TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Raimond Lume 179814IADB

TALTECH ÜLIÕPILASESINDUSE RAHASTUSPLATVORMI ARENDUS JÄTKUSUUTLIKU TARKVARAARENDUSE PÕHIMÕTTEID JÄRGIDES

bakalaureusetöö

Juhendaja: Märt Kalmo
MSc

Tallinn 2020

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: Raimond Lume

18.05.2020

Annotatsioon

Tarkvaraprojekti jooksul tuleb teha palju valikuid nii tarkvara arhitektuuri, tehnoloogiate ning ka toetavate süsteemide osas. Igal taolisel otsusel on otsene mõju projekti pikaajalisele jätkusuutlikkusele ning vähese kogemusega on sageli õiget valikut leida raske.

Käesoleva töö raames asub autor üle võtma tarkvaraprojekti olemasoleva koodibaasiga ning kirjeldab selle käigus esinenud erinevaid probleeme ning murekohti, mis muutsid uuele arendajale projektis arendusprotsessi tülikaks ning ebamugavaks. Taolisel kujul ei olnud projekti arendus jätkusuutlik ning tulevaste arenduste soodustamiseks tuli olukorda parandada.

Töö jooksul realiseerib autor lahendused, mis väldivad projekti üle võtmisel tekkinud probleemide esinemist tulevikus ning tagavad uuele arendajale projektis võimalikult lihtsa ning murevaba arenduskogemuse.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 27 leheküljel, 7 peatükki, 7 joonist.

Abstract

Development of TalTech Student Union's Financing Platform Following Sustainable Software Development Practices

A typical software project requires many different choices must be made in terms of software architecture, technologies and supporting systems. All such decisions have a direct impact on the long-term sustainability of the project and with little experience the right choices can often be hard to make.

The thesis is written around an existing incomplete software project, which the author starts to take over. Many different problems and issues arise, which make the initial development process cumbersome and inconvenient for the author, having just started in the project. As the same issues would need to be taken on by any new developer, the overall sustainability of the project did not seem promising.

To improve the situation, the author begins to analyze and implement different solutions to solve and prevent the numerous problems.

The thesis is in estonian and contains 27 pages of text, 7 chapters, 7 figures.

Lühendite ja mõistete sõnastik

API	*Application programming interface,* rakendusliides
CD	*Continuous deployment,* pidev evitamine
CI	*Continuous integration,* pidev integratsioon
CORS	*Cross-Origin Resource Sharing*
FTP	*File Transfer Protocol*
IDE	*Integrated Development Environment*
JSON	*JavaScript Object Notation*
OS	*Operating System,* operatsioonisüsteem
REST	*Representational State Transfer*
SPA	*Single Page Application*
SSH	*Secure Shell*
ÜE	Tallinna Tehnikaülikooli Üliõpilasesindus
YAML	*YAML Ain't Markup Language*

Sisukord

Jooniste loetelu

1 Sissejuhatus

Tarkvaraprojekti jooksul tuleb teha palju valikuid nii tarkvara arhitektuuri, tehnoloogiate ning ka toetavate süsteemide osas. Igal taolisel otsusel on otsene mõju projekti pikaajalisele jätkusuutlikkusele ning vähese kogemusega on sageli õiget valikut leida raske.

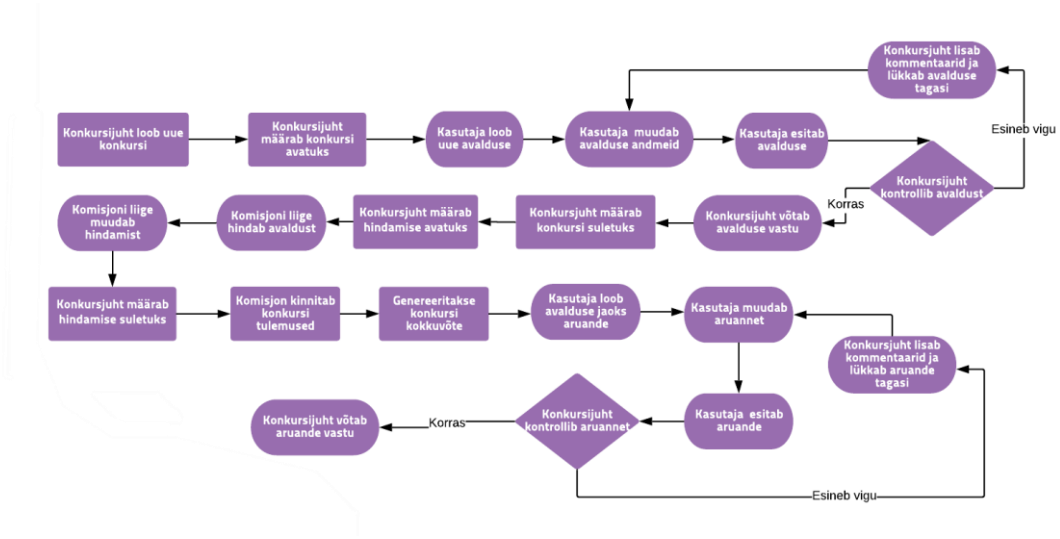
Käesoleva töö raames asub autor üle võtma tarkvaraprojekti olemasoleva koodibaasiga ning kirjeldab selle käigus esinenud erinevaid probleeme ning murekohti, mis muutsid uuele arendajale projektis arendusprotsessi tülikaks ning ebamugavaks. Taolisel kujul ei olnud projekti arendus jätkusuutlik ning tulevaste arenduste soodustamiseks tuli olukorda parandada.

Bakalaureusetöö analüüsi osas asub autor kaardistama ning lähemalt analüüsima esinenud probleeme ning nende põhjuseid ja pakub välja võimalikke lahendusi projekti olukorra parandamiseks. Autor analüüsib erinevaid praktikaid ning tehnoloogiaid, mis aitaksid püstitatud probleeme lahendada ning väldiksid taoliste murekohtade esinemist tulevikus.

Töö praktilises osas asub autor projekti jätkusuutlikkuse parandamiseks tehtud analüüsi põhjal realiseerima erinevaid lahendusi ning kirjeldab selle käigus esinenud katsumisi ning saadud õppetunde.

1.1 Projekti taust

Tallinna Tehnikaülikoolis võtab tudengielu rahastamine aset läbi üliõpilasesinduse (edaspidi ÜE). Tudengiorganisatsioonidel ning ka eraisikust tudengitel on võimalik erinevate konkursside raames taotleda rahastust oma tegevuste ja projektide läbi viimiseks. Rahastuse taotlemiseks vajalike avalduste täitmine on tülikas ning pikk protsess, samuti ei ole lihtne ülesanne rahastuskomisjonil esitatud tööde ülevaatus ning hindamine (v. t. Joonis 1). Täita tuleb suuri ja keerukaid Exceli vorme, mida on raske valideerida ning kontrollida. Kogu suhtlus konkursside ümber käib läbi meilivestluse.



Joonis 1. Rahastuskonkursi voog.

Aastal 2019. otsustas Tehnikaülikooli ÜE rahastuskonkursside lihtsamaks haldamiseks luua veebipõhise infosüsteemi. Töö otsustati tellida esialgu infotehnoloogia teaduskonna tudengilt Krõõt Grete Mändilt, kes plaanis infosüsteemi arenduse ümber valmis kirjutada ka oma bakalaureusetöö [1]. Infosüsteemi suure mahu ning ajapiirangute tõttu lepiti kokku, et infosüsteemi ehitatakse järk-järgult, seega Mändi lõputöö raames valmis vaid rakendusliides ning selle taga olev andmemudel.

Süsteemi reaalseks kasutamiseks on vajalik rakendusliidese peale ehitada ka kasutajaliides, mille raames pöörduti käesoleva töö autori poole.

1.2 Probleemi püstitus

Projekti üle võttes esines mitmeid murekohti. Esiteks oli arendamiseks vajaliku lokaalse keskkonna üles seadmine ebaselge ning tülikas protsess, milleks kulus autoril lõppkokkuvõttes terve päev. Olemasolev vastav juhend oli väga napisõnaline ning viitas tihti aegunud informatsioonile, mistõttu tuli hulgaliselt tekkinud probleemidele hakkama eraldi uusi lahendusi otsima.

Samuti oli tülirikas olemasolevas programmikoodis muudatuste tegemine – puudus tagasiside muudatuste mõjust kogu süsteemile, tihti põhjustas esialgu tunduv väike muudatus probleeme süsteemi teistes osades, mida ei osanud ette näha. Lisaks põhjustas raskusi projekti ebaselge struktuur, koodis oli raske navigeerida ning üles leida konkreetset faili, kus vajalikke muudatusi teostada.

Üliõpilasesinduse soovil pidi olema rakendus olema avalikult kättesaadav. Rakendusliidese loomisel antud ülesandega ei tegeletud ning projekti üle võtmise hetkel puudus lahendus nii rakendus- kui ka loodava veebipõhise kasutajaliidese avalikuks evitamiseks (ing. k. *deploy*) lõppkasutajatele.

