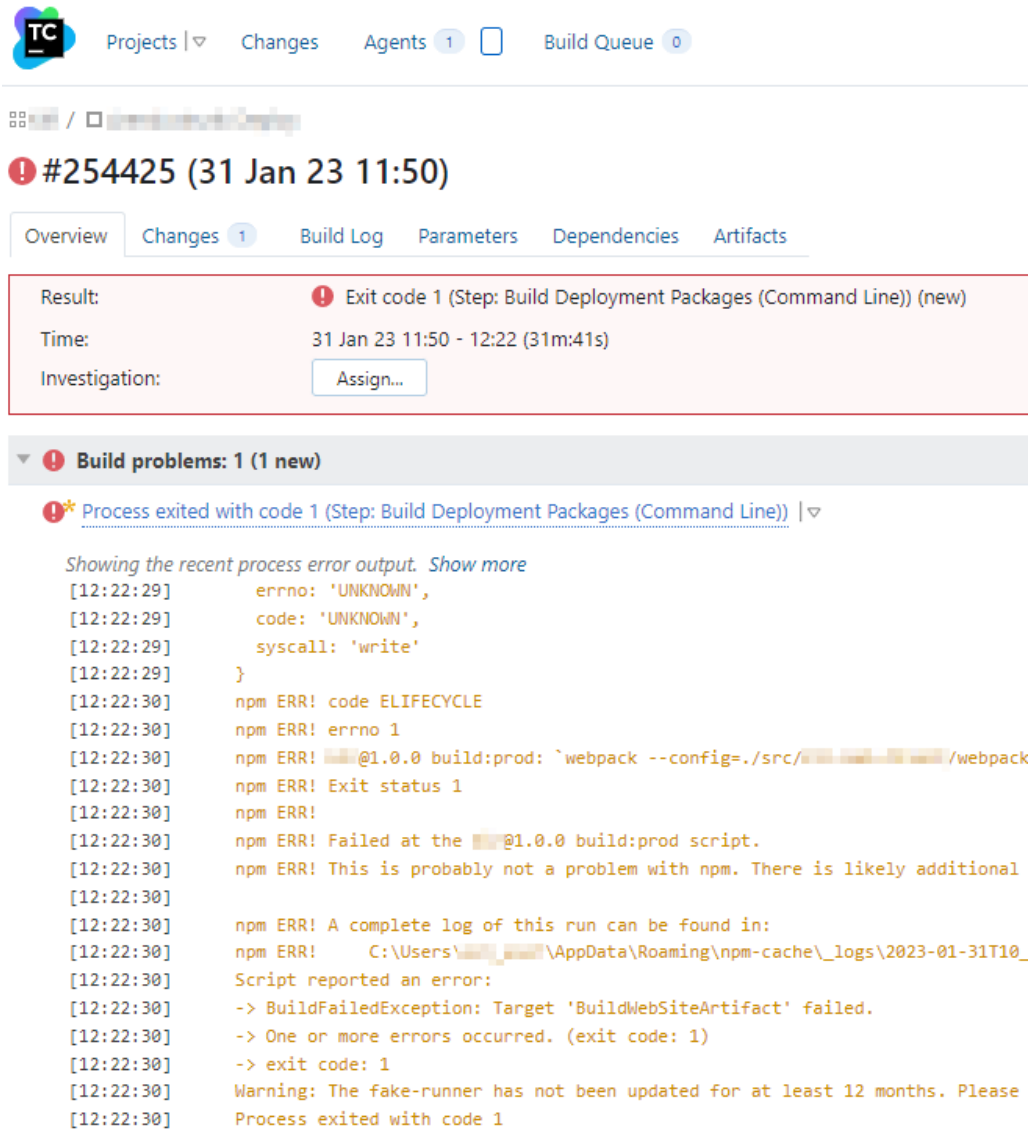
2.2 Lahendamist vajavad probleemid

Senine automatiseerimine toetub automatiseerimisplatvormile TeamCity. Hetkel on antud platvormist kasutuses versioon 2019.1.1 mis avaldati 24.06.2019. Praegu on TeamCity platvormist uusim versioon 2022.12.2, mis on avaldatud 03.02.2023. [5] Seega on hetkel kasutusel olev platvorm kõige uuemast võimalikust versioonist maas enam kui kolme aastat. Kogu selle perioodi jooksul pole platvormile uuendust tehtud, mis tähendab, et uuendamata jätmine pole olnud juhuslik, vaid sellega ei tegeleta. Automatiseerimisplatvorm töötab Windows Server 2008 R2 operatsioonisüsteemil, mis algselt väljastati 22.10.2009 ja mille kõikvõimalikud toed tootja poolt on lõppenud. [6] Serveri töötab ainult asutuse sisevõrgus (vt. joonis 2).

Joonis 2. TeamCity platvorm.

TeamCity platvorm peab logi erinevate tegevuste kohta sh kasutajate ja rollide muutmise ning konfiguratsioonisätete kustutamise ja muutmise üle. [7] jooniselt 2 näeme, et viimane sündmus, mis vajas auditeerimist juhtus 49 kuud tagasi. Sellest võib järeldada, et platvorm võidi kasutusele võtta vähemalt neli aastat tagasi.

Rakenduse lähtekoodi kompileerimise protsess lõppeb sageli veateatega (vt. joonis 3) mis tähendab, et protsess tuleb uuesti manuaalselt käivitada.



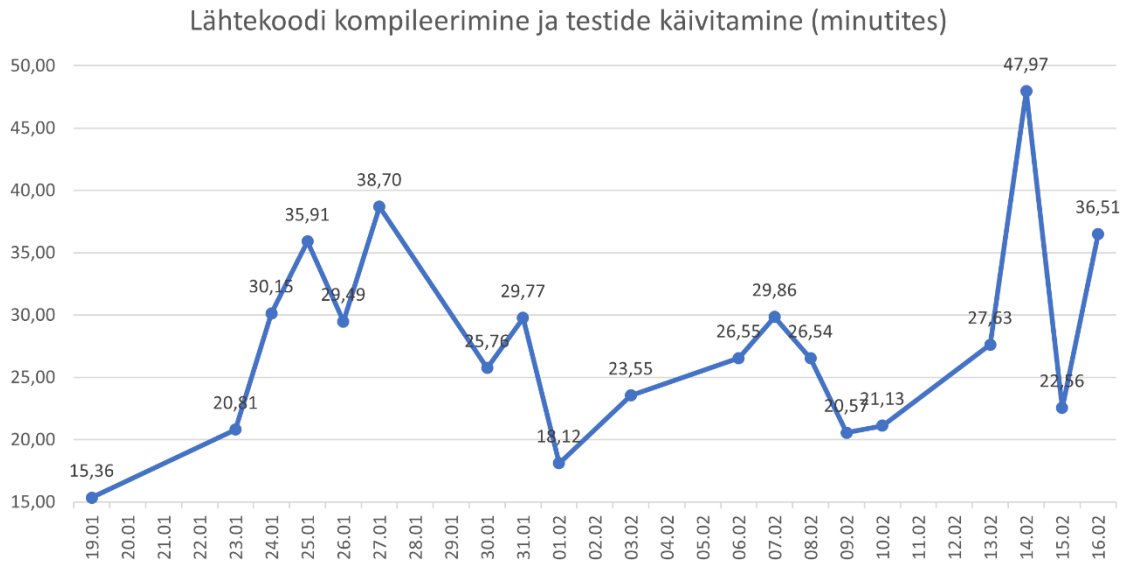
Joonis 3. Kompileerimisprotsessi lõpp veateate läbi.

Veateateid uurides selgub, et mõningad vead on põhjustatud serveri vähesest operatiivmälu hulgast. Ei ole teada, miks ei kasutata mälu puudumisel automaatselt saalefaili ja kas veateade operatiivmälu vähesuse kohta vastab tõele.

Samuti on server, kus TeamCity automatiseerimisplatvorm töötab, vaikimisi ainuke agent, mis teostab nii rakenduse lähtekoodi kompileerimist kui ka loob kõikide paigalduskeskkondade jaoks vajalikud tehised [8]. See tähendab lisakoormust, kuna konfigureerimiseks on vaja töös hoida veebilehte ja lisaks tegeleda lähtekoodi kompileerimisega.

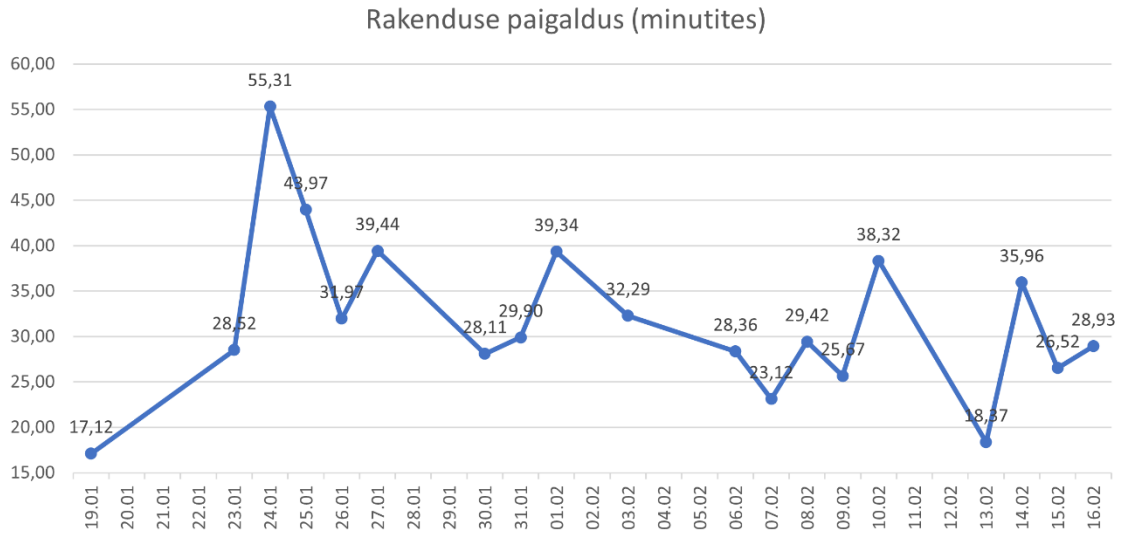
Olemasolevatele probleemidele lisaks on senised rakenduse lähtekoodi kompileerimise ja paigaldamise etapid juba niigi ajakulukad. Halvimal juhul võib ühe koodimuudatuse arenduskeskkonda paigaldamine võtta kuni tund. Arenduskeskkonda paigaldusele eelneb

kaks etappi, millest esimene on rakenduse testprojektide lähtekoodi kompileerimine ning testide käivitamine mis annab esmase kiire tagasiside. Kõik see võtab keskmiselt 27 minutit aega (vt. joonis 4).



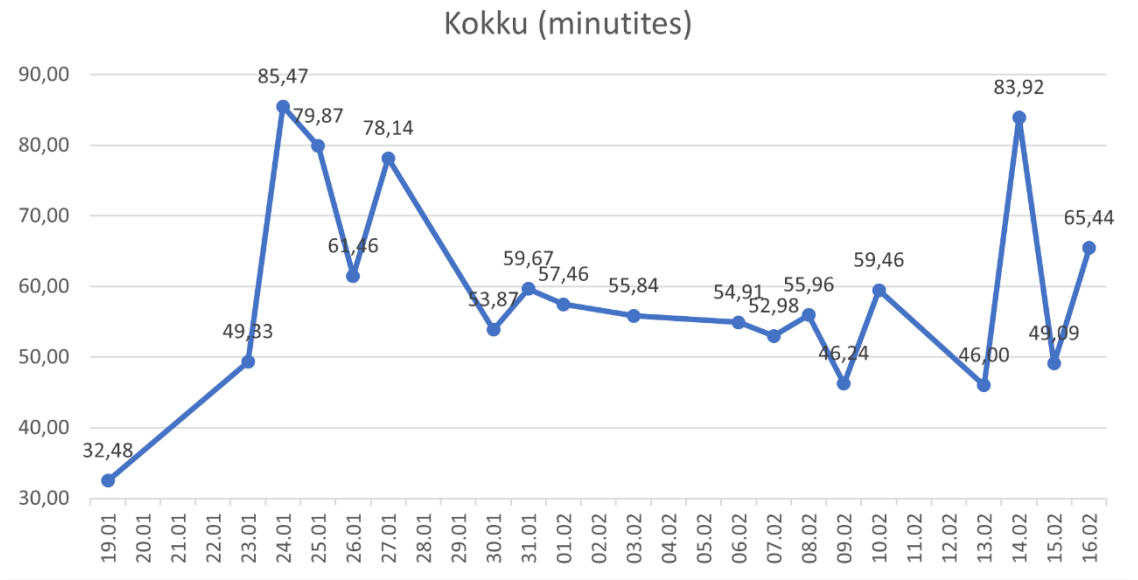
Joonis 4. Rakenduse lähtekoodi kompileerimise ajakulu.

Kui rakenduse lähtekoodi kompileerimisprotsess õnnestub, siis järgmine samm on rakenduse paigaldus. Rakendus paigaldatakse Windows Server 2019 veebiserverisse, mis peale edukat paigaldusprotsessi selle kasutajatele kättesaadavaks muudab. Paigaldusprotsess on analoogselt kompileerimisele samuti väga ajakulukas ning võtab keskmiselt 31 minutit aega (vt. joonis 5).



Joonis 5. Rakenduse paigalduse ajakulu.

Mõlemas ajahinnangu andmisel on arvestatud ka mõningast ootel oleku aega, mis on üldjuhul kaks kuni kümme minutit. Kahe etapi koguaeg on keskmiselt 59 minutit. See tähendab, et peale koodimuudatuse salvestamist koodihoidlasse, läheb vähemalt tund mööda enne kui antud muudatust arendusserveris testida saab (vt. joonis 6).



Joonis 6. Rakenduse lähtekoodi kompileerimise ja paigalduse kumulatiivne ajakulu.

Kogu ajakulu vaadates on näha, et hetkel ei ole võimalik lisatud täienduste kohta kiirelt tagasisidet saada.

Täiendavaks probleemiks on hetkel ka praeguse konveieri keerukus. Nimelt on TeamCity platvormil konveieri etapid konfigureeritud ühes asukohas, konveieris kasutatavad muutujate väärtused defineeritud teises menüüs. Konveier käivitab programmeerimiskeeles F# loodud 1124 realise faili, mis täiendavalt viitab käsurea failile, mis käivitab PowerShelli failis olevad käsud. Oleks hallatav ja ülevaatlikum kui kõik käsud ja muutujad oleks ühes asukohas. TeamCity võimaldab luua konveierfaile ning hoida neid koodihoidlas keeles Kotlin kirjutatud failidena [9], mis leevendab eelnevat probleemi.

Probleemseks on osutunud ka programmeerimiskeeles F# loodud fail, kuna kasutatud keelest ei omata piisavalt põhjalike teadmisi. Antud failist on võimalik sõltuvust vähendada tõstes sealt ükshaaval välja sammud, mida fail täidab ning asendada need käsurea käskudega koos vajalike võtmetega ning tõsta konveierfaili. Teine võimalus oleks kasutada mõnda teist tarkvarateeki mille programmeerimiskeelega ollakse rohkem tuttavam.

Ka on näha teekide uuendamata jätmise probleemi antud failis. Erinevad teegid mängivad rolli F# faili töös, kuna see erineb projektis kasutatavast keelest ning samuti tegeleb fail ka andmebaasi migratsioonidega, mille teek on samamoodi uuendamata.

Järgnevalt antakse ülevaade kasutuselolevatest teekidest, nende versioonidest ning hetkel kõige uuemast stabiilsest versioonist ja kuupäevast (vt. tabel 1).

Tabel 1. Kompileerimis- ja paigaldusprotsessi teegid.

Teegi nimi	**Teegi versioon**	**Hetke-versiooni avaldamise kuupäev**	**Viimane versioon**	**Teegi avaldamise kuupäev**	**Teegi kirjeldus**
FSharp.Core	5.0.1	10.02.2021	7.0.200	14.02.2023	F# kompilaator, baasteek ja tööriistad
FSharp.Data.TypeProviders	5.0.0.6	11.06.2018	6.0.1	20.11.2021	F# pärandtüübid
fake-cli	5.20.4-alpha.1642	23.10.2020	6.0.0	21.02.2023	Teek F# tööriista käsurea käskude toetamiseks

Eelnevalt väljatoodud probleemid kuuluvad kolme kategooriasse, milleks on vananenud riistvara, aeglane kompileerimis- ja paigaldusprotsess ning iganenud tarkvarateegid, millest sõltub lähtekoodi kompileerimine ja paigaldus. Aeglase kompileerimis- ja paigaldusprotsessi ning võimalike ebaõnnestumiste põhjal võib väita, et protsess ise kiiremaks ei lähe vaid võib aja jooksul halveneda. Täiendav lisakeerukus tuleneb nii F# keeles loodud failist, millest puudub põhjalik arusaam kui ka kompileerimis- ja paigalduskonveieri komplitseeritusest ning raskesti mõistetavusest.