1.3 Eesmärk

Projekti arendus olemasoleval kujul ei ole jätkusuutlik. Käesoleva töö eesmärk on analüüsida ning kaaluda erinevaid võimalusi senise lahenduse jätkusuutlikkuse parandamiseks. Samuti tuleb analüüsida erinevaid võimalusi nii rakendus- kui ka kasutajaliidese evitamiseks.

Praktilise osa eesmärk leida olemasoleva rakendusliidese peale ehitatava kasutajaliidese realiseerimisel lahendus, mis väldiks rakendusliidese seniste probleemide tekkimist ning tagaks ka uue süsteemi jätkusuutlikkuse.

1.4 Metoodika

Käesoleva töö metoodika on arendusuuring (ing. k. *Design-Based Research*).

Autor alustab probleemi defineerimisega, mille mõistmiseks analüüsib autor olemasoleva rakendusliidese arenduse jooksul tehtud otsuseid ning süsteemi puuduseid jätkusuutlikkuse aspektist.

Seejärel analüüsitakse ning kaalutakse erinevaid võimalikke variante püsitatud probleemide lahendamiseks.

Arendatakse ning realiseeritakse analüüsi osas välja töötatud lahendused ning ei ole välistatud, et arenduse käigus tekib omakorda probleeme, mis vajavad lahendamist.

Arenduse järel analüüsib autor tehtud tööd ning annab hinnangu loodud lahenduste mõjust püsititatud probleemidele ning sõnastab saavutatud tulemused.

2 Probleemi analüüs

2.1 Arenduskeskkonna üles seadmine

Rakendusliidese arendamiseks vajaliku arenduskeskkonna seadistamine oli pikk ning tülikas protsess, mis võttis kokkuvõttes ligi päeva. Projekti olemuse tõttu on tõenäoline, et rakendusliidesega peavad tulevikus tööd tegema ka teised arendajad, kellel tuleb samuti läbida sama ajamahukas protsess arenduskeskkonna seadistamisel. Sellises seisus ei tundu arendus jätkusuutlik.

2.1.1 Andmebaasilahenduse valimine

Suuremad probleemid esinesid seoses andmebaasiga, mida rakendusliides oma tööks vajab. Eelneva töö autor kasutas arendamise jooksul oma personaalses virtuaalserveris jooksvat andmebaasiserverit, mida käesoleva töö kirjutamise ajaks enam kasutada ei olnud võimalik, seega tuli leida uus lahendus.

Esimene võimalus oli lasta kliendil ehk üliõpilasesindusel rentida arenduse eesmärgil eraldi andmebaasiteenus. See oleks toonud juurde lisakulu, kuid oleks taganud arenduseks vajaliku andmebaasi olemasolu ka tulevastele arendajatele. Antud lahendus võib aga muutuda tülikaks olukorras, kus samal ajahetkel kasutavad andmebaasi korraga mitu arendajat. Andmemudelis tehtud muudatused tuleb koordineerida eelnevalt teiste arendajatega, et vältida konflikte andmete ning migratsioonidega. Lisaks tuleks leida lahendus, kuidas antud andmebaasi parameetreid tulevaste arendajatega turvaliselt jagada, et ei oleks vajadust neid eraldi eelnevatelt arendajatelt küsima hakata, kes ei pruugi olla alati kättesaadavad.

Teistsugune variant oleks igal arendajal kasutada arenduseks oma arvutis jooksvat lokaalset andmebaasiserverit. See aitaks vältida lisakulusid kliendile ning vähendab vajadust arendajate vaheliseks koordinatsiooniks, samuti on arendajatel lihtsam on töötada erinevates koodiharudes, kus igas harus võib olla erinevas seisus andmebaasimudel. Antud variandi puhul lisandub aga ajakulu ja keerukus arenduskeskkonna üles seadmisel. Tuleks leida lahendus, mis oleks arendajatele

võimalikult lihtne ja kiire üles seada. Seadistusele kuluva ajakulu vähendamiseks võiks olla andmebaasi konfiguratsioon arendajate vahel jagatav, samuti ei tohiks rolli mängida arendaja arvuti operatsioonisüsteemi valik ega muud taolised muutujad.

2.1.2 Delikaatsete andmete jagamine

Lisaks andmebaasi üles seadmisele tekkis probleeme ka rakendusliidese enda käivitamisega. Rakendusliidese autor oli koodivaramust turvalisuse kaalutlustel välja jätnud *Spring Boot*-i konfiguratsioonifaili (*application.properties*), kus on defineeritud erinevad parameetrid, mida rakenduse tööks vaja on, seal hulgas ka osade väliste API-de võtmed, mida otse läbi versioonihalduse jagada ei ole turvaline. Ilma antud failita ei olnud võimalik rakendust käivitada ning lõpuks tuli see rakendusliidese autoril eraldi käesoleva töö autorile saata.

Et taolist olukorda edaspidi vältida ning seeläbi ka arenduskeskkonna üles seadmisele kuluvat aega vähendada, tuleks leida lahendus, mis lubaks jagada erinevaid konfiguratsioonifaile, ilma et seal sisalduv delikaatne informatsioon versioonihalduses saadaval oleks. Selleks võib kasutada spetsiaalseid saladuste haldamiseks mõeldud raamistikke või varjata saladusi kasutades keskkonnamuutujaid. Leitav lahendus võiks olla integreeruv ka erinevate väliste teenustega, mida tulevikus on mõeldud kasutada, näiteks pideva integratsiooni teenused.

2.2 Sujuva arenduskogemuse tagamine

2.2.1 Arendajale automaatse tagasiside andmine

Projektis tööd alustades oli erinevatel põhjustel vajalik teha üle koodibaasi mitmeid muudatusi. Mitmel korral selgus alles hiljem, et tehtud modifikatsioonid põhjustasid erinevaid probleeme teistes rakenduste osades, mida autor uue arendajana projektis ei osanud kuidagi ette näha. Hulganisti mitteproduktiivset aega kulus taoliste „liblikaefektist“ põhjustatud probleemide lahendamiseks.

Vigade esinemine tarkvaraarenduses on paratamatu, olgu need arendaja inimlikud vead või mõne raamistiku uuendusest põhjustatud probleemid. Esimene samm vigadega võitlemiseks on nendest teadlik olla. Käesoleva projekti jätkusuutlikkuse parandamiseks on tarvis leida lahendus, mis annab arendajale esimesel võimalusel tagasisidet tema

muudatuste mõjust rakendusele ning teavitab ka erinevate vigade esinemisest süsteemist, lubades arendajal rohkem aega pühendada vigade parandamisele kui nende otsimisele.

Variant parema tagasiside tagamiseks oleks kasutada üksusteste. Rakendusliideses on suurel hulgal erinevaid äriloogilisi validatsioone, peamiselt avalduse esitamise faasis, mis aitavad ära hoida ebakorrektsete andmetega avalduste sattumist süsteemi [1]. Sellised validatsioonid on üks peamisi põhjuseid, miks üliõpilasesindus otsustas rahastuskonkursside haldamiseks infosüsteemi tellida. Taoliste kriitiliste komponentide korrektse valideerimise kontrollimiseks oleks hea võimalus kasutada üksusteste (ing. k. *unit test*), mis annaksid arendajale tagasiside ning meelekindlust, et tehtud muudatusel ei ole negatiivset mõju ärikriitiliselt oluliste komponentide tööle.

Lisaks komponendisiseste validatsioonidele on süsteemi töös äriloogiliselt tähtsal kohal ka komponentide vaheline suhtlus, mis täidavad süsteemi peamisi protsesse – konkursile avalduste esitamine, avalduse suunamine komisjonile hindamiseks ja avaldusele tagasiside andmine. Taolistes protsessides osalevad mitmed eri komponendid, mille omavahelise suhtluse korrektse toimimise kontrollimiseks võib kaaluda integratsiooniteste, mis annaksid arendajale tagasisidet, kui tema tehtud muudatuste tõttu on häiritud komponentide vaheline suhtlus või töö.

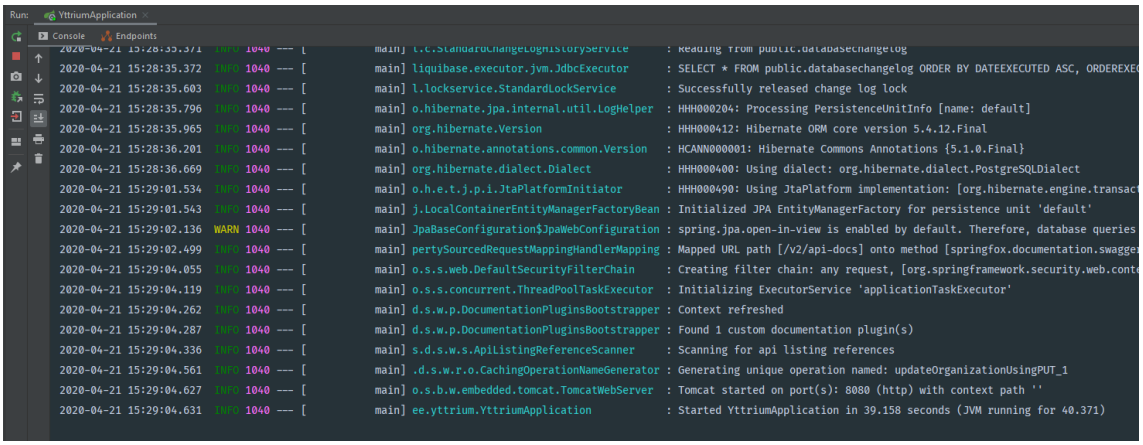
Automaattestide kirjutamine on lisanduv ajakulu, mis otseselt süsteemile funktsionaalsust ei lisa. Töö autori arvates testide kirjutamisele kulutatud aeg mõistlik investeering projekti üldisesse töökindlusesse ning jätkusuutlikkuse. Projekti olemuse tõttu ei pruugi olla alati võimalus arendajatel oma muudatuste kohta meeskonnaliikmetelt arvamust ja tagasiside küsida ning erinevad vead võivad jääda märkamata. Automaattestid oleksid sellises olukorras suureks abiks ning annaks arendajale meelekindlust, et tema muudatuste tõttu ei ole häiritud süsteemi peamine funktsionaalsus. Samuti on automaattestid uuele arendajale projektis ka hea dokumentatsioonivorm, läbi mille on võimalik saada kiire ülevaade mõne komponendi või protsessi tööst [2].

Kui automaatestide jooksutamine on protsess, mida arendaja peab ise läbi viima, siis autori kogemuse põhjal see tihti unustatakse ning testid ei täida oma eesmärki. Taolise olukorda vältimiseks tuleks leida lahendus, mis eemaldaks protsessist inimfaktori ning tagaks testide automaatse jooksutamise ja annaks arendajale koheselt tagasisidet, kui mõni test peaks tema muudatuste tõttu ebaõnnestuma.

2.2.2 Koodi silumise protsessi lihtsustamine

Uue arendajana projektis oli algul raske saada ülevaadet, kuidas suhtlevad omavahel erinevad süsteemi komponendid ning millisel ajahetkel teatud loogikat välja kutsutakse.

Olemasolevas rakendusliideses pakub taolist ülevaadet vaid *Spring Boot*-i sisse ehitatud süsteemilogi (Joonis 4) näol, mis annab vaikimisi hästi infot süsteemi enda protsesside ning esinenud süsteemivigade kohta. Süsteemilogist ei ole aga võimalik saada infot erinevate äriloogiliste protsesside voost (näiteks validatsiooni- või teenuskihis). Taoline informatsioon muudaks autori arvates lihtsamaks koodi silumise protsessi, andes arendajale rohkem informatsiooni andmete liikumise ning protsesside sammude kohta.



Joonis 2. Spring Boot vaikimisi süsteemilogi.

2.3 Rakenduse evitamine

Loodav veebipõhine rahastusplatvorm pidi olema avalikult kättesaadav. Rakendusliidest arendades antud nõude peale ei keskendutud, mistõttu puudus lahendus nii olemasoleva API kui ka loodava kasutajaliidese evitamiseks.

2.3.1 Lähtetingimused

Lahenduse leidmisel tuleb arvestada kasutatavate tehnoloogiatega:

- rakendusliides on Java-põhine rakendus, mis kasutab *Spring Boot* raamistikku
- andmebaasitehnoloogiana on kasutuses *PostgreSQL*
- kasutajaliideseks on plaanitud *Vue.js* raamistikuga ehitatud *Single Page Application*

Üliõpilasesindusel oli juba eelnevalt tellitud *Wavecom*[1] teenusepakkujalt veebimajutusteenus ning domeen *tipikas.ee*. Rahastusplatvormi evitamisel eelistas ÜE, et võimalusel kasutataks olemasolevaid teenuseid, kuid oli nõus ka vajadusel põhjendatud lisakuludega.

2.3.2 Pideva evitamise praktika võimaldamine

Autori arvates oleks käesoleva projekti puhul mõistlik järgida pideva evitamise ehk *continuous deployment* praktikat, mis tähendab võimalikult tihedast rakendus- ja kasutajaliidese automatiseeritud evitamist.

Praktika jälgimine muudaks rakenduse evitamise nii praegustele kui ka tulevastele arendajatele oluliselt lihtsamaks – uue versiooni evitamiseks piisaks vaid muudatuste sisse viimisest ning kogu ülejäänud protsess toimub automaatselt. Seetõttu puudub projektis ka edaspidi otsene vajadus administratiivsete pädevuste järele, mis lihtsustab ka uute arendajate leidmist projekti.

Küll aga pole mõistlik lahendus, kus iga arendaja tehtud muudatus jõuab otse lõppkasutajateni. Kaob ära võimalus uut funktsionaalsust eelnevalt testida ning selle kohta kliendilt tagasiside küsida. Seetõttu oleks autori arvates parem välja töötada lahendus, kus on võimalik pidevat evitamist kasutades üles seada kaks keskkonda – üks oleks mõeldud eelkõige rakenduse testimiseks ning kliendile uue funktsionaalsuse demonstreerimiseks, teine reaalseks kasutuseks lõppkasutajatele.

[1] https://wavecom.ee/

3 Tehnoloogiate analüüs

3.1 Andmebaasiserveri lahenduse valimine

Peatükis 2.1 tõi autor välja, et arenduskeskkonna üles seadmisele kuluva aja vähendamiseks tuleb leida kindel lahendus rakenduse arendamisel kasutatava andmebaasi olemasolu tagamiseks. Autor pakkus probleemi lahendamiseks esialgu välja kaks varianti: tellida arenduseks eraldi andmebaasiteenus või kasutada selleks lokaalset andmebaasiserverit. Antud variandid esitas autor ka kliendile, kes eelistas arenduskulude vähendamise eesmärgil viimast varianti.

Sellest tulenevalt oli tarvis leida lokaalse PostgreSQL andmebaasiserveri jaoks lahendus, mis rahuldaks probleemi analüüsis püstitatud nõudeid:

- ajakulu vähendamise eesmärgil peaks olema andmebaasi konfiguratsioon arendajate vahel jagatav
- lahendus peaks olema võimalikult universaalne, et vältida probleeme arendajate erinevate operatsioonisüsteemide või muude taoliste isikupäraste muutujatega

Levinum variant taolise probleemi lahendamiseks oleks kasutada PostgreSQL enda poolt pakutud ametlikke installereid[1], mis on saadaval kõikidele levinumatele operatsioonisüsteemidele. Installer paigaldab arvutisse PostgreSQL teenuse ning soovi korral ka erinevad haldustööriistad. Andmebaasiserveri konfigureerimine toimub installatsiooni käigus ning hilisemad muudatused seadistustes käivad läbi käsurea või vastavate graafiliste tööriistade. Antud variandi korral puudub mugav võimalus seadistuste jagamiseks teiste arendajatega ning samuti on ebamugavad versiooniuuendused, kus iga uue versiooni puhul tuleb alla laadida eraldi installer.

Teine ning autori arvates mugavam variant andmebaasi jooksutamiseks oleks kasutada *Docker Compose*-l põhinevat lahendust, mis lubab andmebaasi seadistust arendajate vahel jagada ning ei nõua eraldi andmebaasiserveri installeerimist, kasutades selle asemel ametlikku PostgreSQL konteinerit[2]. Kasutades *Docker Compose* teenust on võimalik

[1] https://www.postgresql.org/download/

[2] https://hub.docker.com/_/postgres

defineerida eraldi YAML formaadis konfiguratsioonifail, kus on võimalik seadistada erinevaid andmebaasi parameetrid ning antud faili on võimalik ka läbi versioonihalduse jagada. Eeldusel et arendajal on juba eelnevalt *Docker Compose* installeeritud, piisaks uuel arendajal projektis lokaalse andmebaasi kasutamiseks antud eelseadistatud *Docker Compose* faili jooksutamisest ning puudub vajadus andmebaasiserver eraldiseisvaks installeerimiseks ja seadistamiseks.

3.2 Delikaatsete andmete hoiustamine koodivaramus

Peatükis 2.1.2 tõi autor välja vajaduse lahenduse järgi, mis lubaks jagada erinevaid konfiguratsioonifaile, ilma et seal sisalduv delikaatne informatsioon versioonihalduses saadaval oleks.

Antud probleemi puhul oleks variant kasutada spetsiaalselt Git versioonihaldussüsteemis saladuste hoidmiseks loodud teenust avatud lähtekoodiga *git-crypt*, mis lubab kasutajal määrata kindlad failid koodivaramus, mis krüpteeritakse koodi üleslaadimisel ning dekrüpteeritakse allalaadimisel. Teenust on võimalik kasutada kahel viisil – asümmeetrilist või sümmeetrilist krüptograafiat kasutades. Asümmeetrilise variandi korral on tarvis iga arendaja eraldi autentida, kasutades selleks GPG[1] võtmeid. Sümmeetrilise variandi puhul kasutavad kõik arendajad failide dekrüpteerimiseks ühte võtit, mida tuleb turvaliselt meeskonna vahel jagada, kasutades selleks väliseid suhtluskanaleid [3].

git-crypt'i poolt krüpteeritud failide dekrüpteerimiseks peab olema arendajal teenus oma arvutis installitud. Käesoleva töö kirjutamise hetkel on teenus toetatud vaid Linuxi ja macOS-i operatsioonisüsteemidel, Windowsi tugi on alles arendamisel [4]. Seetõttu ei ole autori arvates mõistlik antud lahendust kasutada, arvestades et tarkvara viimane versioon (töö kirjutamise hetkel 0.6.0) väljastati aastal 2017 [5].