2.3 Eesmärgid

Lõputöö eesmärk on asutuses kasutusel olevat hetkelahendust kaasajastada ning leida hallatav lahendus automatiseerimisele, mis toetaks ka tulevaid täiendusi nii rakendusele kui ka automaatikale.

Eesmärgid on kiirendada lähtekoodi kompileerimis- ja paigaldusprotsessi parandamaks tagasiside kiirust, vähendada paigaldusprotsessi keerukust ning hoida kõik vajalikud sõltuvused ajakohased. Kiirem protsess aitab rakendust kiiremini arenduskeskkonda paigaldada, et testijad saaksid paralleelselt uut funktsionaalsust kontrollida kuniks arendustega lõpuni jõutakse. Probleemivaba rakenduse lähtekoodi kompileerimine ja tehiste loomine vähendab ka riist- ja inimressurssi kasutust.

Lisaks on eesmärk praegune paigaldusprotsess muuta selgemaks, paremini mõistetavaks ning hallatavamaks. Paigaldusprotsessi keerukus tuleneb praeguse konveieri konfigureerimiseks vajalike parameetrite ja tegevuste paiknemisest mitmes erinevas kohas. Kasutuses oleval automatiseerimisplatvormil on erinevad menüüd paigalduseks vajalike andmete hoidmiseks. Loetavust ning hallatavust saab parandada tuues konveieri andmed kokku ühte konveierfaili, mis on paremini loetav ning mida saab hoida koos rakenduse koodiga repositooriumis. See tagab konveierfaili erinevate versioonide talletamise võimaluse, nägemaks kuidas konveierfail on ajas muutunud. Ning kuna kõik info on ühes failis on ka seda lihtsam hoomata ning muudatusi teha.

Eesmärk on ka tagada kompileerimiseks vajalike komponentide ajakohasena hoidmine.

2.4 Pidevintegratsioon

Pidevintegratsioon, mis on tuntud ka lühendina CI on tarkvara arenduse tava, mille järgimine võimaldab vigu kiiremini leida. Vigu on võimalik leida kiiremini tänu automatiseeritud protsessidele, mis kompileerivad lähtekoodi ja käivitavad testid. Selle eelduseks on programmikoodi täienduste lisamine koodirepositooriumi peaharusse vähemalt kord päevas, mille tulemusel on muudatused väiksemad ning tekkinud mestimisprobleeme on lihtsam lahendada. Koodirepositoorium peab olema ühtne asukoht tarkvaraprojektiga seotud failidele [10].

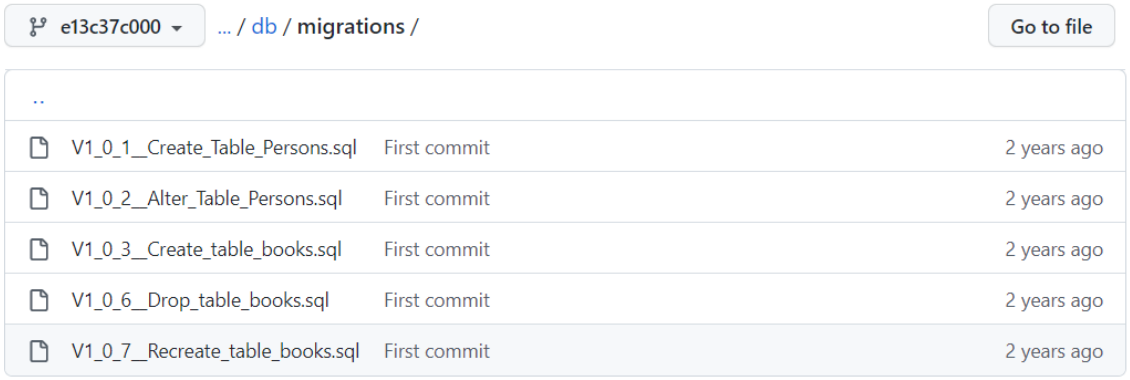
Eelnevalt välja toodud lahendatavad probleemid hõlmavad endas aeglast lähtekoodi kompileerimist ja paigaldusprotsessi mis läheb vastuollu pidevintegratsiooni praktikatega ning lisaks on protsess liialt keerukas. Praeguse automatiseerimisprotsessi parandamine ja lihtsustamine on hea alus pidevintegratsiooni juurutamiseks tulevikus.

2.5 Andmebaasi migreerimine

Arendataval hajussüsteemil on koos programmikoodiga pidevas muutumises ka andmebaas. Andmebaasi muudatusi tehakse tarkvara arendajate poolt, lisades koodirepositooriumisse SQL lausetest koosnevaid skripte, mis täidetakse automatiseerimisplatvormi konveieri poolt. Järgnevalt antakse lühike ülevaade andmebaasi migreerimisvõimalustest.

Relatsioonilise andmebaasi skeemi üheks komponendiks on relatsiooniskeemid. Üks relatsiooniskeem esindab ühe andmebaasi tabeli kirjeldust [11]. Mõningatel juhtudel on vaja andmebaasi skeemi muuta, nt. lisades uut tabelit või muutes tabeli andmevälja. Andmebaasi skeemis muudatuste tegemiseks tuleb kirjutada ja käivitada skripte, mis muudavad andmebaasi skeemi või tabelite kirjeldust. Nende skriptide käsitsi käivitamine on ajakulukas ja veaaldis. Parem lahendus on neid skripte käivitada automaatselt ning sellist lähenemist kutsutakse andmebaasi migreerimiseks [12].

Andmebaasi migreerimine tähendab tööriista abil SQL lausetest koosnevate skriptide kasutamist järkjärguliseks muudatuste tegemiseks. Hea tava on nummerdatud migreerimisskriptide hoidmine rakenduse koodiga koos ühtses koodirepositooriumis (vt. joonis 7).



Joonis 7. Migratsiooniskriptid koodirepositooriumis.

See läheb kokku põhimõttega hoida kõik asjad ühes kohas ning annab võimaluse saada hea ülevaade muudatuste ajaloost. Nummerdamine on oluline, et oleks võimalus pidada järge, mis järjekorras skripte tuleb käivitada.

Tabelis hoitakse kirjeid skriptide kohta, mis juba on selle andmebaasi muutmiseks käivitatud. Seda infot kasutavad ära erinevad tööriistad mis käivitavad juba rakendamata skripte ning uuendavad tabelit peale igat õnnestunud muudatust [13].

Lähenemisi kuidas andmebaasis muudatusi teha on kaks [14]:

- rakendada muudatusskriptid koos rakenduse paigaldusega;
- eraldada muudatuste tegemine rakenduse paigaldusest.

Muudatuste tegemisel koos rakenduse paigaldusega on järgnevad probleemid:

- Vähene kontroll muudatuste tegemise üle. Rakendus ei pruugi käivituda tekkinud probleemide tõttu ning vajab seeläbi sekkumist.
- Ajakulukamate muudatuste puhul pikeneb rakenduse uuesti kättesaadavuse aeg.
- Raskendab muudatuste koordineerimist ning andmete lukustamist kui tegemist on rakendusega, kus mitu serverit töötavad ühe andmebaasiga.

Muudatuste tegemine eraldiseisva protsessina tähendab kõigepealt muudatuste rakendamist ning kui see õnnestub, siis leitakse sobiv aeg uue versiooni paigaldamiseks. Selline viis nende kahe protsessi eraldamiseks võimaldab paremini hallata, kuidas ja millal andmebaasi skeemi muudatusi tehakse. Lisaks jäävad ära eelnevalt kirjeldatud probleemid, mis tekivad kui muudatused on seotud rakenduse paigaldamisega. Kuid sellelgi lähenemisel on puudused, millele peab tähelepanu pöörama, kui soovitakse kahte protsessi lahus hoida:

- rakenduse kood peab ühilduma andmebaasi skeemi praeguse kui ka plaanitud muudatustega;
- arvestama peab täiendava protsessi olemasoluga skeemi muutmiseks. [15]

Üldiselt alustatakse lähenemisega, kus andmebaasiskeemi muudatused tehakse vahetult enne rakenduse paigaldust. Mõningad raamistikud (Ruby on Rails, Django, .NET) sisaldavad juba vaikimisi tööriistu nende muudatuste rakendamiseks. [16]

Andmebaasiskeemi ühildumist nii hetkel kasutuses oleva skeemi kui ka plaanitud muudatustega saab teha rakendades muudatusi paralleelselt (*Parallel Change*), mis on kolmest sammust koosnev refaktoreerimise tehnika. Lühidalt tähendab see uute muudatuste lisamist olemasolevasse struktuuri nii, et vana kood jääb tööle muutmata

kujul. Seejärel uuendatakse järk-järgult vana koodi kasutus uutele muudatustele vastavaks ning kui vana kood enam ei ole kasutusel, siis eemaldatakse see rakendusest ning viiakse lõplikult vastavusse ainult uute muudatustega. [17]

Eelnevalt kirjeldatu on vajalik ka andmebaasi muudatuste jaoks, mis peavad olema tagasipööratavad. Tagasipööratavad andmebaasi muudatused on ka üheks eelduseks keerukamatele rakenduse paigaldusprotsessidele. Andmebaasi mõlema versiooniga ühilduva koodi kirjutamisel tuleb lähtuda järgnevatest põhimõtetest: kõik uued veerud peavad lubama null väärtust ehk väärtuse puudumist, päringuid tuleb teha kasutades veergude nimesid mitte lähtudes veergude järjestusest ning rakenduse kood tuleb kirjutada eeldusega, et uued veerud ei pruugi olemas olla ning ka andmebaasis olevad protseduurid on versioniseeritud või kasutatavatel parameetritel on vaikeväärtused. Kuigi sel on oma hüved, nõuab tagasipöörde võimaluse toetamine süsteemset ning distsiplineeritud lähenemist. [18]

3 Analüüs

Järgnevas peatükis esitatakse automatiseerimisserveritele esitatavad nõuded, võrreldakse automatiseerimisservereid ja lähtekoodi haldavaid tööriistu, selgitamaks välja projekti praeguseid komponente kõige paremini toetavad vahendid.

3.1 Nõuete määramine

Automatiseerimisserveritele esitatavad nõuded saadi asutuse dokumentatsioonist ja intervjueerides tiimis olevaid tarkvaraarendajaid ning asutuse peaarhitekti.

Asutuse dokumentatsioonis on olemas tarkvaraarenduse head tavad, mis soovitavad kasutada automatiseerimissüsteemi ja kirjeldavad erinevaid ülesandeid ja toiminguid, mida automatiseerimisserver peab täitma. Lisaks sisaldavad head tavad täiendavaid tegevusi, mis on seotud andmebaasi ja selle skeemis muudatuste tegemiseks vajalike skriptidega.

Nõuded rakendusele koostati põhinedes headele tavadele ning peaarhitekti soovitustele. Mittefunktsionaalsed nõuded:

- Igal ajahetkel peab olema võimalus luua andmebaas, mille struktuur ja ülesehitus vastab täielikult hetkeseisule.
- Peamises harus olev kood peab olema igal ajahetkel kompileeritav, käivitatav ja töötav.
- Rakenduse lähtekoodi hoitakse ühes tsentraliseeritud kohas, koodirepositooriumis.
- Koodirepositooriumis on olemas koodibaasi peamine haru ehk peaharu, mis sisaldab kõiki töötavaid ja terviklikke muudatusi alates algusest kuni praeguse hetkeni.
- Koodirepositooriumiga ühendamiseks peab olema süsteemne kasutaja, mitte tarkvaraarendaja isiklik kasutaja.
- Ennem andmebaasiskriptide koodirepositooriumisse kandmist tuleb veenduda, et skriptid on vormistatud korrektselt ning nende käivitamisel ei teki vigu.

Funktsionaalsed nõuded:

- Iga peaharusse sisse kantava muudatuse järel käivitatakse automaatselt testid.
- Andmebaasiskript käivitatakse rakenduse uue versiooni paigaldamisel.
- Rakenduse kompileerimisprotsess peab olema automatiseeritud kasutades selle jaoks vastavat tarkvara.
- Rakenduse paigaldusprotsess erinevatesse keskkondadesse peab olema automatiseeritud.

Nõuded eristuvad rakenduse koodi hoidmise, koodirepositooriumi ülesehituse, andmebaasi kui ka sellega seotud muudatuste rakendamise, kompileerimisprotsessi ning paigaldusprotsessi järgi. Suur osa nõudeid kattub ka eelnevalt kirjeldatud pidevintegratsiooni põhimõtetega ja on abiks automatiseerimise planeerimisel.

3.2 Automatiseerimisplatvormide analüüs

Asutuses on võimalik kasutada Jenkinsi automatiseerimisplatvormi mille paiga- ja muudatusehaldus on tagatud asutuse it-talituse poolt. Lisaks on asutuses olemas arhitektid, kellelt on võimalik saada tuge antud platvormi kasutamist puudutavates küsimustes.

Kuna asutuse poolt pakutav platvorm on tagatud tugiteenustega, jäetakse automatiseerimisserverite võrdlusest välja tasulised pilveteenused ning keskendutakse avavara litsentsiga platvormidele, mille paigaldamise ja halduse kohustus jääb kasutajatele endile. Kui tasulistel platvormidel on kasutajatugi hinna sees, siis avavara projektide puhul saab toetuda ainult selle projektiga seotud kommuunile. Seetõttu on oluline, et platvormil oleks põhjalik dokumentatsioon, et tekkinud probleemidele oleks võimalik sealt vastuseid leida.

Automatiseerimisserveri kasutusviise on vähemalt kaks – variant kus kogu töö teeb ära ühele serverile paigaldatud tarkvara või jagatud vastutusega hajutatud variant, kus peamine server, millele on paigaldatud tarkvara, kasutab töö koormuse jaotamiseks kliente, mida nimetatakse ka *master*/*slave* arhitektuuriks. Serverile jääb hulk ülesandeid mida klientidele ei delegeerita kuid kliendid, mida erinevad platvormid nimetavad

erinevalt (*agent*, *runner*, *slave*, *node*) täidavad serveri poolt neile suunatud ressursinõudlikud tööd, vabastades peamise serveri koormusest. [19]

Arvestades automatiseerimistarkvara hinda, litsentsi, dokumentatsiooni põhjalikkust ja automatiseerimisserveri töö koormuse jaotamiseks klientide kasutamise võimaluse olemasolu, hinnatakse järgnevaid automatiseerimisservereid:

- TeamCity;
- Jenkins;
- GoCD;
- Drone.

TeamCity platvorm ei vasta küll avavara litsentsile, kuid jääb valikusse kuna on hetkel kasutuses olev platvorm ning tasuta pakutav funktsionaalsus katab vajaliku. Loodud Tšehhi ettevõtte Jetbrains poolt [20] ning proovivara võimaldab platvormil luua kuni 100 konveierit ning kasutada kuni kolme agenti nende käivitamiseks [21].

Jenkins on Javas kirjutatud MIT litsentsi kasutav automatiseerimisserver [22]. Rahastus toimib läbi annetuste, ühisrahastuse või seotud mittetulundusühingute kaudu [23].

GoCD platvorm, kasutab Apache 2.0 litsentsi, põhiliselt Java ning Typescripti abil kirjutatud [24] ning sponsoreeritud ettevõtte Thoughtworks poolt [25].

Drone on Go keeles kirjutatud platvorm mis kasutab samuti Apache 2.0 litsentsi [26]. 2020 augustis omandas Harness Inc Drone platvormi [27] lubades investeerida ja panustada Drone platvormiga seotud kogukonda.

Platvormi paigaldamise võimalused kasutamiseks on erinevaid. Kasutatakse nii täitmiskõlblike programmifaile (*executables*), mis paigaldavad arvutisse servertarkvara, kui ka konteineritel põhinevaid lahendusi (vt. tabel 2). Mõningad platvormid pakuvad ka rakenduse faile kokku pakitud kujul, mis ei vaja rakenduse paigaldust arvutisse, vaid võimaldab käsurea käskude abil rakendust käivitada [28]. Ka toetavad paljud pilveteenused mainitud automatiseerimisserverite paigaldamist juba eelnevalt selle teenuse jaoks soovituslike sätetega pilvepõhisele platvormile [29].