Teine võimalus konfiguratsioonifailide jagamiseks ilma delikaatseid andmeid avalikustamata oleks kasutada selleks keskkonnamuutujaid. Saladused salvestatakse rakendust jooksutava masina keskkonnamuutujatesse, mille viidatakse konfiguratsioonifailides lihttekstis saladuste asemel. See lahendus ei eemalda eraldi

[1] https://gnupg.org/

saladuste jagamise vajadust arendajate vahel, kuid võimaldab siiski turvaliselt konfiguratsioonifaile koodivaramus hoiustada. Samuti toetavad antud lahendust ka enamus CI/CD teenused, mistõttu ei tohiks võimalikud tulevased integratsioonid saladuste hoidmise osas probleeme tekitada.

3.3 Automaatne testide jooksutamine

Peatükis 2.2.1 tõi autor välja, et automaattestide jooksutamine muuta järjepidevamaks, tuleks antud ülesanne automatiseerida. Autori arvates oleks mõistlik antud probleemi lahendamiseks suure valiku tõttu kaaluda erinevaid pideva integratsiooni teenuseid.

Pidev integratsioon ehk *continuous integration* (edaspidi CI) on praktika, kus erinevate arendajate koodimuudatused integreeritakse ülejäänud koodibaasiga võimalikult tihti, tavaliselt mitu korda päevas. CI protsessi jooksul kontrollitakse kõiki uusi muudatusi enne integreerumist, et vältida vigase koodi jõudmist jagatud koodibaasi. See tähendab enamasti automaattestide jooksutamist ning rakenduse ehitamist iga uue muudatuse korral [6].

Rakendusliidese ehitamisel tehtud otsused ning projekti olemus seavad võimalikule lahendusele teatud piirangud:

- rakendusliidese koodi hoitakse GitHub versioonihaldusplatvormil, leitav CI lahendus peaks olema platvormiga ühilduv
- kliendi soovil ei tohi tuua leitav lahendus juurde lisakulusid
- lahendus peab toetama Java ning Gradle tehnoloogiaid, mida kasutakse rakenduse ehitamiseks ning testide jooksutamiseks

Turul enim kasutatav pideva integratsiooni tööriist on Jenkins [7]. Tegemist on avatud lähtekoodi ning tasuta teenusega, millel on hea tugi ka käesolevas projektis kasutatud tehnoloogiatele, nimel Java ning Gradle. Samuti ei teki probleeme rakendusliidese kasutatud koodivaramuga ühildumisel.

Käesoleva projekti puhul muudab Jenkinsi kasutuse raskemaks asjaolu, et platvorm on mõeldud kasutamiseks *on-premise* ehk enda riistvara peal, milleks puudus antud juhul võimalus. Variant oli Jenkinsi jooksutamiseks rentida eraldi virtuaalserver, kuid see oleks toonud sisse lisakulu nii finantsilises kui ajalises mõttes.

Autori arvates oli parem variant kasutada teenust nimega *GitHub Actions*. Teenus toetab Gradle raamistikku ning seda on võimalik suletud lähtekoodiga koodivaramute puhul kasutada tasuta kuni 2000 minutit kuus, mis antud väikese mahuga projekti puhul on igati piisav [8]. Antud variant on autori arvates etem, kuna rakendusliidese kood oli juba eelnevalt hoiustatud *GitHub* platvormil ning seetõttu oleks CI üles seadmine relatiivselt lihtne protsess. Samuti on oleks testide tagasiside nähtaval samal veebiplatvormil, mida projektis juba eelnevalt versioonihalduseks kasutati.

3.4 Logide tasemepõhine kasutamine

Olemasoleva lahenduse logide kasulikkuse parandamiseks on kõige autori arvates lihtsam kasutada *Spring Boot* raamistikuga (täpsemalt *spring-boot-starter-web* pakett) kaasa tulevat *Logback*[1] raamistikku. *Logback* täidab *SLF4J*[2]API-t, mis defineerib ära viis peamist logimise taset: *TRACE, DEBUG, INFO, WARN, ERROR.* Antud tasemete järgi on hiljem vastavalt vajadusele võimalik logisid filtreerida. Arendamise jooksul tulevad kasuks detailsemad tasemed, mis annavad parema ülevaate rakenduse töö käigust ning aitavad esinenud vigu diagnoosida. Juba kasutuses oleva rakenduse puhul pakuvad rohkem väärtust kõrgemad tasemed, kus liiane info pigem segab ning rohkem tähelepanu vajavad vea- ja hoiatusteated.

Tasemepõhise logimise puhul peaks olema tasemete kasutamise põhimõtted meeskonna seas üheselt kokku lepitud. Kui samu tasemeid kasutatakse valede sündmuste logimiseks, võivad tähtsamad ning kohest tähelepanu nõudvad veateated teiste ridade vahele ning tasemed ei täida enam oma eesmärki.

3.5 Pideva evitamise lahenduse valimine

Peatükis 2.3.2 tõi autor välja, et rakenduse evitamiseks oleks tarvis leida lahendus, mis järgiks pideva evitamise praktikat ning lubaks üles seada kahte paralleelset keskkonda – üks puhverkeskkond rakenduste testimiseks ja uue funktsionaalsuse demonstreerimiseks kliendile ning teine keskkond, mida on reaalseks kasutamiseks lõppkasutajate poolt.

---

[1] http://logback.qos.ch/

[2] http://www.slf4j.org/

Kahe keskkonna paralleelne jooksutamine tähendab ka seda, et kahte versiooni on vaja igast rakenduse komponendist: rakendusliidesest, kasutajaliidesest ning andmebaasist.

3.5.1 Kasutajaliidese evitamine

Kasutav kasutajaliides on *Single Page Application* (edaspidi SPA), mis suhtlusel kasutajaga ei väljasta rakenduses uuele lehele navigeerides tervet uut veebilehte, vaid kirjutab selle asemel olemasoleva veebilehe dünaamiliselt üle [9]. Seetõttu on SPA puhul lihtsam ka evitamine, milleks piisab tavalisest staatilisest veebiserverist.

Sellest tulenevalt on võimalik kasutajaliidese evitamiseks kasutada üliõpilasesinduse olemasolevat veebimajutusteenust, mille kasutamist eelistas ka klient. Kahe paralleelse keskkonna jaoks oleks tarvis läbi veebimajutusteenuse konsooli luua mõlema keskkonna jaoks eraldi alamdomeenid ning ka kaustad, kuhu ehitatud failid paigaldada.

CD praktika järgmiseks on lisaks tarvis leida eraldi lahendus, mis uute koodimuudatuste puhul kasutajaliidese SPA failid valmis ehitaks ning seejärel need automaatselt üliõpilasesinduse veebiserveri vastavasse kausta paigaldaks.

Antud ülesande lahendamiseks on autori arvates kõige mõistlikum kasutada uuesti *GitHub Actions* platvormi, mille kasuks otsustati ka peatükis 3.3 CI realiseerimisel. Platvorm toetab kasutajaliidese SPA failide genereerimiseks vajalikku tehnoloogiat (*Node.js*) ning võimaldab *rsync* või *scp* tööriistu kasutades kopeerida genereeritud faile üle SSH ühenduse edastada. Antud lahenduse kasutamine ei tooks sisse lisakulusid ning ei vajaks järgmise platvormi toomist projekti, sest kasutajaliidese koodivaramu oli juba eelnevalt *GitHub* platvormil.

3.5.2 Rakendusliidese evitamine

Rakendusliidese evitamiseks olemasoleva veebimajutusteenuse kasutamine ei olnud võimalik, sest piiratud õiguste tõttu ning puuduvate tööriistade tõttu ei olnud võimalik platvormil rakendusliidest ehitada ega käivitada. Seetõttu tuli rakendusliidese evitamiseks leida teistsugune lahendus.

Võimalus oleks olnud rentida rakendusliidese evitamise eesmärgil eraldi virtuaalserver, kus õigustega probleeme ei tekiks ning kõik vajaminevad tööriistad on võimalik endal paigaldada. CD protsessi realiseerimiseks oleks võinud kasutada 3.3 peatükis mainitud Jenkins platvormi. Virtuaalserveri lahendus oleks suhteliselt madalate kuludega, kuid

lahenduse üles seadmine nõuaks hulga aega ning vaeva. Antud lahenduse puhul on tarvis tegeleda aeg-ajalt ka serveri haldusega ning seetõttu peaks projektis olema ka administreerimispädevustega inimesi, mida CD praktikate järgimisega oli eesmärk vältida.

Autori arvates oleks rakendusliidese evitamisel mõistlik variant kaaluda ka teenuse sisse ostmist. Võimalus oleks kasutada näiteks *Heroku*[1] platvormi, mis võimaldaks nii CD praktikate järgimist kui rakenduse majutust ning toetab ka paralleelsete keskkondade kasutamist [10]. Taolise välise teenuse kasutamine eemaldaks serveri haldamise kohustuse ning vähendaks oluliselt lahenduse üles seadmisele kuluvat aega. Antud lahendus oleks tõenäoliselt veidi kulukam, kuid autori arvates tasub see lisanduva töökindluse ning kasutuslihtsuse näol end pikemas perspektiivis ära.

[1] https://www.heroku.com/

4 Teostus

Peatükis 3 analüüsis autor erinevaid lahendusvõimalusi peatükis 2 analüüsitud probleemide lahendamiseks. Järgnevalt kirjeldab autor, kuidas antud lahenduste realiseerimine kulges, kirjeldab esinenud probleeme ning hindab saavutatud tulemust.

4.1 Arenduskeskkonna lahenduse realiseerimine

4.1.1 Andmebaasiserveri loomine

Peatükis 3.1 tehtud analüüsi põhjal otsustas autor lokaalse andmebaasi üles seadmiseks kasutada *Docker* tehnoloogial põhinevat lahendust.

Lahenduse kasutamiseks tuli autoril esmalt vastav tarkvara oma isiklikku arvutisse installeerida. Windows või MacOS operatsioonisüsteemi peal on antud protsess suhteliselt lihtne, võimalus on kasutada *Docker Desktop* lahendust, mis installeerib ühe paketina nii *Docker Engine* kui ka *Docker Compose* [11].

Autor kasutas aga arenduseks Ubuntu operatsioonisüsteemi, mistõttu puudus võimalus kasutada *Docker Desktop* varianti ning nii *Engine* kui *Compose* tuli installeerida eraldi. Selle jaoks oli *Docker*-i poolt olemas eraldi juhis[1], mille järgimisel probleeme ei tekkinud kuid nõudis siiski teatud aja.

Docker-i installeerimisega ühele poole jõudnud, asus autor andmebaasi üles seadma. Selleks kasutas autor PostgreSQL ametlikku konteineri *image*-t[2], mida oli võimalik otse läbi käsurea *Docker Hub* platvormilt alla laadida, kasutades selleks käsku „*docker pull postgres*". Alla laetud konteineri jooksutamiseks kasutas autor esialgu „*docker run“* käsku, mille argumentidena oli võimalik ära defineerida vajalikud parameetrid, nimelt andmebaasi nimi, port, kasutajanimi ning parool.

Esialgu plaanis autor andmebaasikonteineri jooksutamiseks kasutatava käsu jagamiseks luua eraldi skriptifail, mida seejärel läbi versioonihalduse jagada oleks võimalik jagada.

[1] https://docs.docker.com/engine/install/ubuntu/

[2] https://hub.docker.com/_/postgres

See aga oleks tähendanud, et skripte oleks tulnud luua kaks – üks Windowsi ning teine Unix-põhistele platvormidele.

Et andmete duplitseerimist vältida, otsustas autor kasutada andmebaasi käivitamiseks *Docker Compose* tööriista, mida enamasti kasutatakse küll mitme konteineri samaaegseks jooksutamiseks, kuid sobib antud juhul ka üksiku andmebaasikonteineri jooksutamiseks. Postgres konteineri jooksutamiseks lõi autor eraldi konfiguratsioonifaili (Joonis 3), milles defineeriti ära andmebaasiühenduseks vajalikud parameetrid ning kasutav versioon konteinerist.

```
version: "3"
services:
  db:
    image: "postgres:11.4"
    container_name: "postgres-yttrium-api"
    ports:
      - "5433:5432"
    environment:
      POSTGRES_PASSWORD: postgres
      POSTGRES_USER: postgres
      POSTGRES_DB: yttrium
      SCHEMA: public
    restart: unless-stopped
```

Joonis 3. *Docker Compose* konfiguratsioonifail andmebaasikonteineri jooksutamiseks

Loodud konfiguratsiooni käivitamiseks tuli jooksutada konfiguratsioonifailiga samas kaustas käsku „*docker-compose up“*.

4.1.2 Rakendusliidese andmebaasiühenduse loomine

Andmebaasiühenduse loomiseks tuli rakendusliidese *Spring Boot* konfiguratsioonifailis *application.properties* ära defineerida järgmised parameetrid:

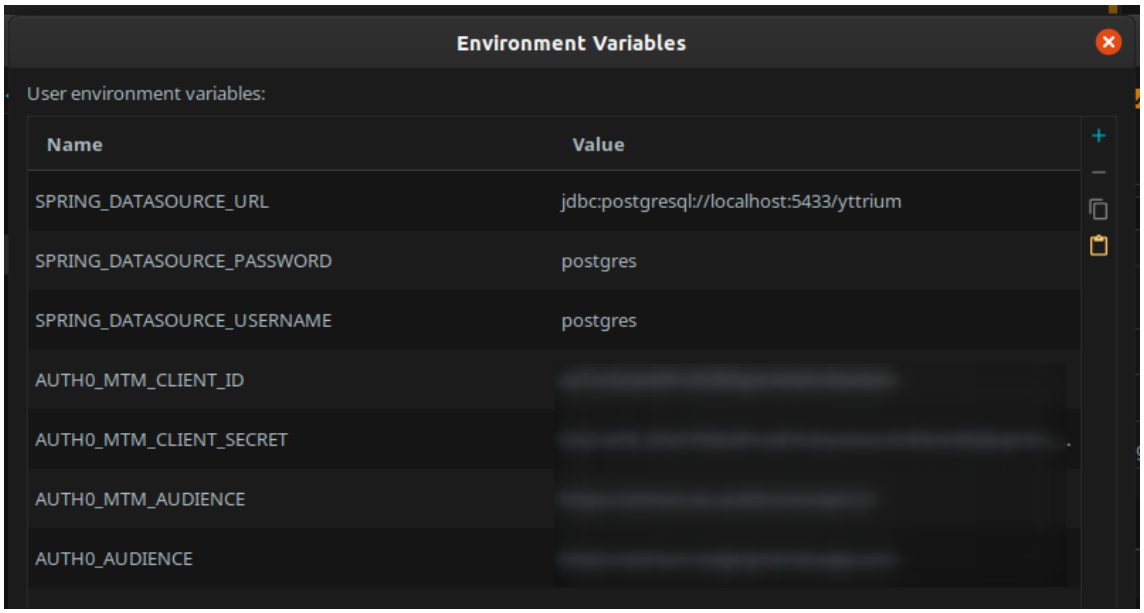
- andmebaasi URL
- andmebaasi kasutajanimi ja parool
- *DataSource driver* (PostgreSQL)

Peatükis 2.1.2 tõi autor välja, et turvalisuse kaalutustel jäätis rakendusliidese esialgne autor antud konfiguratsioonifaili seal sisalduvate saladuste tõttu koodivaramust välja. Peatükis 3.2 tehtud analüüsi põhjal otsustas autor probleemi lahendada kasutades konfiguratsioonifaili sees saladustele ning keskkonnast sõltuvatele muutujatele viidata keskkonnamuutujatega (näidis Joonis 4).

```
application.properties
1  spring.datasource.url=${SPRING_DATASOURCE_URL}
2  spring.datasource.username=${SPRING_DATASOURCE_USERNAME}
3  spring.datasource.password=${SPRING_DATASOURCE_PASSWORD}
4  spring.datasource.driver-class-name=org.postgresql.Driver
5  spring.jpa.hibernate.ddl-auto=validate
6  spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
7
8  spring.liquibase.change-log=classpath:/db/changelog.xml
9
10 spring.security.oauth2.resourceserver.jwt.issuer-uri=https://yttrium.eu.auth0.com/
11 auth0.audience=${AUTH0_AUDIENCE}
12
13 auth0.mtm.client.id=${AUTH0_MTM_CLIENT_ID}
14 auth0.mtm.client.secret=${AUTH0_MTM_CLIENT_SECRET}
15 auth0.mtm.audience=${AUTH0_MTM_AUDIENCE}
```

Joonis 4. Keskkonnamuutujatega *Spring Boot* konfiguratsioonifail.

Antud keskkonnamuutujad tuli defineerida samas kontekstis rakendusliidese protsessiga. Autori poolt kasutatud *IntelliJ IDEA* IDE lubas seda mugavalt teha kohe *Spring Boot* rakenduse *Run Configuration*-is, kus on võimalik jooksutava protsessi jaoks defineerida eraldi keskkonnamuutujad (Joonis 5). Antud lahendust kasutades viitas autor eelmises peatükis loodud andmebaasile ning rakendusliides sai edukalt lokaalse andmebaasiga ühendatud.



Joonis 5. Keskkonnamuutujate seadistamine *IntelliJ IDEA*-s

4.2 Esialgsete automaattestide loomine

Peatükis 2.1.1 tehtud analüüsi põhjal otsustas autor, et rakenduse töökindluse ning jätkusuutlikkuse parandamiseks oleks mõistlik rakenduse kriitilisemad komponendid katta automaattestidega.