Tabel 2. Automatiseerimisplatvormide paigaldamisvõimalused.

	TeamCity	**Jenkins**	**GoCD**	**Drone**
Windows	Programmifail ja käivitatav komplekt kompileeritud faile kättesaadavad kodulehelt [30]	Kodulehelt kättesaadav programmifail [31]	Programmifail kodulehelt kättesaadav [32]	Konteineriseerit ud rakendus kättesaadav Dockeri avalikust hoidlast [33]
Linux		Suurele osale distributsioonide st on kompileeritud failid kättesaadavad ning väiksele hulgale pakette haldab kolmas osapool [34]	Paketid enamlevinud distributsioonidele on kodulehelt kättesaadavad [35]	
MacOS		Kolmandate osapoolte poolt hallatud pakett [36]	Programmifail kodulehelt kättesaadav [37]	

Kui rakenduse lähtekoodi kompileerimisprotsessi pole testitud konteineris, mõni rakenduse komponent ei tööta konteineris või ei õnnestu konteineri loomine on võimalus lähtekood kompileerida Windows operatsioonisüsteemiga kliendis. Windowsil põhinev klient on vajalik kuna kompileeritaval rakendusel on sõltuvused vanematest .NET raamistikest, mis töötavad ainult Windows platvormil [38]. Seetõttu peavad automatiseerimisserverid toetama Windows operatsioonisüsteemi kliendi lisamist, milles lähtekood kompileerida, testid käivitada ning õnnestumise puhul tekkinud tehised paigaldamiskeskkonda kopeerida.

Windows operatsioonisüsteemiga klient on toetatud kõikidel automatiseerimisserveritel peale Drone, kus seda on võimalik kasutada kuid see on beetaversioonis ning ei ole soovituslik toodangukeskkonna jaoks.

Automatiseerimisplatvormid võimaldavad kasutada suurel hulgal pistikprogramme (vt. tabel 3), mis laiendavad olemasolevaid võimalusi ning lisavad uut funktsionaalsust, mida algselt platvormil ei ole [39]. Nendeks võivad olla nii kompileerimistööriistad erinevate automatiseerimiseks vajalike sammude toetamiseks, konveieri käivitamist abistavad

laiendused, aruandlusega seotud, platvormipõhised kui ka erinevaid teavitusvõimalusi pakkuvad pistikprogrammid.

Tabel 3. Automatiseerimisserverite pistikprogrammide arv.

	TeamCity	**Jenkins**	**GoCD**	**Drone**
Pistikprogrammide arv	173 [40]	1877 [41]	95 [42]	131 [43]

Üheks kasulikuks täienduseks on erinevate teavituste saatmise laiendus. Konveieri töö tulemustest teavitamine on info edastamise seisukohalt oluline ning autor pidas vajalikuks laienduse olemasolu, mis võimaldab saata teavitusi Teamsi rakenduse vestlusesse. E-kirjade saatmine ning postkastis hoidmine suurendab digiprügi hulka, lisaks võtab ka tühi e-kiri ruumi [44] ning ilma neid kustutamata võib neid postkasti kuhjuda. Teamsi rakendusse teavituse saatmine jõuab koheselt selle kanaliga liitunud inimesteni. Teamsi teavitusvõimalust võimaldav laiendus on olemas ainult TeamCity ja Jenkins platvormil [45].

Täiendav funktsionaalsus, mida automatiseerimisplatvormid pakuvad on veebipõhine masinloetav liides millele saab HTTP (*Hypertext Transfer Protocol*) päringuid saata. Info vahetamiseks kasutatakse JSON vormingut või päringuid serveri veebiaadressidele, mis võimaldavad aadressireal kasutada parameetreid soovitud vastuse või muudatuse saavutamiseks. Automatiseerimisserveri rakendusliidesele saab saata päringuid kasutades neid erinevateks eesmärkideks, muuhulgas [46]:

- Serveri konfiguratsiooni ja konveierite info pärimiseks.
- Konveierite käivitamiseks ning haldamiseks kui ka nende töö tulemuste pärimiseks.

Vaikimisi käivitatakse konveierfailis defineeritud käsud järjestikku, mis tähendab, et ennem ei käivitu järgmine, kuni eelnev pole lõpetanud. Parema kiiruse saavutamiseks on mõistlik mõningad sammud käivitada paralleelselt [47]. Järgnevalt hinnatakse platvormide võimekust käivitada samme paralleelselt (vt. tabel 4).

Tabel 4. Paralleelselt käivitatavate sammude tugi.

TeamCity	**Jenkins**	**GoCD**	**Drone**
Võimaldab konveierfailis kirjeldada märksõna *parallel* abil paralleelselt käivitatavaid samme [48]	Sammud märksõnadega *parallel* [49] ja *matrix* [50]võimaldavad konveierfailis kirjeldada samme, mis peavad paralleelselt käivituma	Ühes faasis (*stage*) olevad üksteisest mittesõltuvad sammud käivitatakse paralleelselt [51]	Käivitatavale sammule saab määrata märksõnaga *depends_on* sammud mis täidetakse paralleelselt enne käivitatavat sammu [52]

Erinevad platvormid tõlgendavad parallelismi erinevalt. Mõningad võimaldavad ette antud käske samal kliendil käivitada paralleelselt, kuid kasutatakse ka horisontaalset skaleerimist, kus mitu klienti tegelevad ette antud käskude paralleelselt käivitamisega. [53]

3.3 Rakenduse lähtekoodi tööriistade analüüs

Rakenduse lähtekood on kirjutatud kõrgkeeles, mis on mõeldud arvutile täitmiseks. Kuigi rakendust saab käivitada rakenduse lähtekoodi kasutades, on see mõeldud kasutamiseks ainult tarkvara arenduse etapis. Kasutatava rakenduse ehk täitmisprogrammi loomiseks on vajalik lähtekood kompileerida loomaks käivitatav rakendus, mis on valmis paigaldamiseks soovitud keskkonda, näiteks Windows Serveri IIS veebiserverisse või kasutamiseks teises arvutis.

Kõrgkeeles kirjutatud lähtekoodi kompileerimiseks on erinevaid tööriistu ja tarkvarateeke, mis mõningal juhul sisemuses kasutavad neidsamu tööriistu, kuid omavad täiendavat funktsionaalsust võimaldamaks teha enamat kui tegeleda ainult kompileerimisprotsessiga. Kompileerimist abistavad tarkvarateeke saab kombineerida erinevate kõrgkeeltega ning nii luua lisaks lähtekoodi kompileerimisele täiendavaid võimalusi. [54]

Eesrakenduse lähtekoodi ei kompileerita täitmisprogrammiks, vaid failid töödeldakse ning valmistatakse ette pidades silmas laadimiskiirust ning veebilehitseja eripära uute

andmete laadimisel. Vajalik on läbida teatud hulk samme, et koodifailid oleksid veebilehitseja poolt kiiremini laetavad.

3.3.1 Tagarakenduse lähtekoodi kompileerimise tööriistade analüüs

Kaks käsurea tööriista .NET raamistikule põhinevate rakenduste lähtekoodi kompileerimiseks on NAnt ja MSBuild (*Microsoft Build Engine*) [55].

NAnt on välja kasvanud Java Ant tööriistast [56] ning MSBuild on Microsofti poolt loodud tööriist. MSBuild tööriista kasutavad tarkvaraarenduskeskkond Visual Studio kui ka .NET raamistiku käsurea tööriistad (*dotnet*). NAnt viimane versioon on olnud 2012 juunist saadik 0.92 [57] ning kuigi avalikus koodirepositooriumis on näha projekti kallal tegutsemist ka aastal 2022, siis aktiivne töö projekti kallal on lõppenud aastal 2015. Sellest olenemata siiski kasutatakse antud tööriista veel üksikutes projektides [58].

MSBuild kasutab XML vormingus projektifaili ning selles olevaid võti-väärtus paare kompileerimisprotsessi konfigureerimiseks. Tarkvaraarenduskeskkond Visual Studio kasutab samuti iga projekti juures projektifaili, mida saab muuta tarkvara kaudu või kasutades tekstiredaktorit. Seeläbi Visual Studio kasutab MSBuild tööriista, kuid MSBuild ei sõltu Visual Studiost. [59]

.NET raamistikul on käsurea tööriistad, mida saab käivitada märksõna *dotnet* abil [60] ning üheks nendest on lähtekoodi kompileerimiseks kasutatav käsklus *publish* (*dotnet publish*) [61]. Seda käsklust kasutades pöördutakse taustal MSBuild poole ning kõik parameetrid saadetakse samuti edasi. Kompileerimise käigus loetakse projektifaili sisu ning avalikustatakse sisu määratud kausta [62].

Kui käsurea tööriist sobib etteantud käsu täitmiseks ning integreerimiseks etappidesse, kus täidetakse käsurea lauseid, siis kompileerimistööriistadeks on tehtud terveid tarkvarateeke, mis sisaldavad täiendavat võimekust ning funktsionaalsust [63]. Samuti on need erinevate keelte jaoks loodud. Mõningad nendest võimalikest on järgmised:

- NUKE — C# [64];
- FAKE — F# [65];
- Psake — PowerShell [66].

Kui käsureatööriistadel on tööks vajalik projektifaili olemasolu, siis tarkvarateekidel põhinevad tööriistad võivad vajada enne kasutamist täiendavate konfigureerimisfailide lisamist enne kui neid kasutada saab [67]. Lihtsamal kujul koosnevad tarkvarateekidel põhinevad lahendused ühest põhifailist [68], mis tegeleb kompileerimisprotsessi ning vajadusel ka paigaldusega. Sama lahendus on kasutusel ka autori tiimis arenduses oleva tarkvara kompileerimis- ja paigaldusprotsessi käivitamiseks. Antud faili tööd saab juhtida sisendparameetrite abil [69] ning protsessi käigus kasutatakse ära teegi poolt pakutavat täiendavat funktsionaalsust etappide täitmiseks, mida ilma antud tööriistata pidi tegema täiendavaid tööriistu kasutades. Mõningad täiendavad tegevused, mida tarkvarateek lihtsustab on seotud nii kaustade ja failide haldamisega ning nende tihendamisega [70] kui ka teavituste saatmisega. Lisaks saab antud teekide abil pöörduda MSBuild tööriista poole programmeerimiskeele või mõne teise keele funktsioone kasutades ning konfigureerida neid aheldades soovitud meetodeid – järgnevalt on näitena esitatud NUKE teegi programmikood (vt. joonis 8) [71].

```
MSBuild(_ => _
    .SetTargetPath(SolutionFile)
    .SetTargets("Clean", "Build")
    .SetConfiguration(Configuration)
    .EnableNodeReuse());
```

Joonis 8. Tööriista konfigureerimine teegi meetodite abil.

Kui tarkvarateekidel põhinevaid tööriistu võib olla algselt veidi keerukam tööle häälestada, siis vastukaaluks pakuvad need lisafunktsionaalsusi, mis võivad keerukamate projektide puhul suureks abiks olla. Lihtsamatel juhtudel saab kasutada käsureatööriistu kombineerides neid skriptimiskeeltega.

3.3.2 Eesrakenduse lähtekoodi tööriistade analüüs

Eesrakenduse programmikoodi puhul valmistatakse lähtekood ette toodangukeskkonna jaoks. Selle jaoks, et programmikood oleks sobiv ning ei pärsiks veebilehe kiirust, tuleb see tükeldada väiksemateks failideks, minimeerida sisu, eemaldada sellest kasutamata kood ning lõpuks genereerida failidele omapärane nimi, et sundida veebilehitsejat alla laadima uut sisu, mitte kasutama vahemälus olevat [72]. Neid ülesandeid aitavad täita tööriistad, mis integreerivad endas pistikprogramme erinevate ülesannete täitmiseks. Mõningad neist tööriistadest (*bundlers*), mille pistikprogrammid pakuvad tuge arendatava hajusrakenduse poolt kasutuselolevate .vue (*Vue Single-File Component*) ja .scss (*Sassy Cascading Style Sheets*) laiendustega failide töötlemiseks on järgnevad:

- Webpack;
- Esbuild;
- Vite.

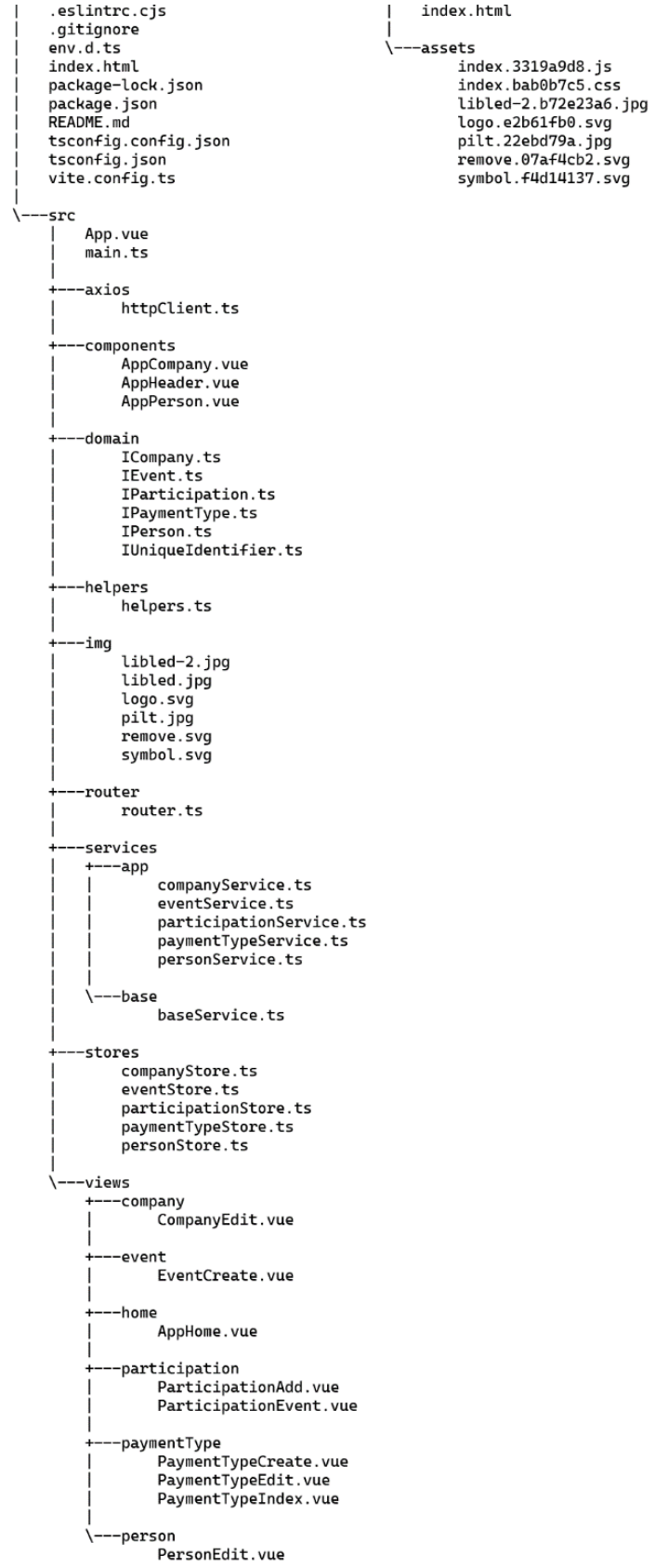
Eesrakenduse lähtekood koosneb suurest hulgast üksteisest sõltuvatest failidest, mille funktsionaalsus on jaotatud erinevatesse moodulitesse. Funktsionaalsust ühes moodulis saab kasutada teises ja vastupidi. Teises failis asuva sisu kasutamiseks peab soovitud sisu importima viidates failile, kus soovitud sisu asub ning samuti peab see olema teise faili poolt eksporditud [73]. Selliste sõltuvuste abil on võimalik koostada moodulite graaf, mis sisaldab endas infot erinevate moodulite vaheliste sõltuvuste kohta.

Kui kõik üksteisest sõltuvad failid on leitud, koostatakse nendest mitu väiksemat tükki ning neid kasutatakse ja laetakse veebilehitsejas vastavalt vajadusele. Ilma tükeldamata oleks tulemus üks suur fail, mille kasutamine võib piirata jõudlust kuna veebilehitseja saab selle sisu lugeda alles peale selle allalaadimist, mis võib aeglasemate ühenduste puhul olla märgatav ajakulu [74].

Peale tükeldamist on failide sisu minimeerimine üks optimeerimise võimalusi, mille eesmärk on teha failid nii väikseks kui võimalik ilma tihendamisprogrammi kasutamata. Seda tehakse eemaldades failidest kõik üleliigne, milleks võivad olla muuhulgas järgnevad tegevused [75]:

- Kommentaaride eemaldamine.
- Tühja ruumi lisavate märgiste (tühik, tabulaator) eemaldamine.
- Kasutamata koodi eemaldamine.
- Muutujate ja funktsioonide nimede lühendamine.

Kõikide tegevuste eduka lõpu puhul on tulemus üksik kaust, mis sisaldab kõiki veebilehe tööks vajalike faile. Järgnevalt on esitatud üks näide, kus mitmekümnest programmi- ja pildifailist koosnevast võrdlemisi väikesest projektist saadi Vite tööriista kasutades kaheksast failist koosnev kogum, mis sisaldab kõike, mida vaja, et ühte veebilehte kuvada (vt. joonis 9). Joonisel on vasakul näha projekti algne failide hulk ning paremal saadud tulemus.



Joonis 9. Failide ettevalmistamine.

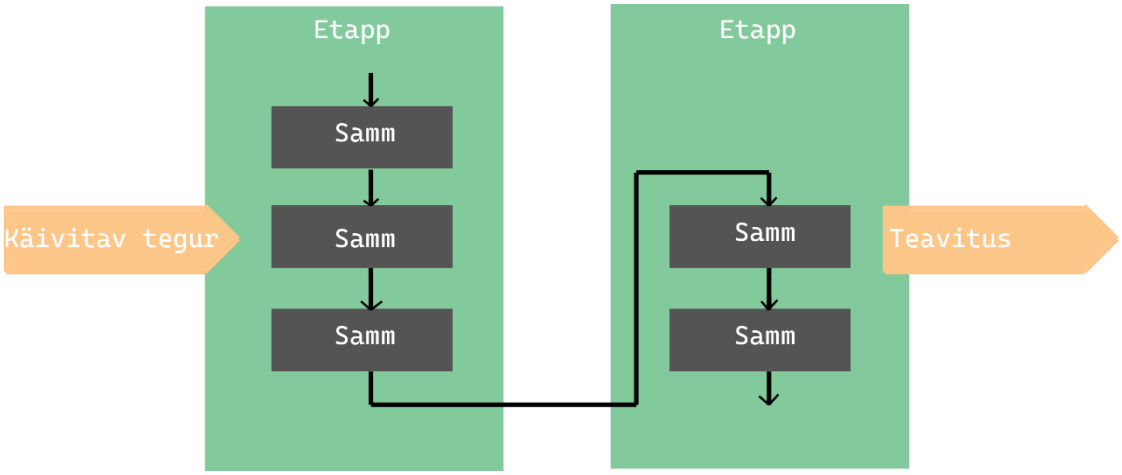
Eelmainitud teisenduste läbiviimiseks tööriista valides on üheks oluliseks komponendiks jõudlus ning sellest tulenevalt aeg. mille jooksul protsessi läbi viiakse. Jõudluse

saavutamiseks on oluline keel, milles tööriist on kirjutatud. Kui esimesed tööriistad kirjutati kasutades JavaScripti, siis uuemad on loodud kasutades keeli nagu Rust ja Go, mis on lähedasemad madalkeelele, võimaldades arvuti ressursse paremini kasutada ning seeläbi jõudlust suurendada [76].

Kui projekt sisaldab palju vanu väliseid sõltuvusi, võib Webpack tänu oma laialdasele pistikprogrammide toele olla parim valik. Pistikprogrammid võivad Webpack uuendamise küll keeruliseks muuta, kuid samas omab Webpack head dokumentatsiooni, juhendeid ja lisatööriistu. Suurte projektide puhul võib Webpack aeglaseks jääda ning sel juhul on Vite sobivam. Kuna Vite kommuun on alles noor kuid kiiresti kasvav (Vite v1.0.0-rc.5 versioon avaldati 23.10.2020 [77]) on võimalik, et puuduolev funktsionaalsus lisatakse üsna kiiresti. Esbuild sobib paremini teekide sisu optimeerimiseks. Samuti sõltub projekti kaasautoritest, millist tööriista kasutada. Oleneb sellest kui tuttavad mingi tööriistaga ollakse. [78]

3.4 Konveieri analüüs

Konveier koosneb erinevatest etappidest, mis omakorda koosnevad sammudest. Konveier käivitub üldjuhul millegi ajendil (*trigger*) ning asub etappides määratud samme täitma, millest oma töö lõppedes teavitab asjast huvitatuid (vt. joonis 10).



Joonis 10. Konveieri töökäik.

Erinevad automatiseerimisplatvormid kohtlevad konveiereid erinevalt ning võimaldavad neid luua nii platvormi kasutajaliideses kui ka määrata üksikute failidena, mille sisu on mõeldud täitmiseks. Faili sisu kirjeldamiseks on võimalik valida erinevate keelte ja vormingute vahel, millest mõni võib olla loetavam kui teine. Lisaks võivad platvormi

nõuded mõjutada failis sisalduvate väärtuste kogust ning keerukust. Manuaalsete sammude vältimiseks on võimalik konveier käivitada automaatselt peale mingit toimingut, näiteks peale koodi lisamist koodirepositooriumi peaharusse. Töö lõppemisest ja tulemustest on võimalik saada automaatne teavitus.

3.4.1 Konveieri faili asukoha analüüs

Kuigi erinevad platvormid võimaldavad konveiereid luua ka graafilise liidese abil, platvormile sisse logides, siis pakub konveieri sätete kirjeldamine konveierfailis ning selle koodirepositooriumis hoidmine mitmeid hüvesid [79]:

- Konveierfaili muudatustest jääb maha jälg.
- On üksainus tõe allikas (*single source of truth*) koodirepositooriumi näol kus hoitakse rakenduse koodi kui ka konveierfaili.
- Konveierfaili muutmine läbib sarnaselt rakenduse koodiga arvustusprotsessi.

Lisaks võimaldab see meeskonna liikmetel vajadusel konveierfaili sisuga tutvuda ilma, et peaks taotlema õiguseid või kasutajakontot sisse logimiseks või otsima keskkondi kuhu sisse logida, et näha mida konveieri sisu endast kujutab [80].

Eelnevalt valitud automatiseerimisplatvormidest toetavad kõik konveierfaili kaudu konveieri kirjeldamist väikeste erisustega (vt. tabel 5).

Tabel 5. Automatiseerimisplatvormide konveierfailide võimalused.

TeamCity	**Jenkins**	**GoCD**	**Drone**
XML vormingus või Kotlini keelel põhinev konveierfail on toetatud	Groovy keelel põhinev konveierfail kui ka väikeste piirangutega deklaratiivne versioon	Pistikprogrammide abil JSON vormingus ja YAML keeles põhinevad konveierfailid on võimalikud	YAML keeles põhinev fail repositooriumis kirjeldamaks konveierit

Ka graafilise liidese kaudu loodud konveieri taustal töötab XML vormingus fail, mis talletab automatiseerimisplatvormil loodud konveieri konfiguratsiooni. Konveierfail ei pea olema tingimata koodiga samas repositooriumis. Asutuses kus autor töötab, hoitakse projektiga seotud konveierfaile eraldi repositooriumis.

3.4.2 Konveieri süntaksi analüüs

Kui kasutatavad vormingud ja keeled on mõeldud inimloetaval kujul andmeid esitama ja kirjeldama ka nii, et arvuti sellest aru saaks siis Jenkins platvormil on lisaks täiendav deklaratiivne versioon, mis on mõeldud lihtsustamaks konveierfaili nii palju kui võimalik, et ka inimesed, kes igapäevaselt koodi ei kirjuta saaksid sellest aru. Deklaratiivne versioon põhineb Groovy keelel mõningate piirangutega [81].

Kui Kotlin ja Groovy keeled võimaldavad failis täita keerukaid ülesandeid kasutades klasse, meetodeid, objekte ja muid keele konstruktsioone siis erinevalt nendest on JSON ja YAML vorming mõeldud lihtsustama selle sisu lugemist ning on oma iseloomult kasutajasõbralikumad. Erinevalt JSON vormingust ei kasutata YAML puhul loogelisi sulge, kantsulge jms erilist tähendust sisaldavaid märke. Siiski on kasutuses tühikute abil tekitatav taane mis aitab määrata võti-väärtuspaaride hierarhiat [82].

3.4.3 Konveieri automaatse käivitamise meetodi analüüs

Automatiseerimisplatvorm jälgib teatud tegureid, peale mille käivitumist või muutumist käivitatakse konveier. Konveieri automaatse käivitamise võimalus on vajalik, et vähendada manuaalselt tehtavaid samme. Mõningad võimalused konveieri automaatseks käivitamiseks on järgnevad [83]:

- Koodirepositoorium teavitab automatiseerimisserverit muudatusest ning käivitab konveieri.
- Automatiseerimisserver kontrollib periooditi koodirepositooriumi seisu ning käivitab konveieri kui on tuvastanud muudatuse.
- Konveier käivitatakse periooditi kindla ajakava alusel ilma väliseid muudatusi arvestamata.
- Saates HTTP päringu valitud parameetritega aadressile, käivitatakse parameetrite abil määratud konveier.
- Konveier käivitub peale eelmise konveieri lõpetamist.

Kindlaksmääratud ajakava alusel käivitatav konveier võimaldab ajamahukaid töid käivitada öösel või ajal, mil inimesi aktiivselt projekti kallal töötamas pole. See võimalus konveierit käivitada on kasulik, kui on suur hulk teste, mille käivitamine võtab kaua aega. Ajakulu saab seada öösse ning hommikul, kui inimesed tööle tulevad, on võimalik näha konveieri tulemust. See tähendab, et peale muudatusi projekti koodis käivitatakse

väiksem hulk teste, mis testivad küll pinnapealselt kuid piisavalt sügavuti, et suuremad vead tulevad välja. [84]

Konveierite ühendamine võimaldab ülesandeid jagada konveierite vahel ning käivitada neid sõltuvalt eelneva konveieri tulemustest [85]. Nii on võimalik arendatava hajusrakenduse testprojektide lähtekoodi kompileerimine ja testimine jätta ühte konveierisse ning selle täitmise õnnestumisel käivitada automaatselt järgmine konveier mis kompileerib ülejäänud projektide lähtekoodi ning valmistab ette eesrakenduse paigalduseks veebiserverisse.

3.4.4 Konveieri tulemustest teavitamise meetodi analüüs

Konveieris kirjeldatud sammude täitmine võib lõppeda edukalt ilma probleemideta või tekib mõne sammu juures viga. Tekkinud vea puhul jäetakse konveieri täitmine pooleli ning automatiseerimisserver on esimene, mis selle kohta infot omab. Kuna vea tekkimisel on vaja see võimalikult kiirelt parandada, on vaja ka tarkvaraarendajatel teada saada kuidas konveier oma töö lõpetas. Mooduseid info edastamiseks on mitmeid:

- e-kiri;
- rühmavestlus;
- muu andmekandja.

Kõige traditsioonilisem on e-kirja saatmine konveieri töö tulemusest. E-kirju on võimalik reeglite abil suunata ette nähtud kaustadesse mis aitab postkasti korras hoida. Lisaks on suur tõenäosus, et asutuses on igal töötajal e-posti aadress olemas ning nad oskavad seda kasutada. Samuti on võimalik, et postkasti tuleb juba niigi palju kirju ning teavitusi puudutavad kirjad jäävad tähelepanuta või mattuvad teiste alla.

Ka on võimalus saata sõnum rühmavestlusse (nt. Teams, Slack), mille kaudu on võimalik teavitada kõiki vestluses olevaid isikuid konveieri staatusest. Kui vahetu suhtluskanalina on rühmavestluse tarkvara juba pidevalt avatud, siis teavituste saatmine sinna on moodus, kuidas ära kasutada juba kasutuses olevat platvormi. Kui e-kirjad ega rühmavestlused ei sobi, siis võib infot saata kasutades muid andmekandjaid. Kui kontorisse on paigaldatud kuvarid mis kuvavad infot ja muid näitajaid projekti kohta, saab ka konveierite tulemused vajadusel seal samuti kuvada.

3.4.5 Andmebaasi migreerimistööriistade analüüs

Andmebaasi migreerimistööriistad vastutavad andmebaasi muudatuste järk-järgult rakendamise eest. Samuti peavad need arvestust juba rakendatud muudatuste üle. Kuna muudatusskriptide käsitsi rakendamine on ajakulukas ja veaaldis on võimalus kasutada antud tegevuste automatiseerimiseks tööriista. Arvestades projektis kasutusel olevat andmebaasi (Microsoft SQL) on võimalikud valikud järgnevad:

- DbUp [86];
- Evolve [87];
- Grate [88].

Kuigi nende tööriistade ülesandeks on rakendada SQL skripte on, igal tööriistal oma moodus, kuidas neid skripte haldama peab. Evolve jaoks on vajalikud failinimed [89] ning failide asukoha saab määrata parameetri abil [90]. DbUp jaoks pole oluline kus skriptid asuvad, vaid selle saab jooksvalt määrata, kust neid otsida. Kui eelnevad tööriistad sõltusid failinimedest, siis DbUp puhul pole ette nähtud reegleid failinimetuse osas ning seetõttu on vajalikud eelnevad reeglid failide asukoha ja nimetuse osas [91].

DbUp ning Evolve loovad ka andmebaasi tabeli, milles peavad järge selles andmebaasis juba rakendatud skriptide üle [92]. Grate ei loo andmebaasi jaoks tabelit ning ei pea järge, vaid tugineb kaustade süsteemile, mis võimaldab määrata, millal skriptid käivitatakse. Kaustad jagunevad kolme tüüpi, mis määravad sealsete skriptide käivitusomadused. Kaustad on skriptidele, mida käivitatakse ainult ühe korra, skriptidele mida käivitatakse kui olemasolevat skripti on muudetud või on uus lisatud ning skriptidele mida käivitatakse iga kord. [93]

Samuti on võimalik luua uus andmebaas, mis vastab kõige uuemale seisule. Seda on võimalik teha kombineerides andmebaasi skeemi kustutamise ja taasloomise ning sellele skriptide rakendamisega. Selliste toimingute puhul tasub veenduda, et tegevusi tehakse soovitud andmebaasi peal, mitte sellisel mille andmed peavad säilima.

Skriptide käivitamine on võimalik lisada rakenduse käivitusprotsessi muutes .NET rakenduste puhul *Program* nimelist faili. Sellist lähenemist kasutavad DbUp [94] ja Evolve [95] mis võimaldavad läbiviidavaid tegevusi ka täiendavalt konfigureerida ning seejärel käivitada. Ebaõnnestumise korral on võimalik ebaõnnestunud migreerimine

logida. Grate puhul on võimalik kasutada käsurida, kus hetkel on ainuke kohustuslik parameeter andmebaasi aadress ning hulk valikulisi parameetreid, mille abil on muuhulgas võimalik määrata SQL failide asukohta, keskkonnamuutujaid ja teisi skriptide rakendamist puudutavaid väärtuseid [96].

3.5 Automatiseerimisplatvormi ning tööriistade valik

Olles eelnevalt võrrelnud automatiseerimisplatvorme ning konveiereid ja erinevaid tööriistu nii lähtekoodi kompileerimiseks, koodifailide ettevalmistamiseks kui ka andmebaasi migratsioonide läbiviimiseks, on vaja neist kasutusele võtta üks komplekt.