Käesoleva töö raames otsustas autor aja kokkuhoiu eesmärgil realiseerida vaid esimesed paar automaattesti, mis aitaksid testida plaanitud pideva integratsiooni teenuseid. Ülejäänud testid olemasoleva koodi katmiseks plaaniti realiseerida jooksvalt enne rahastusplatvormi produktsioonifaasi jõudmist ning uue funktsionaalsuse sisse toomisel järgida võimalusel *Test Driven Development* praktikat, et ka need oleksid koheselt testidega kaetud.

4.3 Pideva integratsiooni ning evitamise teenuste üles seadmine

4.3.1 Uute harude loomine koodivaramus

Kahe paralleelse keskkonna evitamiseks tuli autoril esmalt välja töötada lahendus koodivaramus, mis võimaldaks plaanitud kahe keskkonna koodi eristada. Rakendus- ning kasutajaliidese kood oli hoitud mõlemad *GitHub* platvormil, kuid eraldi koodivaramutes.

Autor kasutas keskkondade eristamiseks Git-i harusid ehk *branch*-e, kus mõlemas koodivaramus jäi lõppkasutajate poolt kasutatav versioon vaikimisi harule *master*, ning testimiseks mõeldud puhverkeskkonna jaoks loodi mõlemas varamus uus haru nimega *staging.*

4.3.2 Rakendusliidese testide automaate jooksutamine

Peatükis 3.3 otsustas autor testide automaatseks jookusutamiseks kasutada *GitHub Actions* platvormi.

Rakendusliidese ehitamiseks kasutatava *Gradle* raamistiku CI üles seadmiseks oli võimalik järgida vastavat ametlikku juhendit[1], tänu millele oli protsess kiire ning murevaba.

[1] https://help.github.com/en/actions/language-and-framework-guides/building-and-testing-java-with-gradle

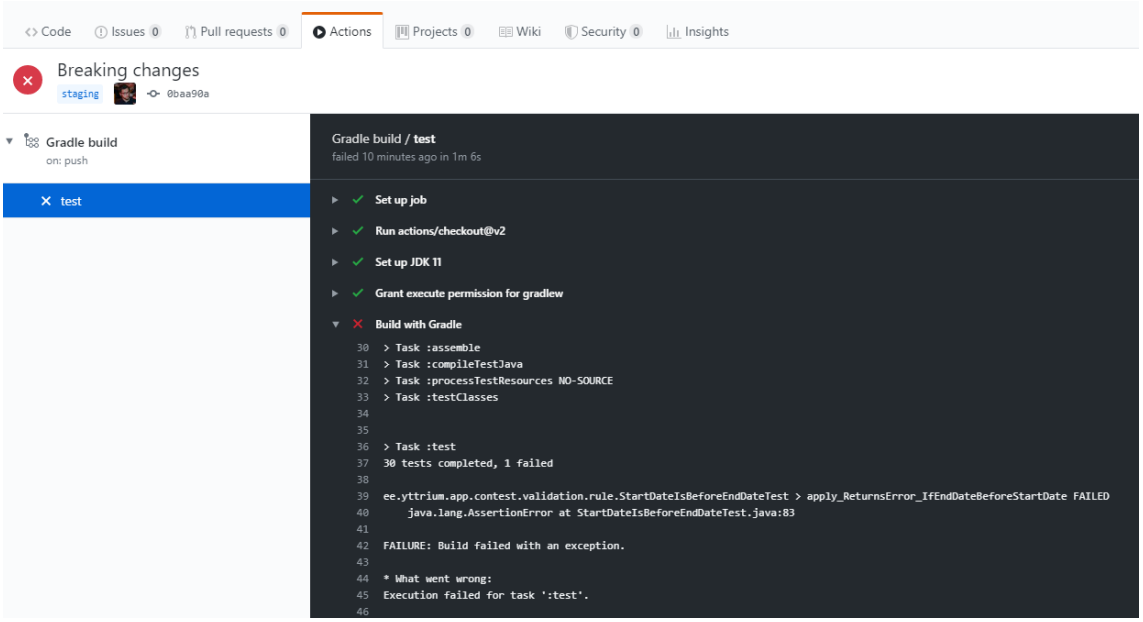
Läbi *GitHub*-i kasutajaliidese tuli luua kindla süntaksiga YAML formaadis konfiguratsioon (Joonis 6). Antud failis määras autor esmalt kasutatavad harud (*master* ja *staging)* ning sündmused, mille peale teste jooksutada.

```
name: Gradle build

on:
  push:
    branches: [ master, staging ]
  pull_request:
    branches: [ master, staging ]

jobs:
  test:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Set up JDK 11
      uses: actions/setup-java@v1
      with:
        java-version: 11
    - name: Grant execute permission for gradlew
      run: chmod +x gradlew
    - name: Build with Gradle
      run: ./gradlew build
```

Joonis 6. *GitHub Actions* konfiguratsioon rakendusliidese ehitamiseks.

Testide jooksutamiseks kasutas autor *Gradle build* protsessi, mis lisaks testide jooksutamisele kontrollib, et rakenduse kompileerimisel probleeme ei esine.

Testide tagasiside oli võimalik muudatuste järel näha läbi *GitHub* kasutajaliidese, kus ebaõnnestunud testide korral oli tehtud muudatuste juures punane märge ning viide konkreetsetele ebaõnnestunud testidele (Joonis 7). Lisaks tagasisidele kasutajaliideses saatis platvorm automaatselt muudatuse teinud arendajale eraldi välja ka meili.



Joonis 7. Tagasiside ebaõnnestunud testide kohta *GitHub Actions* platvormil.

4.3.3 Rakendusliidese evitamislahenduse seadistamine

Autor esitas peatükis 3.5.2 tehtud analüüsi ka kliendile, kes nõustus autori poolt tehtud ettepanekuga kasutada rakendusliidese evitamiseks kasutada välist teenust. Autor valis selleks personaalse positiivse kogemuse põhjal *Heroku* platvormi.

Platvormil oli olemas eraldi eelseadistatud konfiguratsioon *Gradle* ja *Spring Boot* tehnoloogiatega rakenduse jooksutamiseks ning ka lihtne integratsioon *PostgreSQL* andmebaasi ühendamiseks rakendusega. *Heroku* pakkus ka otsest integratsiooni *GitHub* platvormiga, mistõttu oli koodivaramuga ühendamine kiire ning mugav protsess.

Protsessi muutis lihtsamaks ka peatükis 4.1.2 loodud lahendus rakendusliidese konfigureerimiseks, sest sama meetodit kasutas *Heroku* andmebaasi ühendamiseks evitatud rakendusega – andmebaasiühenduseks vajalikud parameetrid sisestas platvorm automaatselt rakendusliidese käivitamisel keskkonnamuutujatesse. Samal viisil oli võimalik defineerida ka rakenduses kasutatud saladused.

Master ning *staging* keskkonna seadistamise jaoks oli *Heroku* platvormil spetsiaalne lahendus nimega *Pipelines*[1], mis lubas evitada paralleelseid keskkondi vastavalt seadistatud harule.

4.3.4 Kasutajaliidese automaatne evitamine

Kasutajaliidese evitamiseks otsustas autor peatükis 3.5.1 tehtud analüüsi põhjal jällegi kasutada *GitHub Actions* platvormi, mille läbi kopeerida genereeritud SPA rakenduse failid üliõpilasesinduse veebiserverile.

4.3.4.1 Keskkondade seadistamine

Esmalt otsustas autor välja töötada lahenduse kahe erineva kasutajaliidese keskkonna toetamiseks. Lisaks kasutajaliidese enda koodile oli kahe keskkonna peamine erinevus kasutatav API. Kasutajaliidese *master* ja *staging* keskkonnad pidid kasutama vastava keskkonna rakendusliidest.

Sarnaselt rakendusliidesele otsustas autor antud ülesande lahenda keskkonnamuutujate abil. Erinevalt rakendusliidesest ei olnud kasutajaliidese puhul vajadust hoida koodivaramus delikaatseid andmeid, mistõttu sobis keskkondade seadistamiseks hästi *Vue CLI* tööriista režiimide funktsionaalsus. Rakenduse ehitamisel kasutatava *vue-cli-service build* käsule on võimalik juurde anda „*—mode“* argument, mis seadistab rakenduse kasutama vastava nimega keskkonnamuutujate faili projekti juurkaustas. Näiteks kui kutsuda välja käsk „*vue-cli-service build –mode production“,* kasutab rakendus keskkonnamuutujaid failist *„.env.production“* [12].

Antud viisil lõi autor esialgu *staging* ja *master* keskkonna jaoks koodivaramusse eraldi keskkonnamuutujate failid, kus oli ära defineeritud kasutatava rakendusliidese versiooni URL.

4.3.4.2 Genereeritud rakenduse kopeerimine läbi SSH

Pärast SPA failide genereerimist tuli need evitamiseks toimetada üliõpilasesinduse veebiserverisse.