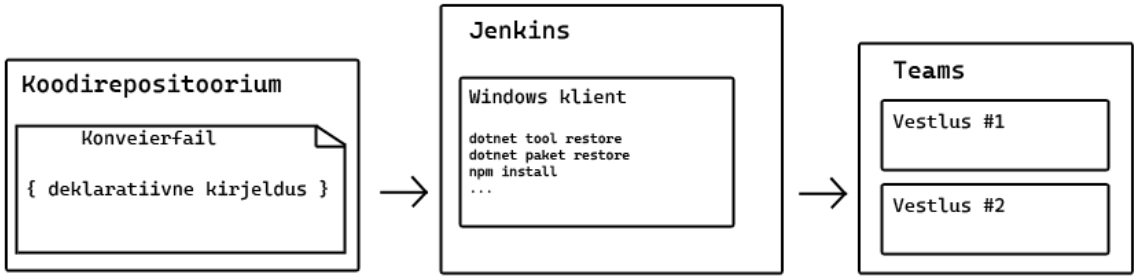
Automatiseerimisplatvormide seast on sobivaim Jenkins. Jenkins platvormi puhul on eeliseks võimalus kasutada konveierfailide täitmiseks platvormist eraldiseisvat klienti. Klient on Windows Server 2019 server, mis tegeleb ainult konveierfailide täitmisega ning mille ressursid ei ole samal ajal hõivatud Jenkins platvormi toimingutega. Lisaks on Jenkins platvormi poolt kasutatav klient võimekama riistvaraga kui hetkel kasutusel olev TeamCity server.

Jenkins platvorm on ka ainuke, millel on tugi teavituste saatmiseks Teams vestlusesse ning võrreldes teiste rakendustega on sellel olemas asutusepoolne tugi ja kompetents. Tugiteenuse olemasolu tõttu puudub vajadus meeskonnal tegeleda tarkvara paikade ja uuendite haldamisega. Lisaks on asutuses olemas inimesed, kes oskavad vajadusel pakkuda tuge küsimustes, mida muidu peaks otsima foorumitest või dokumentatsioonist.

Konveierfaili asukohaks on sobivaim koodirepositoorium, see soovitus tuleb nii eelnevalt kirjeldatud pidevintegratsiooni praktikatest kui ka arhitektidelt. Lähtudes nii lihtsusest kui ka soovist, et kõik saaksid faili sisust aru, on valitud faili sisu kirjeldavaks süntaksiks Groovy keele lihtsustatud deklaratiivne versioon. Konveieri automaatseks käivitamiseks on valitud koodirepositooriumis tuvastatud muudatuste ajendil käivituv konveier, mis saadab teavitusi oma tulemustest Teams rakendusse.

Andmebaasi migreerimisskriptide haldamiseks jääb endiselt kasutusel DbUp kuna selle muutmine eeldab alustamist F# programmifaili tükeldamise või asendamisega mõne teise teegiga, mida on võimalik integreerida C# keele abil. Samuti on hetkel sisse töötatud migreerimistööriista jaoks SQL failide nimetuse ja paigutuse süsteem. F# programmifaili tükeldamine aitaks andmebaasi migreerimisega tegeleva osa eraldada ning seal tööriista

kasutamine välja vahetada, kuid pikas plaanis tekitab raskuseid F# keel. Võimalik oleks võtta kasutusele teek, mida on võimalik integreerida programmeerimiskeelega, millest on tiimi tarkvaraarendajatel põhjalikumad teadmised. Lisaks seni kuni kasutatakse F# keelel põhinevat faili, ollakse sõltuvuses FAKE teegist mille abil hallatakse lähtekoodi kompileerimisprotsessi. Valitud tööriistade abil on kolme etapi (vt. joonis 11) läbimiseks põhilised tööriistad olemas.



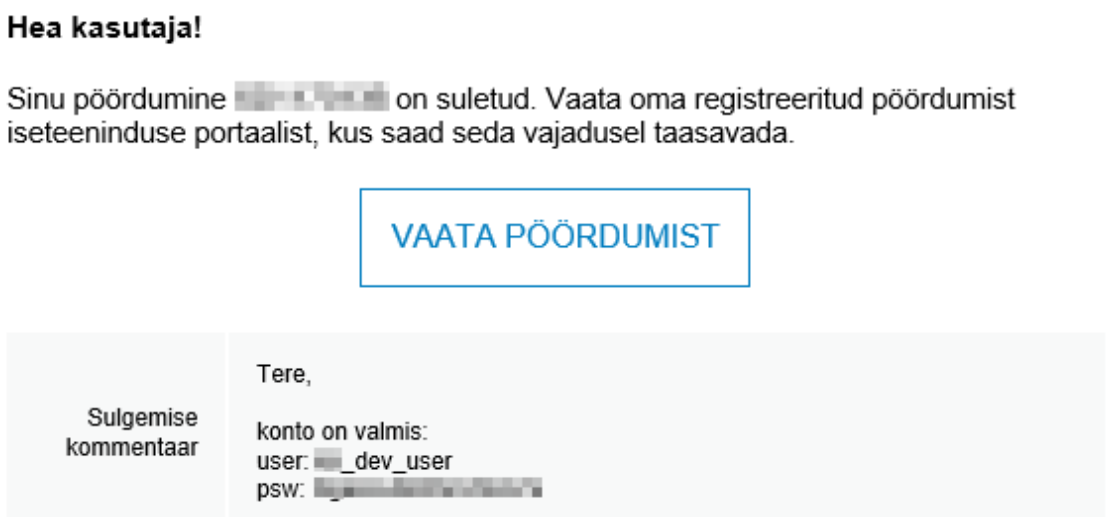
Joonis 11. Töö etapid.

Kuna rakendusel on mitmeid vanu väliseid sõltuvusi vanematest tarkvarateekidest ning hetkel pole kindel, kas laialdane pistikprogrammide kasutusest tulenev funktsionaalsus on olemas ka mõnel teisel eesrakenduse lähtekoodi töötleval tööriistal, peab hetkel kasutusse jääma endiselt Webpack.

Küsides hinnangut teekide FAKE ja Webpack pikaajalisele kasutusele tiimis olevatelt tarkvaraarendajatelt, soosisid vastused pigem praeguste teekide kasutamise jätkamist. Kuid ei välistatud muutuseid tulevikus, juhtides tähelepanu sellele, et eelnevalt peaks võimalike variante nii analüüsima kui ka olemasoleva projekti peal proovima. Täiendavalt toodi välja soov esmajärgus vahetada välja senine TeamCity platvorm.

4 Lahenduse käik

Esimene eesmärk on luua valitud automatiseerimisserveril töötav konveier, mis käivitub automaatselt peale programmikoodi lisamist koodirepositooriumisse ning paigaldab testitud rakenduse failid IIS veebiserverisse. Vältimaks võimalike konflikte olemasoleva arenduskeskkonna rakendusega otsustatakse luua rakenduse värskeimaile seisule vastav andmebaas ning luuakse IIS veebiserverisse koht, kuhu rakendus paigaldatakse. Konveier luuakse pannes rõhku selle loomise kiirusele ja järk-järgult muudatuste testimisele ning seetõttu väga suurt rõhku konveierfaili kirjelduse optimaalsusele ei panda, mille tõttu on esialgses failis vähesel määral korduvat koodi. Lisaks tellitakse kasutajatoelt eelnevalt süsteemne kasutaja, mille eesmärk on luua ühendus koodirepositooriumiga (vt. joonis 12).



Joonis 12. Süsteemne kasutaja repositooriumi jaoks.

Rakenduse IIS veebiserverisse paigaldamise jaoks luuakse mõningad eeldused. IIS veebileht vajab isoleeritud keskkonda (*Application Pool*), mis hoiustab veebilehte ning probleemide tekkimisel ei mõjuta teistes keskkondades töötavaid veebilehti. Kuna plaanitav veebileht pole serveris ainuke, siis kontrollitakse eelnevalt, et veebilehe hoiustamiseks mõeldud keskkond ning soovitud port ei ole kasutuses. IIS veebilehe jaoks vajalike keskkondi luuakse ja kontrollitakse kasutades käsurea programme AppCmd (vt. joonis 13) ja PowerShell, [97] kuna PowerShell käskude puhul ei tekkinud alati IIS haldurisse soovitud tulemust.

```
C:\Windows\System32\inetsrv\appcmd.exe list appool
Get-IISSite | Where-Object {$_.Bindings -Like "*10055*"}
```

Joonis 13. Veebilehe eelduste kontrollimine.

Kui selgub, et IIS veebilehe jaoks soovitud nimetus ja port on kasutuseta, alustatakse veebilehe jaoks vajalike toimingute tegemist. Luuakse isoleeritud keskkond, kaust mida konveieris kasutatakse rakenduse failide paigaldamise jaoks ning veebilehe serveerimiseks (vt. joonis 14).

```
C:\Windows\System32\inetsrv\appcmd.exe add apppool /name:"Keskkond
(jtest)"
New-Item -Path "D:\Inetpub\Virtual" -Name "Kaust.jtest" -ItemType
"directory"
New-SmbShare -Path "D:\Intepub\Virtual\Kaust.jtest" -FullAccess
'Everyone'
```

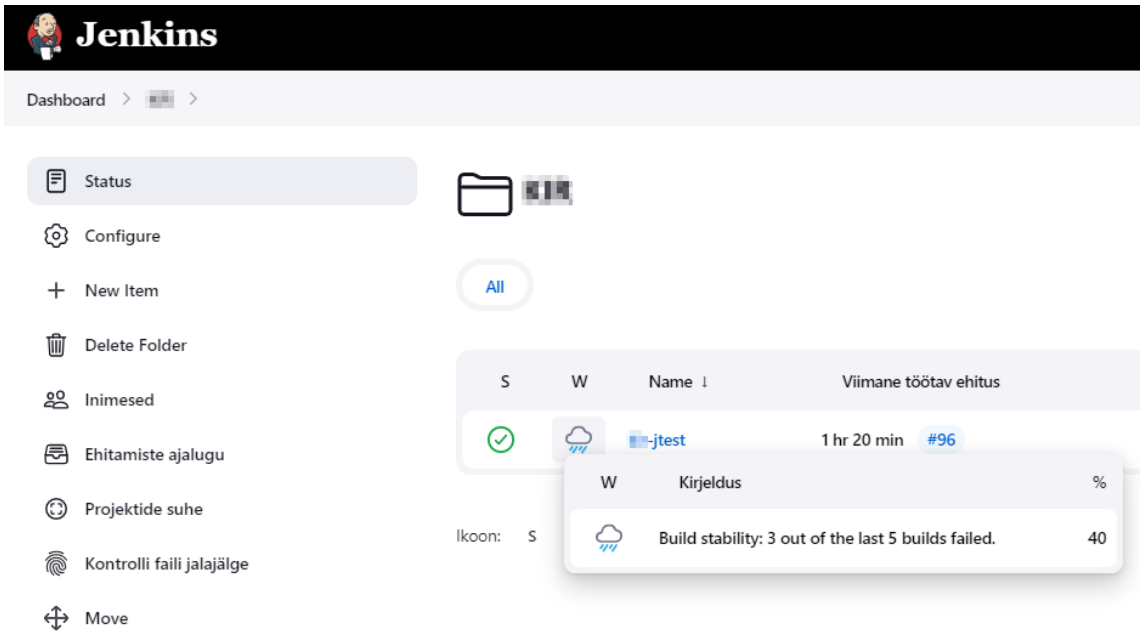
Joonis 14. Veebilehe komponentide loomine.

Kui vajalik keskkond ja kaust on loodud ning kaust ühiskasutusse antud, on eeldused IIS veebilehe loomiseks täidetud. HTTPS (*Hypertext Transfer Protocol Secure*) ühenduse kasutamiseks on veebilehe loomisel vaja leida serverist sertifikaat. Sertifikaadi kasutamiseks viidatakse sellele kasutades sertifikaadi unikaalset sõrmejälge. Kasutades eelnevalt loodud keskkonda, kausta ning porti on võimalik veebileht luua (vt. joonis 15).

```
Start-IISCommitDelay
$TestSite = New-IISSite -Name "Veebileht" -BindingInformation
"*:10055:" -CertificateThumbPrint
"2AAE6C35C94FCFB415DBE95F408B9CE91EE846ED" -CertStoreLocation
"Cert:\MyLocalMachine\My" -PhysicalPath
"D:\Intepub\Virtual\Kaust.jtest" -Protocol https -SslFlag None -
Passthru
$TestSite.Applications["/"].ApplicationPoolName = "Keskkond (jtest)"
Stop-IISCommitDelay
```

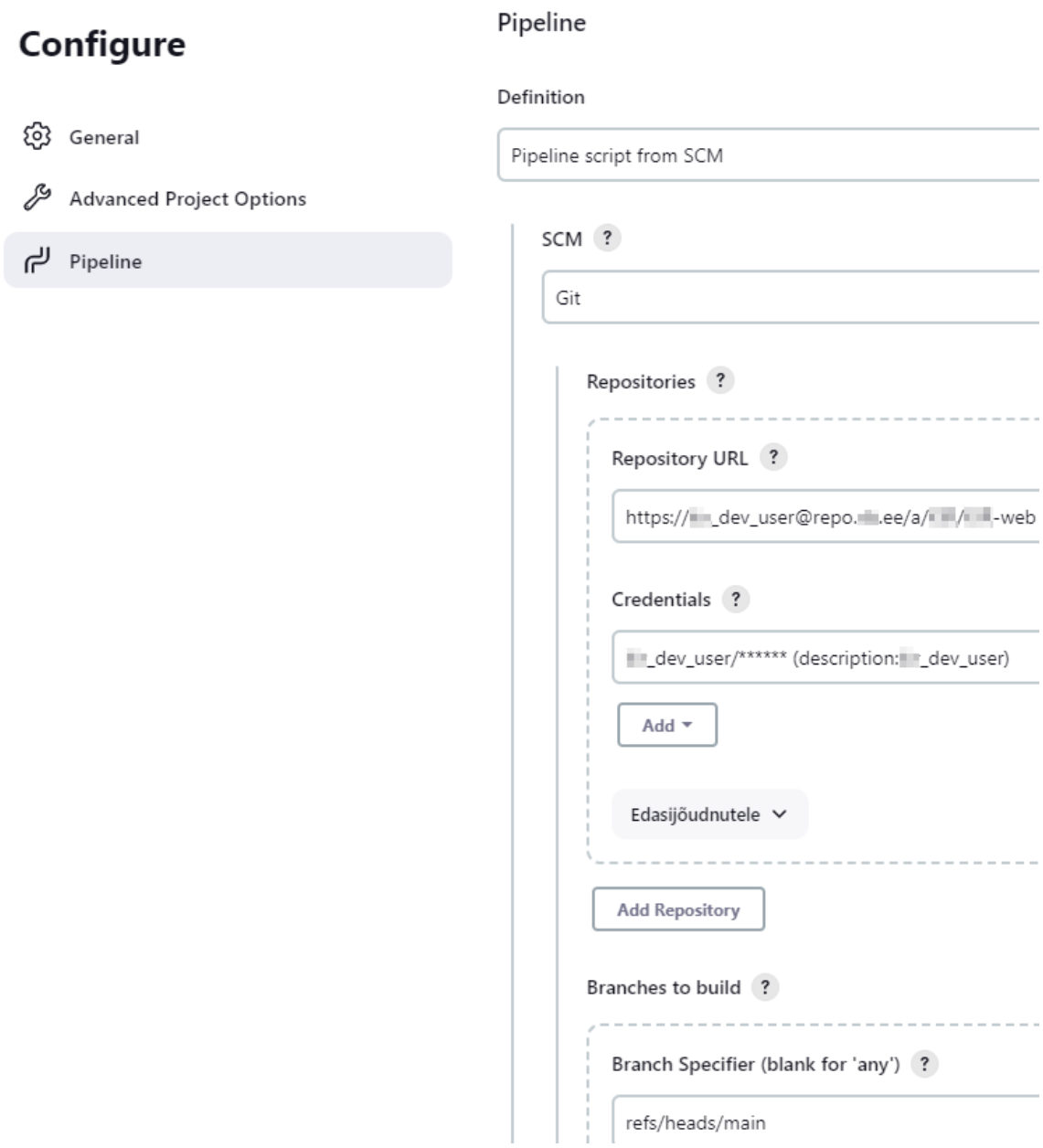
Joonis 15. Veebilehe loomine.

Järgnevalt luuakse Jenkins platvormil kaust, mille eesmärk on rakendusega seotud konveiereid hoiustada. Uut konveierit kirjeldatakse määrates sellele nimi, peale mida on võimalik edasi liikuda teiste täiendavate sammude kirjeldamiseni (vt. joonis 16). Peale esmase kirjelduse lisamise võimalust on võimalik valida tegur, mille tingimuste täitmise korral kirjeldatav konveier käivitatakse. Testimisperioodi ajal käivitatakse konveier manuaalselt kuniks konveier vigadeta toimib.



Joonis 16. Kaust ja konveier Jenkins platvormil.

Järgnevad sammud on konveierfaili spetsiifilised, nende abil määratakse konveierfaili asukoht. Võimalik on valida konveieri kirjeldamiseks Jenkins platvorm või lisada viide koodirepositooriumis asuvale konveierfailile. Kasutatakse koodirepositooriumi valikut, mille puhul valitakse kasutusel olev SCM (*Source Code Management*), milleks on Git. Peale koodirepositooriumi aadressi lisamist on vaja lisada kasutaja andmed, mille abil koodirepositooriumist lähtekood kätte saadakse. Andmeid lisades valitakse kasutaja andmete kirjeldamiseks kasutajanime ja parooli talletamise viis, mis võimaldab andmeid kasutada ainult selle kausta piires. Kui määratud kasutaja ühendus koodirepositooriumiga ebaõnnestub, siis kuvatakse sellesisuline veateade, mille abil on võimalik kontrollida, kas repositooriumiga on võimalik ühendada. Viimaste sammudena määratakse nii koodirepositooriumi peaharu, millele antud konveier rakendub, kui ka konveierfaili nimi ja asukoht repositooriumis (vt. joonis 17).



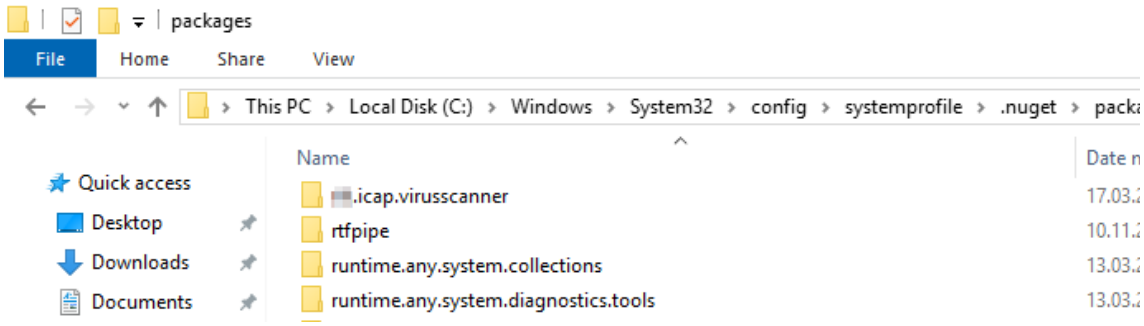
Joonis 17. Konveieri konfigureerimine Jenkins platvormil.

Seejärel luuakse Jenkinsfile nimeline fail rakenduse lähtekoodi juurkausta ning alustatakse selle kirjeldamisega. Esimesed testid hõlmavad kõikide konveieri sammude kohest kirjeldamist konveierfailis ning käivitamist, kuid tekib mitu probleemi. Kui TeamCity platvormil on eraldatud tööriista valik ning parameetrid, mille tulemusel platvorm need ise hiljem konveierit täites kombineerib, siis Jenkins konveierfailis on vaja täpsustada, millist käsurea programmi kasutatakse ning kasutatavad parameetrid lisatakse selle järgi ülakomade vahele. Sellise kirjelduse kasutamine lõppeb mitme süntaktilise veaga, mille tulemusel konveier ei käivitu.

Seejärel otsustatakse konveierfailis olevad sammud kustutada ning alustada sammude kirjeldamisega uuesti neid ükshaaval käivitades, veendumaks, et kirjeldamine on edukas ning sammu täitmisel ei teki vigu. Lähtudes tööriista nimetuse kasutamisest ning parameetrite lisamisest jutumärkide vahele laheneb esimeste sammude kirjeldamine ning täitmine probleemideta kuniks parameetrites ilmnevad mõningad kirjavahemärgid, millel on konveierfaili süntaksis spetsiaalne tähendus. Konveierfaili parameetrite kirjeldamisel kasutatud jutumärkide ning kurakriipsude kasutamise võimaldamiseks peab neid kombineerima üksiku längkriipsuga, mis tuleb paigutada märgi ette. Nii on võimalik kasutada neid märke nii, nagu need kirja kujul on.

4.1 Kompileerimise ja ettevalmistamise ning paigaldamise protsess

Lähtekoodi kompileerimine ning eesrakenduse jaoks failide ettevalmistamine toimub F# faili abil, kasutades konveierfaili eri etappe ja kirjeldades parasjagu vajaminevaid parameetreid. Konveieri etapis rakenduse jaoks vajalike teekide alla laadimisel ilmneb probleem ühe viirusetõrje teegiga. Probleem tekib kuna konveierit täitval kliendil ei ole puuduolevat teeki vahemälus ning ka selle alla laadimine ei õnnestu NuGet konfiguratsioonifailis määratud aadressidelt. Probleem laheneb kui puuduolev kaust vajalike failidega lisatakse kliendi vahemällu, teiste tarkvarateekide hulka (vt. joonis 18).



Joonis 18. Puuduolev teek vahemälus.

Konveierfailis on testide käivitamise sammul vajalik ka keskkonnamuutuja kasutamine. Keskkonnamuutuja kirjeldatakse konveierfailis märksõnaga *environment*, millele lisatakse võti-väärtus paar. Võti-väärtuspaari võtme kirjeldamiseks kasutatakse nimetust "DatabaseSettings__ConnectionString" ja väärtuseks arenduskeskkonna andmebaasi aadressi.

Probleem tekib ka .NET käsurea tööriista kasutades, kui on vaja projektis olevaid testid käivitada. Üksuste testid asuvad üheksas erinevas projektis ning nendes kasutatavad

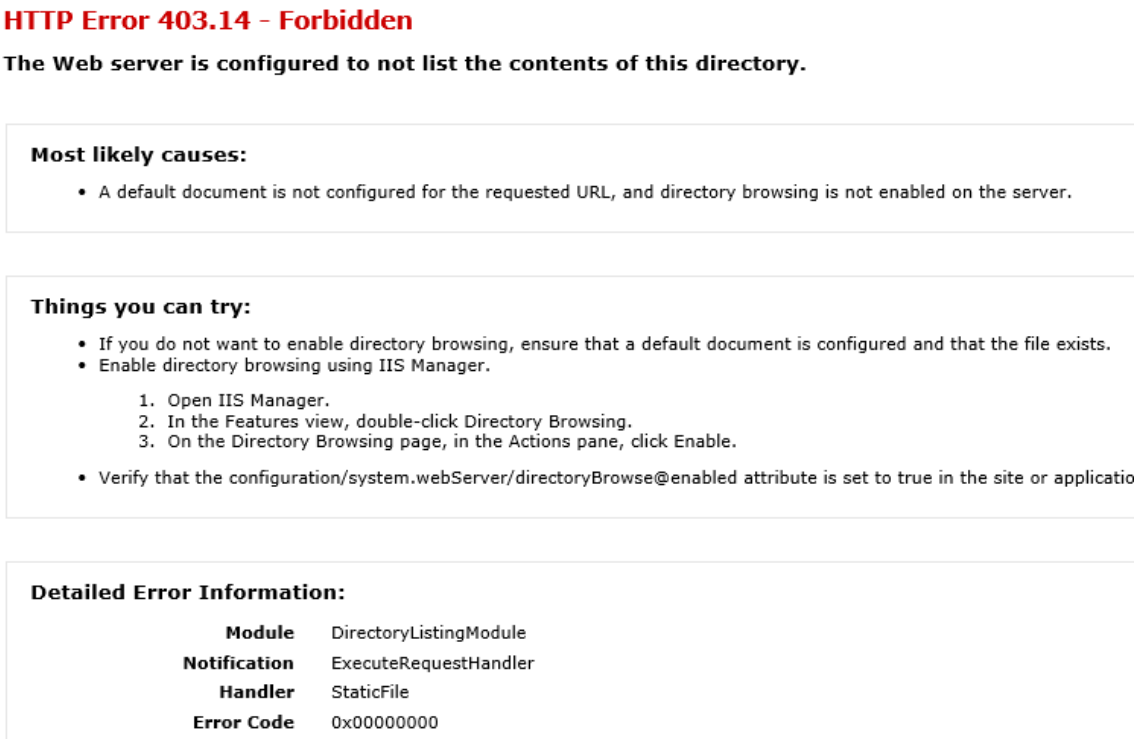
raamistikud jagunevad kaheks. Konveierfaili täitvas serveris on võimalik kasutada ainult uuemat programmi nimetusega „test“, kuid kasutusel olevas TeamCity platvormil on testide käivitamiseks kasutusel programmi vanem versioon nimetusega „vstest“. Vanema versiooni puhul on võimalik testitavate failide asukohta määrata kasutades metamärke (*wildcard*), mis võimaldavad määrata mitme faili asukoha, mille järel mitmes asukohas olevad testprojektide testid käivitatakse järk-järgult üksteise järel. Selliselt mitmele testifailile viitamine ei tööta uuema programmiga ning kuigi sellist viitamist ei loe programm valeks süntaksiks, õnnestub testifailidest käivitada ainult uuema .NET raamistikuga seotud projektid. Peale tiimi endise arhitektiga konsulteerimist selgub, et on vaja leida moodus kuidas teste ükshaaval käivitada. Üheksa erineva testprojekti jaoks üheksa erineva rea kirjutamine konveierfaili võimaldab testid edukalt käivitada, kuid võtab ebamõistlikult kaua aega ning seda ei ole mõistlik lisandunud ajakulu tõttu kasutada. Lahendusena kasutatakse PowerShell käsklust (*foreach*) mille abil itereeritakse testprojektide asukohad ükshaaval, andes võimaluse testprojektides olevad üksuste testid eraldi käivitada (vt. joonis 19).

```
steps {
      powershell 'foreach ($path in Get-ChildItem
.\\build\\tests\\*Testid) { dotnet test $path\\*.Testid.dll }'
}
```

Joonis 19. Testprojektide itereerimine ja testide käivitamine.

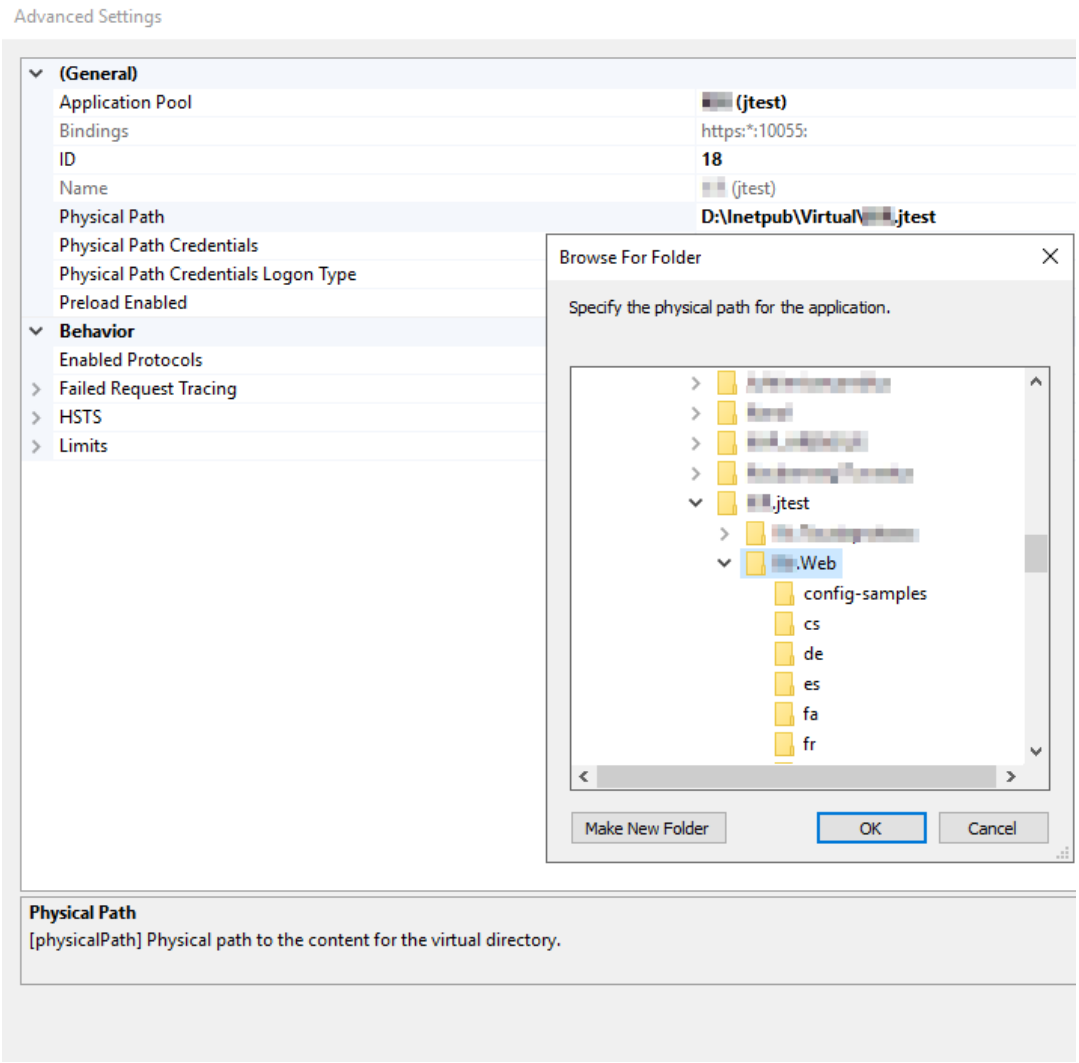
Probleem tekib ka rakenduse failide paigaldamisega IIS veebilehele. Kuigi kaust on jagatud, puuduvad konveierfaili täitval kasutajal kirjutamisõigused sinna failide kopeerimiseks. Peale kirjutamisõiguse lisamist õnnestub konveieri töö ilma vigadeta ning rakenduse failid kopeeritakse edukalt IIS veebilehe jaoks ette nähtud kausta.

Ka veebilehte avades tekib probleem, kuna loodud lehe puhul on märgitud sisu serveerimise asukohaks juurkaust, mille tulemusel üritatakse aadressile minnes kuvada kausta sisu (vt. joonis 20).



Joonis 20. Veateade kausta sisu kuvamisest.

See ei ole soovitud käitumine ning ka IIS veebiserver näitab sellekohast veateadet. Muutes IIS halduris veebilehe serveerimise kaustateed avaneb seejärel ka veebileht (vt. joonis 21).



Joonis 21. Rakenduse kaustatee määramine.

4.2 Andmebaasi migreerimine

Andmebaasi migreerimistööriista kasutus ei muutu ning selle kasutus on migreeritud F# keelsesse faili, mis lisaks tegeleb rakenduse lähtekoodi kompileerimise ning failide paigaldusega IIS veebilehele. Samuti jäävad alles viited PowerShell failile, mida kasutatakse SQL failide haldamiseks.

5 Tulemused ja edasiarenduse võimalused

Kõik etapid, mis on kasutuses platvormil TeamCity õnnestus üle tuua Jenkins platvormi konveierfaili (vt. lisa 2). Siiski tekkis üks väike erand, kus üksuste testimise etapis õnnestus üheksast testprojektist edukalt käivitada kaheksa testprojekti testid. Kui eelneva kaheksa projekti testide käivitamine võttis keskmiselt 13 sekundit projekti kohta, siis testprojekti käivitamist piirava vea parandamisel võib lisanduda ajakulu 10 kuni 20 sekundit.

Konveieri järk-järgult käivitamine ning vigade parandamine andis soovitud tulemuse peale 95 katset. Kui TeamCity platvormil on kaks eraldi konveierit, mis moodustavad ühe terviku, siis Jenkins platvormil on prototüübi raames valminud üks konveier, mis täidab kahes eelnevas konveieris kirjeldatud etapid. Jenkins platvormil võtab konveieri täitmine aega 30 minutit ja 49 sekundit, mis on eelneval platvormil aega võtnud 59 minutiga võrreldes pea poole kiirem (vt. joonis 22).