Esialgu plaanis autor selleks kasutada *rsync* tööriista, mida oli võimalik kasutada *GitHub Action*-i sees genereeritud failide kopeerimiseks, luues üliõpilasesinduse veebiserveriga

[1] https://devcenter.heroku.com/articles/pipelines

esmalt SSH ühenduse. Lahenduse katsetamisel aga selgus, et *rsync*-i taoliseks kasutuseks peab olema tööriist installitud mõlemal ühenduse poolel – nii saatja kui vastuvõtja masinas [13]. Üliõpilasesinduse veebiserveris antud tööriista kasutada võimalik ei olnud ning administraatoriõiguste puudumise tõttu ei olnud võimalik ka serverisse tarkvara juurde lisada.

Seetõttu otsustas autor *rsync*-i asemel kasutada failide kopeerimiseks FTP protokolli, mida toetas ka üliõpilasesinduse veebiserver. FTP-ga failide kopeerimiseks *GitHub*-i poolt ametlikku *Action*-it ei leidunud, küll aga oli olemas kogukonna poolt loodud vabavaraline lahendus[1], mille kasutamisel probleeme ei esinenud.

4.3.4.3 Probleemid CORS-iga

Evitamisel esines kaks suuremat probleemi, mida autor esialgu ei osanud ette näha ning mis vajasid eraldi lahendusi.

Esimene probleem esines seoses CORS (*Cross-Origin Resource Sharing)* mehhanismiga, mille tõttu ei olnud võimalik kasutajaliidesel suhelda rakendusliidesega, kuna nad asusid erinevatel domeenidel [14]. Kasutajaliides asus üliõpilase domeenil *tipikas.ee* ning rakendusliides *Heroku* platvormi domeenil *herokuapp.com.*

Esinenud probleemi lahendamiseks oli kaks võimalust: kas viia rakendusliides üle kasutajaliidesega samale domeenile või seadistada rakendusliides viisil, mis lubaks päringuid ka välise *origin*-ga domeenidelt. Eelneva kogemuse põhjal otsustas autor kasutada viimast varianti, luues selleks eraldi *Filter* komponendi (v. t. Lisa 1).

[1] https://github.com/SamKirkland/FTP-Deploy-Action

5 Lahenduse valideerimine

5.1 Arenduskeskkonna kasutamine

Rakendusliidesele uut funktsionaalsust lisades täitis loodud lahendus oma eesmärki – esialgse ülesseadmise järel ei esinenud arendusperioodi jooksul lokaalse andmebaasiga kordagi probleemi või takistust, mida ei olnud võimalik lahendada paari minutiga. Autor sai keskenduda täielikult rakendusliidese enda arendamisele ning ei pidanud kulutama aega kõrvaliste probleemidega tegelemiseks.

Samuti oli pärast esialgset seadistust triviaalne ka realiseeritud keskkonnamuutujate-põhise konfiguratsiooni kasutamine. Tänu *IntelliJ IDEA* sees eelseadistatud *Run Configuration*-i kasutamisele laeti kasutatavad keskkonnamuutujad automaatselt rakendusliidese käivitamisel, mistõttu edaspidi autor arenduse jooksul konfigureerimisele aega kulutama ei pidanud.

Lisaks pakkus loodud lahendust lisaväärtust olukorras, kus autoril oli tarvis diagnoosida üht produktsioonikeskkonnas raporteeritud probleemi, mille põhjustasid vigased andmed antud keskkonna poolt kasutatavas andmebaasis. Probleemi tuvastamiseks oli tarvis antud vigaseid andmeid näha ka lokaalselt jooksvas rakendusliideses ning tänu loodud konfiguratsiooni-lahendusele ei olnud tarvis produktsioonikeskkonna andmeid ümber kirjutada lokaalsesse andmebaasi, vaid piisas alternatiivse *Run Configuration*-i loomisest, mille keskkonnamuutujad viitasid lihtsalt produktsioonikeskkonna andmebaasile.

5.2 Loodud pideva integratsiooni lahenduse kasutamine

Pideva integratsiooni praktikate üles seadmiseks kulunud aeg tasus end arendamisprotsessi jooksul igati ära, aidates mitmel korral vältida vigase või katkise koodi jõudmist evitatud keskkondadesse.

Enim kasu oli autoril loodud lahendusest rakendusliidese identiteedihaldussüsteemi välja töötamisel. Kliendi soovil oli vaja arendada lahendus, mis lubaks kasutajatel lisaks e-maili ja parooliga süsteemi sisse logimisele kasutada ka Google või TalTech UNI-ID kasutajat. Samuti pidi loodud süsteem võimaldama rollipõhist õiguste haldust, mis lubaks teatud sisu kasutajaliideses näha vaid kindla rolliga kasutajatel. Autor kasutas lahenduse

realiseerimisel *Spring Security* ning *Auth0*[1] raamistikke, mis võimaldasid OAuth 2.0[2] protokolli järgides suurema vaevata täita kliendi poolt seatud nõuded.

Identiteedihaldussüsteemi välja töötamise jooksul oli autoril tarvis uue lahenduse kasutusele võtmiseks erinevaid muudatusi ka rakendusliidese koodis, peamiselt komponentides ning mudelites, mille andmed olid seotud kasutajaga. Konkursi äriloogika refaktoreerimisel ununes autoril tagasi lisada ajutiselt eemaldatud kontroll, mis lubab konkurssi kustutada vaid administraatori rolliga kasutajal. Antud kontrolli puudumine oleks lubanud ka tavakasutajatel otse rakendusliidese vastu päringut tehes loodud konkurssi kustutada, mis oleks olnud suur potentsiaalne turvarisk. Siinkohal oli väga kasuks realiseeritud pideva integratsiooni lahendus, mis jooksutas antud muudatuse peale automaatteste, millest konkursside rollipõhist autentimist kontrolliv üksustest ebaõnnestus. Kuna lahendus oli seadistatud selliselt, et automaatse evitamise tööriistad käivituvad vaid siis, kui kõik automaattestid õnnestuvad, ei evitatud vigast versiooni ning süsteem andis autorile automaatselt tema vigasest muudatusest teada.

Taolisi olukordi, kus loodud pideva integratsiooni lahenduse poolt antud kohene automaatne tagasiside aitas kiiremini tuvastada ning lahendada arenduse käigus esinenud probleeme oli mitmeid.

Pideva integratsiooni tööriistaks valitud *GitHub Actions*-i integreerus olemasolevatesse arendusprotsessidesse lihtsalt, kuna *GitHub* versioonihaldusplatvorm oli juba eelnevalt kasutuses ning seetõttu ei olnud vajadust lisanduvat platvormi kasutusele võtta.

5.3 Evitamislahenduse kasutamine

Sarnaselt arenduskeskkonna jaoks loodud lahendusega ei nõudnud ka kumbki realiseeritud pideva evitamise lahendus peale esialgset seadistamist lisanduvat tähelepanu. Autoril oli võimalik rohkem aega pühendada arendusele, sest ei olnud vajadust eraldi tegelda rakenduse evitamisega, kogu protsess töötas taustal automatiseeritult. Nii rakendusliidese jaoks kasutatav *Heroku* platvormi põhine lahendus

[1] https://auth0.com/

[2] https://oauth.net/2/

kui ka kasutajaliidese jaoks kasutatav FTP-põhine lahendus töötasid arendusprotsessi jooksul järjepidevalt ilma probleemideta.

Evitatud rakendusliides pakkus lisandväärtust ka arendusprotsessi käigus. Eelnevalt oli kasutajaliidese arendamiseks tarvis samaaegselt jooksutada ka lokaalset rakendusliidest, isegi kui kogu arendus toimub kasutajaliidese poolel ning rakendusliidese poolel muudatusi ei tehta. Tänu loodud evitatud keskkondadele oli võimalik luua lokaalse kasutajaliidese jooksutamiseks kaks eraldi konfiguratsiooni, mis vastavalt ülesandele viitab kas lokaalsele rakendusliidesele (juhul, kus arendus toimub mõlema liidesega paralleelselt) või evitatud *staging* keskkonna rakendusliidesele (juhul, kus arendus toimub vaid kasutajaliidese poole).

Kahe evitatud keskkonna lahendus tuli kasuks ka kliendiga ehk üliõpilasesindusega suhtlemisel. Kui oli tarvis enne suuremate muudatuste produktsioonikeskkonda sisse viimist eelnevalt kliendiga nõu pidada ning tagasiside küsida, piisas vaid neile avaliku *staging* keskkonna kasutajaliidese veebiaadressi saatmisest, kus klient sai iseseisvalt uut funktsionaalsust testida enne kui see produktsioonikeskkonda jõudis.

5.4 Väline validatsioon

Käesoleva töö algul seadis autor eesmärgi parandada olemasoleva lahenduse jätkusuutlikkust ning realiseerida uus funktsionaalsus samuti viisil, mis tagaks jätkusuutlikkuse ka tulevikus.

Eesmärgi olemuse tõttu oli raske autoril endal väita, et seatud eesmärk sai tänu käesoleva töö käigus realiseeritud lahendustele lahendatud. Töö autor oli küll ise loodud süsteemiga tuttav, kuid jätkusuutlikkuse arvestatavaks hindamiseks oli tarvis ka välist erapooletut validatsiooni.