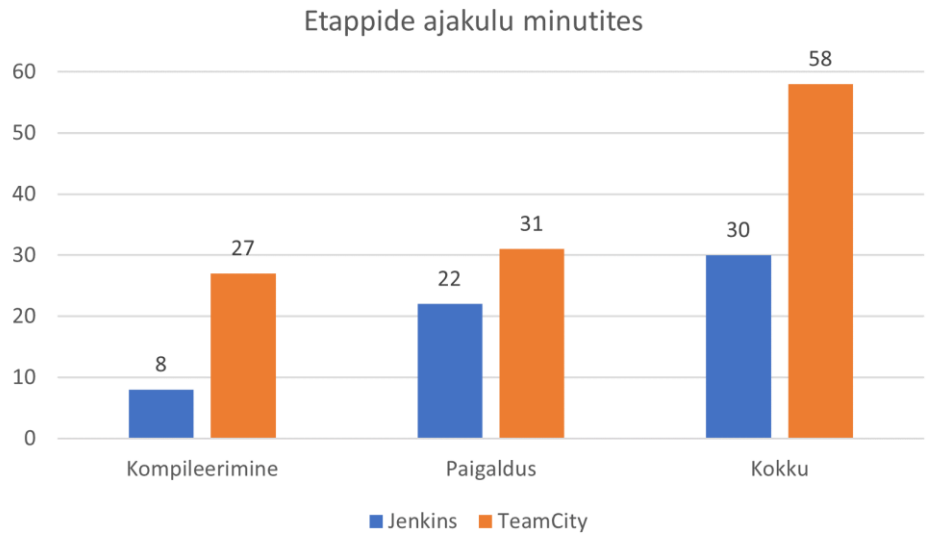
Joonis 22. Konveieri etapid Jenkins platvormil.

Kui võrrelda eelneva platvormi konveierite täitmisaega eraldi, siis esimene konveier, mis kompileerib testprojektide lähtekoodi ning käivitab testid, võtab aega keskmiselt 27 minutit. Jenkins platvormil võtavad samad etapid aega 8 minutit ja 55 sekundit. Mis on 69 protsendiline ajavõit eeldusel, et konveieri täitmine TeamCity platvormil eelnevalt ei ebaõnnestunud. Näitena on toodud üks olukord kus senise konveieri täitmine ebaõnnestus viiel järjestikusel korral ning kestis kokku 245 minutit (vt. joonis 23).

Success	No changes	30 Mar 23 09:09	51m:04s
Exit code 1 (Step: Build Deployment Packages (Command Line))	No changes	29 Mar 23 21:13	36m:06s
Exit code 1 (Step: Build Deployment Packages (Command Line))	No changes	29 Mar 23 18:34	38m:39s
Exit code 1 (Step: Build Deployment Packages (Command Line))	No changes	29 Mar 23 17:57	36m:31s
Exit code 1 (Step: Build Deployment Packages (Command Line))	No changes	29 Mar 23 16:54	42m:50s
Exit code 1 (Step: Build Deployment Packages (Command Line))	3 changes	29 Mar 23 16:05	39m:54s

Joonis 23. Ebaõnnestunud tulemusega konveierid.

Teine konveier senisel platvormil, mis kompileerib ülejäänud projektid, valmistab ette eesrakenduse lähtekoodi paigalduse jaoks ning lõpuks paigaldab rakenduse IIS veebilehele võtab aega keskmiselt 31 minutit. Jenkins platvormil käivitatakse konveieri kaks viimast etappi, mis täidavad samu ülesandeid ning võtavad aega 22 minutit ja 11 sekundit, võrreldes eelmise konveieri ajaga on ajavõit 28 protsenti. Võrreldes eelneva automatiseerimisplatvormi ja Jenkins platvormil käivitatud konveieri ajakulu on uuel platvormil käivitatud konveier märgatavalt kiirem (vt. joonis 24).



Joonis 24. Konveierite täitmisajad.

Konveieri käivitamine sõltub repositooriumis tehtud muudatuse tuvastamisest või teavituse saatmisest Jenkins platvormile. Pidev muudatuste pärimine on ajakulukas ning tarbib liialt ressursse. Sellest tulenevalt on üks võimalik täiendus koodihoidla poolt teavituse saatmine Jenkins platvormile juhul kui koodirepositooriumisse on lisatud muudatus. See vähendab pidevate päringute saatmist ning käivitab konveieri kui repositooriumisse on lisatud täiendusi.

Hilisemaks täienduseks on planeeritud muutujate kasutamine konveierfailis. Kirjeldamisel on kasutatud korduvaid väärtuseid, mille sisu on võimalik omistada ühele muutujale ning seeläbi, on muutujat kasutades võimalik vältida korduvaid kirjeid failis. Täiendav edasiarenduse võimalus on lahutada konveierfail kaheks eraldi failiks jagatud vastutusega. Esimeses failis kirjeldatud sammud piirduvad testide käivitamisega ning teine konveierfail käivitub ja täidab selles kirjeldatud sammud peale esimese konveieri edukat lõpetamist.

Teavituste saatmine konveieri töö tulemustest Teams rakendusele vajab Jenkins platvormile pistikprogrammi paigaldamist. Pistikprogrammi kasutamise eelduseks on Teamsi selle paigaldamise loa olemasolu ning veebikonksu (*webhook*) [98] genereerimine, mis seotakse Teams rakenduse vestluskanaliga. Hiljem saab kasutada veebikonksu aadressi konveierfailis kirjeldades, mis sorti ning mis sõnastusega teavitusi saadetakse. Teavitused saadetakse Teams kanalisse, mis veebikonksu luues määratud on.

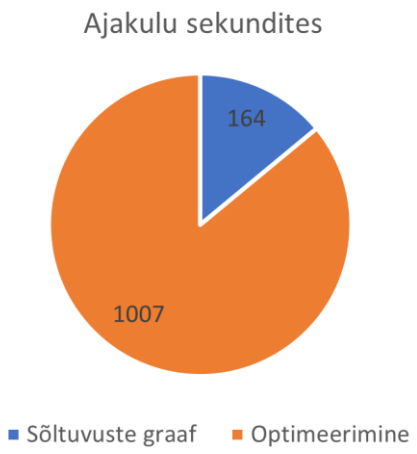
Praegusel konveieril on ühe sammuna kasutusel tööriist ESLint, mis kontrollib eesrakenduse failide programmikoodi leidmaks sealt probleemseid mustreid, kuid analoogne kontroll puudub tagarakenduse koodi puhul. Asutuses on olemas SonarQube server mis võimaldab kasutada programmikoodi kontrollimise tööriista. Antud tööriist on võimalik võtta kasutusele Jenkins platvormi pistikprogrammina, mille järel saab seda kasutada konveierfailis tagarakenduse programmikoodi kontrolliks. Võimaliku edasiarendusena plaanitakse välja selgitada kuidas antud tööriist konveieris kasutusele võetakse ning seejärel lisatakse koodi kontrollimise samm ka konveierfaili.

Kuna projekti koodirepositooriumis on mitu haru, vajavad ka need konveierit. Toetamaks mitut haru, peab loodud konveierfaili prototüübi kasutusele võtmiseks looma uue mitut haru toetava konveieri. Senine konveierfaili prototüüp eelmainitud täienduste lisamisel sobib uue konveieri arenduskeskkonna konveierfaili kirjeldamiseks. Mitme haru

toetamiseks on konveier võimalik luua kasutades asutuse poolt loodud konveierit, mis kasutab sisendandmetena programmikoodi senise repositooriumi nime ning kasutaja andmeid millega repositooriumiga ühendutakse. Konveieri kasutamise tulemusena luuakse Jenkins platvormile kaust, mis sisaldab mitut haru toetavat konveierit koos näidis konveierfailiga. Loodud konveierfailis on võimalik etappide kirjelduses määrata, millise koodirepositooriumi haru korral antud etapp tuleb läbida.

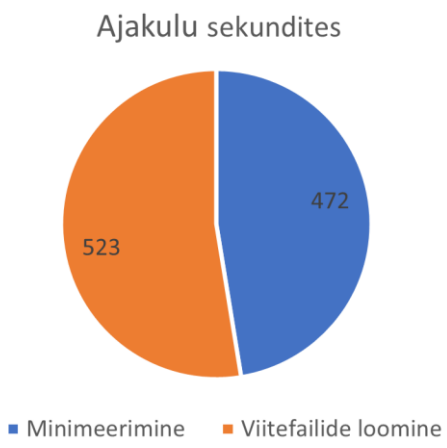
5.1 Rakenduse automatiseerimine

Kui andmebaasi muudatuste testimine ja rakendamine ning üksuste teste sisaldavad projektid kompileeritakse testimise sammu käigus varem, siis ülejäänud projektide lähtekoodi kompileerimine koos eesrakenduse failide ettevalmistamisega toimub hiljem. Kõik eelnevad sammud võtavad aega alla kahe minuti, kuid rakenduse projektide kompileerimine ning eesrakenduse jaoks failide ettevalmistamine võtab kokku aega 21 minutit. Pelgalt eesrakenduse failidega tegelemise peale kulub 20 minutit, kus luuakse sõltuvuste graaf ning tehakse optimeerimisi (vt. joonis 25).



Joonis 25. Üldine eesrakenduse ajakulu jaotus.

Enim aega kulub eesrakenduse etapile kus faile optimeeritakse, mille kestvuseks on 17 minutiline etapp erinevatest optimeerimistegevustest mis tehakse eesrakenduse failidega, kui neid veebilehitseja jaoks ette valmistatakse. Kaks ajakulukat sammu optimeerimisel on failide sisu minimeerimine ning viitefailide (*source maps*) [99] loomine, mis aitavad minimeeritud koodi kokku panna originaalis loodud koodiga (vt. joonis 26).



Joonis 26. Optimeerimissammude ajakulu.

Viitefaile kasutatakse, kuna minimeeritud programmikoodi on silumisel raske lugeda. Failide sisu minimeeritakse 8 minutit ja viitefaile luuakse ligilähedale 9 minutit.

Üheks võimalikuks edasiarenduseks on optimeerimisprotsessi kiirendamine, mis võimaldab rakenduse failid kiiremini paigalduse jaoks ette valmistada, lühendades kogu protsessile kuluvat aega. Nii failide sisu minimeerimiseks, kui ka viitefailide loomiseks kasutatakse pistikprogramme, mida on võimalik Webpack konfiguratsioonifailis vahetada mõne teise vastu või muuta olemasoleva seadistust. Selliselt on võimalik proovida teisi programme või muutes seadistusi leida optimaalseim ning kiireim lahendus optimeerimisprotsessi kiirendamiseks.

5.2 Andmebaasi migreerimistööriist

SQL skripte rakendatakse arenduskeskkonna puhul konveieri töö käigus automaatselt, kuid teiste keskkondade puhul edastatakse rakenduse täitmisprogramm koos skriptidega süsteemiadministraatorile, kes need skriptid käsitsi rakendab. Üks võimalik edasiarendus on SQL skriptide rakendamine igas keskkonnas automaatselt.

SQL skriptide automaatset rakendamist ilma süsteemiadministraatori juuresolekuta loetakse võimalike probleemide tekkimise tõttu siiski liiga riskantseks ning seda hetkel ei soosita.

6 Kokkuvõte

Lõputöö eesmärk oli luua konveieri prototüüp, mille täitmiskiirus on kiirem kui senine. Lisaks oli eesmärk lihtsustada konveieri kirjelduse lugemist ning sellest arusaamist ja parandada seadmete jõudlust, millel konveier täidetakse.

Töös uuriti ja hinnati võimalike automatiseerimisplatvorme ja protsessis kasutavate komponentide sobivust arenduses oleva hajussüsteemiga, eesmärgiga parandada automatiseerimisprotsessis esinevaid probleeme ja lühendada arenduse tagasiside saamisele kuluvat aega. Lisaks käsitleti platvormide eripärasid ja komponentide kasutusvõimalusi. Töö teoreetilises osas läbiviidud analüüs võib osutuda kasulikuks spetsialistidele, kes soovivad alustada või täiustada rakenduse kompileerimis- või paigaldusprotsessi automatiseerimist. Analüüsi tulemused võivad aidata kaasa automatiseerimisplatvormi kasutuselevõtu planeerimisel ning muudatuste elluviimisel.

Võttes eeskujuks pidevintegratsiooni praktikad, loodi analüüsi tulemusel sobivaimaks osutunud automatiseerimisplatvormile uus konveier, mis vastas asutuses kehtestatud nõuetele ning täitis senise automatiseerimisprotsessi kohustusi optimeeritud kujul.

Loodud konveierfail võimaldab senist konveierit täita kiiremini kui eelneval platvormil. Lisaks on seda lihtsam edaspidi hallata, kuna kõik kirjeldatud sammud ja etapid on ühes failis. Ühe faili kasutamine konveieri kirjeldamiseks võimaldab kompileerimis- ja paigaldusprotsessist kiiremini ja lihtsamini aru saada. Ka on võimalik uuel platvormil kasutada paremat riistvara kui eelneval, mis parandab jõudlust. Uut platvormi kasutades on võimalik saada ka asutuse sisest tuge.

Tulevikus on võimalik konveierit täiendada toetamaks mitme koodirepositooriumi haru kasutamist. Lihtsustada veelgi konveierite täitmist jagades vastutusalad kaheks erinevaks failiks ning täiendada konveieri töö tulemuste teavitamisprotsessi.

Kasutatud kirjandus

[1] „What is SDLC? Understand the Software Development Life Cycle,“ [Võrgumaterjal]. Available: https://phoenixnap.com/blog/software-development-life-cycle. [Kasutatud 16 02 2023].

[2] „Trunk Based Development,“ [Võrgumaterjal]. Available: https://trunkbaseddevelopment.com/. [Kasutatud 18 02 2023].

[3] R. Leszko, Continuous Delivery with Docker and Jenkins - Third Edition, Packt, 2022.

[4] „Vue.js,“ [Võrgumaterjal]. Available: https://v2.vuejs.org/. [Kasutatud 25 02 2023].

[5] „Other Versions - TeamCity,“ [Võrgumaterjal]. Available: https://www.jetbrains.com/teamcity/download/other.html. [Kasutatud 18 02 2023].

[6] „Windows Server 2008 R2 - Lifecycle,“ [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/lifecycle/products/windows-server-2008-r2. [Kasutatud 18 02 2023].

[7] „Tracking User Actions,“ [Võrgumaterjal]. Available: https://www.jetbrains.com/help/teamcity/2019.1/tracking-user-actions.html. [Kasutatud 18 02 2023].

[8] „Build Agent - TeamCity,“ [Võrgumaterjal]. Available: https://www.jetbrains.com/help/teamcity/2019.1/build-agent.html. [Kasutatud 25 02 2023].

[9] „Configuration as Code, Part 1: Getting Started with Kotlin DSL,“ [Võrgumaterjal]. Available: https://blog.jetbrains.com/teamcity/2019/03/configuration-as-code-part-1-getting-started-with-kotlin-dsl/. [Kasutatud 02 03 2023].

[10] M. Fowler, „Continuous Integration,“ [Võrgumaterjal]. Available: https://www.martinfowler.com/articles/continuousIntegration.html#PracticesOfContinuousIntegration. [Kasutatud 09 03 2023].

[11] P. Raspel, „Relatsiooniline andmebaas,“ [Võrgumaterjal]. Available: https://enos.itcollege.ee/~priit/1.%20Andmebaasid/1.%20Loengumaterjalid/. [Kasutatud 12 03 2023].

[12] „Common DB schema change mistakes,“ [Võrgumaterjal]. Available: https://postgres.ai/blog/20220525-common-db-schema-change-mistakes#three-categories-of-db-migration-mistakes. [Kasutatud 08 05 2023].

[13] „DevOps tech: Database change management,“ [Võrgumaterjal]. Available: https://cloud.google.com/architecture/devops/devops-tech-database-change-management. [Kasutatud 12 03 2023].

[14] „Choose an appropriate migration type,“ [Võrgumaterjal]. Available: https://docs.gitlab.com/ee/development/migration_style_guide.html#choose-an-appropriate-migration-type. [Kasutatud 08 05 2023].

[15] „How to Handle Database Schema Change?,“ [Võrgumaterjal]. Available: https://www.bytebase.com/blog/how-to-handle-database-schema-change. [Kasutatud 14 05 2023].

[16] T. Chen, „How to Handle Database Schema Change?,“ [Võrgumaterjal]. Available: https://www.bytebase.com/blog/how-to-handle-database-schema-change. [Kasutatud 13 03 2023].

[17] D. Sato, „ParallelChange,“ [Võrgumaterjal]. Available: https://www.martinfowler.com/bliki/ParallelChange.html. [Kasutatud 14 03 2023].

[18] „Database backups and rollbacks,“ [Võrgumaterjal]. Available: https://octopus.com/docs/deployments/databases/common-patterns/backups-rollbacks#making-database-changes-backwards-compatible. [Kasutatud 14 03 2023].

[19] „Software Architecture Types: Major Usage And Statistics,“ [Võrgumaterjal]. Available: https://incora.software/insights/software-architecture-types. [Kasutatud 30 04 2023].