Selleks võttis autor ühendust arendajaga, kes oli juba käesolevale tööle eelnevalt projektiga tutvunud ning plaanis tulevikus potentsiaalselt arendusmeeskonnaga liituda. Antud arendajaga otsustas autor korraldada kasutajatesti, kus testijal oli ülesandeks võimalikult vähese abiga üles seada arenduskeskkond, teha nii rakendus- kui kasutajaliideses väike muudatus ning seejärel kinnitada muudatuste kajastumist evitatud *staging* keskkonnas. Testi edu mõõtmiseks jälgis autor, et testi jooksul ei esineks arendajal samu probleeme, mis ilmnesid autoril projekti üle võttes. Samuti ei tohiks

testijal tekkida vajadust välise abi järele, kogu test pidi olema võimalik läbida kasutades loodud juhendeid. Testi järgselt tuli arendajal hinnata ka üldist arenduskogemust ning kogu protsessi intuitiivsust.

Kasutajatesti tulemus oli positiivne. Kuigi arendajal tuli andmebaasilahenduse kasutamiseks eelnevalt tegeleda *Docker*-i tarkvara installeerimisega, võttis lõpuks nii rakendus- kui kasutajaliidese üles seadmiseks testitaval kokku veidi üle ühe tunni, autoril võttis sama protsess aega ligi päeva. Välist abi seal juures testija ei vajanud. Muudatuste tegemine oli samuti triviaalne ning ette antud ülesanded said probleemideta lahendatud. Arenduskogemuse osas tõi testija positiivsena välja kogu pideva integratsiooni ning evitamise süsteemi – automaattestid andsid kindlust, et tema muudatustel ei olnud negatiivset mõju süsteemi kui terviku tööle, evitamise protsess oli „üllatavalt lihtne ning kiire“. Ainsa negatiivse aspektina tõi ta välja, et rakendusliidese evitamise protsess võiks olla läbipaistvam – kasutajaliidese evitamise protsess oli täielikult *GitHub*-is jälgitav, kuid rakendusliidese ehitamisel esinenud võimalike hoiatuste nägemiseks tuleb iga muudatuse korral eraldi külastada *Heroku* platvormi veebiliidest.

6 Tulemused

Autori hinnangul on kõik peatükis 1.2 püsitatud probleemid saanud töö jooksul lahendatud ning seatud eesmärk täidetud – loodud lahendused aitavad parandada olemasoleva rakendusliidese jätkusuutlikkust ning samuti sai realiseeritud rakenduse evitus.

Peatükis 2.1 tehtud analüüsi põhjal hindas autor, et arenduskeskkonna üles seadmisel põhjustasid suuremat ajakulu eelkõige puuduv lahendus rakendusliidese tööks vajaliku andmebaasi loomiseks ning jagamata jäänud konfiguratsioonifailid. Tänu peatükis 4.1 loodud lahendustele on autori hinnangul arenduskeskkonna üles seadmise protsess etem nii väiksema ajakulu kui ka üldise mugavuse aspektist. Uuel arendajal projektis on võimalik kiiremini asuda tööle reaalse arendustööga ning ideaaljuhul puudub ka vajadus välise abi järele.

Olemasoleva koodibaasi muudetavust ning üldist arendusmugavust aitavad parandada realiseeritud automaattestid ning pideva integratsiooni teenused, mis teste ka automaatselt jooksutavad. Antud lahendus tagab, et kui arendaja tehtud muudatuste tõttu peaks mingil põhjusel olema häiritud süsteemi töö, antakse talle sellest koheselt teada nii CI kasutajaliideses kui ka emaili teel. Loodud lahendus annab arendajale meelekindlust muudatuste tegemisel ning tagab süsteemi parema töökindluse, mis omakorda suurendab ka kogu projekti jätkusuutlikkust.

Rakenduse evitamiseks realiseeritud lahendus loodi samuti jätkusuulikkust meeles pidades. Pideva evitamise praktikate järgmine lubab arendajal rakenduse uut versiooni evitada vaid koodivaramu kindlasse harusse muudatuse sisse viimisega. Kogu ülejäänud protsess on automatiseeritud ja uuel arendajal projektis puudub reaalne vajadus serverite halduseks.

Loodud kahe evitatud keskkonnaga lahendus muudab autori arvates mugavamaks ka üldise täis-pinu arendusprotsessi, kus vastavalt vajadusele on eelseadistatud konfiguratsioonidega kiiresti võimalik vahetada lokaalse ning evitatud keskkondade vahel.

7 Kokkuvõte

Käesoleva töö raames asus autor üle võtma tarkvaraprojekti olemasoleva koodibaasiga ning kirjeldas selle käigus esinenud erinevaid probleeme ning murekohti, mis muutsid uuele arendajale projektis arendusprotsessi tülikaks ning ebamugavaks. Taolisel kujul ei olnud projekti arendus autori arvates jätkusuutlik ning tulevaste arenduste soodustamiseks tuli olukorda parandada.

Töö esimeses osas tutvuti eesmärgi saavutamiseks esmalt probleemiga lähemalt ning kaardistati peamised rakenduse jätkusuutlikkust mõjutavad tegurid. Seejärel analüüsiti ning kaaluti erinevaid võimalusi antud probleemide lahendamiseks läbi erinevate tehnoloogiate või protsesside kasutusele võtmisega.

Töö teises osas realiseeris autor tehtud analüüsi põhjal erinevad lahendused püstitatud probleemide lahendamiseks ning kirjeldas sünteesi jooksul esinenud katsumisi ning nendest saadud õppetunde.

Töö tulemusena seati üles erinevad protsessid ning süsteemid, mis kokkuvõttes muudavad uuele arendajale projektiga liitumise ning seal arendustööde tegemise võimalikult lihtsaks ja mugavaks. Loodud lahendus hoiab ära autoril projekti üle võttes tekkinud probleemide esinemise tulevikus ning aitab parandada kogu tarkvaraprojekti jätkusuutlikkust.

Kasutatud kirjandus

[1] K. G. Mänd, „TalTech üliõpilasesinduse veebipõhise rahastusplatvormi rakendusliidese arendus järgides kasutuslookeskse arhitektuuri põhimõtteid,“ Tallinna Tehnikaülikool, Tallinn, 2019.

[2] S. Magni, „Software tests as a documentation tool,“ 2020. [Võrgumaterjal]. Available: https://dev.to/noriste/software-tests-as-a-documentation-tool-36pl.

[3] A. Ayer, „git-crypt - transparent file encryption in git,“ 2020. [Võrgumaterjal]. Available: https://www.agwa.name/projects/git-crypt/.

[4] A. Ayer, „Experimental Windows Support,“ 2020. [Võrgumaterjal]. Available: https://github.com/AGWA/git-crypt/blob/master/INSTALL.md#experimental-windows-support.

[5] GitHub, „Releases of git-crypt,“ 2020. [Võrgumaterjal]. Available: https://github.com/AGWA/git-crypt/releases.

[6] M. Rehkoph, „Importance of CI,“ 2020. [Võrgumaterjal]. Available: https://www.atlassian.com/continuous-delivery/continuous-integration.

[7] Stackshare, „What are the best Continuous Integration Tools?,“ 2020. [Võrgumaterjal]. Available: https://stackshare.io/continuous-integration.

[8] GitHub, „GitHub Actions,“ 2020. [Võrgumaterjal]. Available: https://github.com/features/actions.

[9] Mozilla, „SPA (Single-page application),“ 2020. [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Glossary/SPA.

[10] Heroku, „Managing Multiple Environments for an App,“ 2020. [Võrgumaterjal]. Available: https://devcenter.heroku.com/articles/multiple-environments.

[11] Docker, „Docker Desktop,“ 2020. [Võrgumaterjal]. Available: https://www.docker.com/products/docker-desktop.

[12] Vue JS, „Modes and Environment Variables,“ 2020. [Võrgumaterjal]. Available: https://cli.vuejs.org/guide/mode-and-env.html.

[13] SS64.com, „rsync,“ 2020. [Võrgumaterjal]. Available: https://ss64.com/bash/rsync.html.

[14] Mozilla, „Cross-Origin Resource Sharing (CORS),“ 2020. [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

Lisa 1 – CORS filter Spring raamistikus

```
@Component
@Order(Ordered.HIGHEST_PRECEDENCE)
public class SimpleCORSFilter implements Filter {

    @Override
    public void init(FilterConfig fc) {
    }

    @Override
    public void doFilter(ServletRequest req, ServletResponse resp,
                         FilterChain chain) throws IOException,
ServletException {
        HttpServletResponse response = (HttpServletResponse) resp;
        HttpServletRequest request = (HttpServletRequest) req;
        response.setHeader("Access-Control-Allow-Origin", "*");
        response.setHeader("Access-Control-Allow-Methods", "POST, GET,
OPTIONS, DELETE, PUT");
        response.setHeader("Access-Control-Max-Age", "3600");
        response.setHeader("Access-Control-Allow-Headers", "x-requested-with,
authorization, Content-Type, Authorization, credential, X-XSRF-TOKEN");

        if ("OPTIONS".equalsIgnoreCase(request.getMethod())) {
            response.setStatus(HttpServletResponse.SC_OK);
        } else {
            chain.doFilter(req, resp);
        }

    }

    @Override
    public void destroy() {
    }

}
```