[20] „Jetbrains Corporate Overview,“ [Võrgumaterjal]. Available: https://resources.jetbrains.com/storage/products/jetbrains/docs/corporate-overview/en-us/jetbrains_corporate_overview.pdf. [Kasutatud 30 04 2023].

[21] „Buy TeamCity,“ [Võrgumaterjal]. Available: https://www.jetbrains.com/teamcity/buy/#on-premises. [Kasutatud 30 04 2023].

[22] „Jenkins Github,“ [Võrgumaterjal]. Available: https://github.com/jenkinsci/jenkins. [Kasutatud 30 04 2023].

[23] „Donate to Jenkins,“ [Võrgumaterjal]. Available: https://www.jenkins.io/donate/#why-donate. [Kasutatud 30 04 2023].

[24] „GoCD Github,“ [Võrgumaterjal]. Available: https://github.com/gocd/gocd. [Kasutatud 30 04 2023].

[25] „gocd,“ [Võrgumaterjal]. Available: https://github.com/gocd/gocd. [Kasutatud 19 03 2023].

[26] „Drone Github,“ [Võrgumaterjal]. Available: https://github.com/harness/drone. [Kasutatud 30 04 2023].

[27] „Harness Acquires Continuous Integration Pioneer Drone.io and Commits to Open Source,“ [Võrgumaterjal]. Available: https://www.prnewswire.com/news-releases/harness-acquires-continuous-integration-pioneer-droneio-and-commits-to-open-source-301106473.html. [Kasutatud 19 03 2023].

[28] „Advanced installation of GoCD server using zip installer,“ [Võrgumaterjal]. Available: https://docs.gocd.org/current/installation/install/server/zip.html. [Kasutatud 09 05 2023].

[29] „Deploying Jenkins in public cloud,“ [Võrgumaterjal]. Available: https://www.jenkins.io/download/#deploying-jenkins-in-public-cloud. [Kasutatud 09 05 2023].

[30] „TeamCity Other Versions,“ [Võrgumaterjal]. Available: https://www.jetbrains.com/teamcity/download/other.html. [Kasutatud 30 04 2023].

[31] „Jenkins Windows installer,“ [Võrgumaterjal]. Available: https://www.jenkins.io/download/thank-you-downloading-windows-installer-stable/. [Kasutatud 30 04 2023].

[32] „GoCD Windows,“ [Võrgumaterjal]. Available: https://www.gocd.org/download/#windows. [Kasutatud 30 04 2023].

[33] „Server Installation,“ [Võrgumaterjal]. Available: https://docs.drone.io/server/overview/#server-installation. [Kasutatud 30 04 2023].

[34] „Jenkins download,“ [Võrgumaterjal]. Available: https://www.jenkins.io/download/. [Kasutatud 30 04 2023].

[35] „GoCD Debian,“ [Võrgumaterjal]. Available: https://www.gocd.org/download/#debian. [Kasutatud 30 04 2023].

[36] „macOS Installers for Jenkins LTS,“ [Võrgumaterjal]. Available: https://www.jenkins.io/download/lts/macos/. [Kasutatud 30 04 2023].

[37] „GoCD OSX,“ [Võrgumaterjal]. Available: https://www.gocd.org/download/#osx. [Kasutatud 30 04 2023].

[38] „Install .NET Framework for developers,“ [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/dotnet/framework/install/guide-for-developers. [Kasutatud 30 04 2023].

[39] „13 Key Features Every Modern CI/CD Tool Should Contain,“ [Võrgumaterjal]. Available: https://codefresh.io/blog/15588/. [Kasutatud 30 04 2023].

[40] „TeamCity pistikprogrammid,“ [Võrgumaterjal]. Available: https://plugins.jetbrains.com/search?products=teamcity. [Kasutatud 30 04 2023].

[41] „Jenkins plugins,“ [Võrgumaterjal]. Available: https://plugins.jenkins.io/ui/search/. [Kasutatud 30 04 2023].

[42] „GoCD plugins,“ [Võrgumaterjal]. Available: https://www.gocd.org/plugins/. [Kasutatud 30 04 2023].

[43] „Drone plugins,“ [Võrgumaterjal]. Available: https://plugins.drone.io/. [Kasutatud 30 04 2023].

[44] „Outlook Item (.msg) File Format,“ [Võrgumaterjal]. Available: https://interoperability.blob.core.windows.net/files/MS-OXMSG/%5bMS-OXMSG%5d.pdf. [Kasutatud 28 03 2023].

[45] „Office 365 Connector,“ [Võrgumaterjal]. Available: https://plugins.jenkins.io/Office-365-Connector/. [Kasutatud 30 04 2023].

[46] „Remote access to your Jenkins using REST API,“ [Võrgumaterjal]. Available: https://medium.com/@rathourarvi/remote-access-to-your-jenkins-using-rest-api-3d0c0bdb48a. [Kasutatud 09 05 2023].

[47] „How we used parallel CI/CD jobs to increase our productivity,“ [Võrgumaterjal]. Available: https://about.gitlab.com/blog/2021/01/20/using-run-parallel-jobs/. [Kasutatud 01 04 2023].

[48] „Build Chain DSL Extension,“ [Võrgumaterjal]. Available: https://www.jetbrains.com/help/teamcity/kotlin-dsl.html#Build+Chain+DSL+Extension. [Kasutatud 01 04 2023].

[49] „Parallel,“ [Võrgumaterjal]. Available: https://www.jenkins.io/doc/book/pipeline/syntax/#parallel. [Kasutatud 30 04 2023].

[50] „Matrix,“ [Võrgumaterjal]. Available: https://www.jenkins.io/doc/book/pipeline/syntax/#declarative-matrix. [Kasutatud 30 04 2023].

[51] „Concepts in GoCD - Stage,“ [Võrgumaterjal]. Available: https://docs.gocd.org/current/introduction/concepts_in_go.html#stage. [Kasutatud 02 04 2023].

[52] „Parallelism,“ [Võrgumaterjal]. Available: https://docs.drone.io/pipeline/exec/syntax/parallelism/. [Kasutatud 02 04 2023].

[53] „Parallel CI – Comparing CI Systems Parallelization,“ [Võrgumaterjal]. Available: https://www.incredibuild.com/blog/parallel-ci-comparing-ci-systems-parallelization. [Kasutatud 30 04 2023].

[54] „8 Build Automation Tools Open Source,“ [Võrgumaterjal]. Available: https://medium.datadriveninvestor.com/8-build-automation-tools-open-source-1a505ba9e19f. [Kasutatud 01 05 2023].

[55] „MSBuild or NAnt?,“ [Võrgumaterjal]. Available: https://scottdorman.blog/2007/12/24/msbuild-or-nant/. [Kasutatud 01 05 2023].

[56] „Apache .NET Ant Library,“ [Võrgumaterjal]. Available: https://ant.apache.org/antlibs/dotnet/. [Kasutatud 01 05 2023].

[57] „NAnt,“ [Võrgumaterjal]. Available: https://nant.sourceforge.net/. [Kasutatud 02 04 2023].

[58] „Boo Github,“ [Võrgumaterjal]. Available: https://github.com/boo-lang/boo#building. [Kasutatud 01 05 2023].

[59] „MSBuild,“ Microsoft, [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/visualstudio/msbuild/msbuild?view=vs-2022. [Kasutatud 02 04 2023].

[60] „Dotnet command,“ [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/dotnet/core/tools/dotnet. [Kasutatud 01 05 2023].

[61] „Dotnet publish,“ [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/dotnet/core/tools/dotnet-publish. [Kasutatud 01 05 2023].

[62] „Dotnet publish options,“ [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/dotnet/core/tools/dotnet-publish#options. [Kasutatud 01 05 2023].

[63] „.NET Build Automation,“ [Võrgumaterjal]. Available: https://dotnet.libhunt.com/categories/1777-build-automation. [Kasutatud 01 05 2023].

[64] „Nuke,“ [Võrgumaterjal]. Available: https://nuke.build/. [Kasutatud 01 05 2023].

[65] „FAKE,“ [Võrgumaterjal]. Available: https://fake.build/. [Kasutatud 01 05 2023].

[66] „PSake Github,“ [Võrgumaterjal]. Available: https://github.com/psake/psake. [Kasutatud 01 05 2023].

[67] „Project Structure,“ [Võrgumaterjal]. Available: https://nuke.build/docs/getting-started/setup/#project-structure. [Kasutatud 01 05 2023].

[68] „Build.cs & Build.fsx regex GitHub,“ [Võrgumaterjal]. Available: https://github.com/search?q=path%3A%2F%28%5E%7C%5C%2F%29build%5C.cs%24%7C%28%5E%7C%5C%2F%29build%5C.fsx%24%2F&type=code. [Kasutatud 01 05 2023].

[69] „Passing Values through the Command-Line,“ [Võrgumaterjal]. Available: https://nuke.build/docs/fundamentals/parameters/#passing-values-through-the-command-line. [Kasutatud 01 05 2023].

[70] „Fake.IO Namespace,“ [Võrgumaterjal]. Available: https://fake.build/reference/fake-io.html. [Kasutatud 01 05 2023].

[71] „NUKE Fluent API,“ [Võrgumaterjal]. Available: https://nuke.build/docs/common/cli-tools/#fluent-api. [Kasutatud 01 05 2023].

[72] „Post development,“ [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools/Deployment#post_development. [Kasutatud 07 04 2023].

[73] „A background on modules,“ [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules#a_background_on_modules. [Kasutatud 07 04 2023].

[74] „Code splitting,“ [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Code_splitting. [Kasutatud 07 04 2023].

[75] „Minification,“ [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Minification. [Kasutatud 07 04 2023].

[76] [Võrgumaterjal]. Available: https://www.smashingmagazine.com/2022/06/future-frontend-build-tools/. [Kasutatud 07 04 2023].

[77] „Vite v1.0.0-rc.10 Githubis,“ [Võrgumaterjal]. Available: https://github.com/vitejs/vite/tags?after=v1.0.0-rc.10. [Kasutatud 13 05 2023].

[78] F. Rappl, Modern Frontend Development with Node.js, Packt, 2022.

[79] „What is Pipeline as Code, and How Can You Leverage It?,“ [Võrgumaterjal]. Available: https://www.harness.io/blog/pipeline-as-code. [Kasutatud 09 05 2023].

[80] „What is pipeline as code?,“ [Võrgumaterjal]. Available: https://about.gitlab.com/topics/ci-cd/pipeline-as-code/. [Kasutatud 09 05 2023].

[81] [Võrgumaterjal]. Available: https://www.jenkins.io/doc/book/pipeline/syntax/#declarative-pipeline. [Kasutatud 08 04 2023].

[82] „YAML Syntax,“ [Võrgumaterjal]. Available: https://www.redhat.com/en/topics/automation/what-is-yaml#yaml-syntax. [Kasutatud 08 04 2023].

[83] „Triggers Overview,“ [Võrgumaterjal]. Available: https://spinnaker.io/docs/guides/user/pipeline/triggers/. [Kasutatud 10 05 2023].

[84] „The Next 11 Things You Should Do For CI/CD Pipeline Optimization,“ [Võrgumaterjal]. Available: https://hackernoon.com/the-next-11-things-you-should-do-for-cicd-pipeline-optimization-tb1a3wse. [Kasutatud 10 05 2023].

[85] „Understanding upstream and downstream jobs,“ [Võrgumaterjal]. Available: https://www.oreilly.com/library/view/jenkins-2x-continuous/9781788297943/af3de8e1-f3e9-4089-8e03-609cef1c0444.xhtml. [Kasutatud 10 05 2023].

[86] „Supported Databases,“ [Võrgumaterjal]. Available: https://dbup.readthedocs.io/en/latest/supported-databases/. [Kasutatud 10 05 2023].

[87] „SQL SERVER,“ [Võrgumaterjal]. Available: https://evolve-db.netlify.app/requirements/sqlserver/. [Kasutatud 10 05 2023].

[88] „grate supports the following DMBS’s,“ [Võrgumaterjal]. Available: https://erikbra.github.io/grate/#grate-supports-the-following-dmbss. [Kasutatud 10 05 2023].

[89] „Naming,“ [Võrgumaterjal]. Available: https://evolve-db.netlify.app/configuration/naming/. [Kasutatud 10 05 2023].

[90] „Options,“ [Võrgumaterjal]. Available: https://evolve-db.netlify.app/configuration/options/. [Kasutatud 10 05 2023].

[91] „FileSystemScriptProvider,“ [Võrgumaterjal]. Available: https://dbup.readthedocs.io/en/latest/more-info/script-providers/#filesystemscriptprovider. [Kasutatud 10 05 2023].

[92] „Evolve Metadata table,“ [Võrgumaterjal]. Available: https://evolve-db.netlify.app/concepts/#metadata-table. [Kasutatud 01 05 2023].

[93] „Grate Directory run order,“ [Võrgumaterjal]. Available: https://erikbra.github.io/grate/getting-started/#directory-run-order. [Kasutatud 01 05 2023].

[94] „DbUp configuration,“ [Võrgumaterjal]. Available: https://dbup.readthedocs.io/en/latest/usage/#deploying. [Kasutatud 01 05 2023].

[95] „Evolve Quick Start,“ [Võrgumaterjal]. Available: https://evolve-db.netlify.app/getting-started/lib/#quick-start. [Kasutatud 01 05 2023].

[96] „Grate configuration,“ [Võrgumaterjal]. Available: https://erikbra.github.io/grate/configuration-options/. [Kasutatud 01 05 2023].

[97] „Command-line shells,“ [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands?source=recommendations#command-line-shells. [Kasutatud 15 04 2023].

[98] „Create Incoming Webhooks,“ [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/microsoftteams/platform/webhooks-and-connectors/how-to/add-incoming-webhook?tabs=dotnet. [Kasutatud 18 04 2023].

[99] „Source Maps,“ [Võrgumaterjal]. Available: https://web.dev/source-maps/. [Kasutatud 21 04 2023].

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

Mina, Ivo Mäeoja

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Hajusrakenduse automaatse ehitus- ja paigaldusprotsessi arendus“, mille juhendaja on Kristiina Hakk
   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.

15.05.2023

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 – Konveierfail

```
pipeline {
  agent {
    node {
      label 'arendus[illegible]'
      }
    }
  options {
    disableConcurrentBuilds()
  }
  stages {
    stage('Restore tools') {
      steps {
        powershell 'dotnet tool restore'
      }
    }
    stage('Restore Packages (nuget)') {
      steps {
        powershell 'dotnet paket restore'
      }
    }
    stage('Restore Packages (npm)') {
      steps {
        powershell 'npm install'
      }
    }
    stage('Eslint') {
      steps {
        powershell 'npm run eslint'
      }
    }
    stage('Reset Database') {
      steps {
        powershell 'dotnet fake run build.fsx -e \"connectionString=Server=[illegible]; Database=[illegible]; User Id=[illegible]; Password=[illegible]; TrustServerCertificate=True;\" target ResetDatabase'
      }
    }
    stage('Build Test Projects') {
      steps {
        powershell 'dotnet fake run build.fsx -e \"environmentname=[illegible]\" -e \"deploytodirectory=[illegible]\" -e \"buildType=build\" target DeployTests'
      }
    }
    stage('Run Unit Tests') {
      environment {
        [illegible]_DatabaseSettings__ConnectionString = 'Data Source=[illegible]; Initial Catalog=[illegible]; User Id=[illegible]; Password=[illegible];'
      }
      steps {
        powershell 'foreach ($path in Get-ChildItem .\\build\\tests\\*Testid) { dotnet test $path\\*.Testid.dll }'
        //powershell 'dotnet test .\\build\\tests\\[illegible].SystemTest\\[illegible].SystemTest.dll'
      }
    }
    stage('Run JavaScript Unit Tests') {
      steps {
        powershell 'npm run test:headless'
      }
    }
    stage('Build Deployment Packages') {
      steps {
        powershell 'dotnet fake run build.fsx -e \"environmentname=[illegible]\" -e \"versionnumber=0.0.001\" -e \"longversionnumber=0.0.001\" target BuildArtifacts'
      }
    }
    stage('Deploy Packages') {
      steps {
        powershell 'dotnet fake run build.fsx -e \"deploytodirectory=\\\\[illegible].jtest\" -e \"environmentname=[illegible]\" -e \"versionnumber=0.0.001\" -e \"connectionString=Server=[illegible]; Database=[illegible]_jtest; User Id=[illegible]; Password=[illegible];\" target DeployApplicationFromArtifacts'
      }
    }
  }
}
```