TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Gert Mänd 211522IABM

AUTOMATSIOONIDE MONITOORIMISE JA KONFIGUREERIMISE PLATVORMI DISAIN JA PROTOTÜÜPIMINE KINDLUSTUSETTEVÕTTE BAASIL

Magistritöö

Juhendaja: Meelis Antoi
MSc
Konsultant: Kevin Lehtsalu
BSc

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: Gert Mänd

10.05.2023

Annotatsioon

Käesoleva magistritöö eesmärk on luua lahendus tarkvararobotite jälgimiseks ja haldamiseks veebirakenduse näol. Tarkvararobotite arvu suurenemisel muutub robotiga seonduvate andmete hallatavus ja osapoolte vaheline suhtlus ebaefektiivseks, seda just erinevate tabeltöötlusprogrammide kasutuse tõttu. Valmiv lahendus on loodud kindlustusettevõtte baasil, kuid universaalse lähenemisega seega sobib kõikidele tarkvararobotitega tegelevatele ettevõtetele.

Töö käigus valmis esimene prototüüp veebirakendusest. Esimene skoop asendab kõik tabeltöötlusprogrammidega seotud andmete haldamised. Rakenduses on võimalik jälgida põhilogisid ehk tarkvararobotite poolt tehtud töid. Nende andmetega tehakse pidevat statistikat veendumaks, et valminud robotid on töökorras ja loovad võimalikult palju väärtust. Samuti on võimalik lisada erinevat tüüpi konfiguratsioone, mis on robotite töö aluseks. Iga tarkvararobot on erinev, sellest tulenevalt on võimalik luua eraldi projekte, mis haldavad projektiga seotud andmeid, sealhulgas logisid, konfiguratsioone ja liikmed.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 66 leheküljel, 6 peatükki, 38 joonist, 5 tabelit.

Abstract

Designing and prototyping an automation monitoring and configuring platform based on an insurance company

The aim of this master's thesis is to find a solution for monitoring and managing software robots in the form of a web application. With the increase in the number of software robots, the manageability of robot-related data and communication between parties becomes inefficient, especially due to the use of various spreadsheet programs. Initially, the solution being developed is based on an insurance company, but it is suitable for all companies dealing with software robots.

During the work, the first prototype of the web application was completed. The first scope replaces all data management related to spreadsheet programs. In the application, it is possible to monitor the main logs, i.e. the work done by software robots. Continuous statistics are made with this data to ensure that the completed robots are operational and provide as much value as possible. In addition, different types of configurations can be added, which form the basis of the robots' work. Since each software robot is different, it is possible to create separate projects that manage project-related data, including logs, configurations, and members.

The thesis is in Estonian and contains 66 pages of text, 6 chapters, 38 figures, 5 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
Backend	Teenusrakendus
Blue Prism	Tarkvararobotite arenduseks mõeldud tarkvara
CRUD	*Create, Read, Update, Delete* - Meetodid
DDD	Domain-Driven-Design, peamine tarkvara kujundamise lähenemisviis
DTO	*Data transfer object*, andmeedastusobjekt
EF	*Entity Framework*, raamistik
Frontend	Kliendirakendus, kasutajaliides, kuvand
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastusprotokoll
JSON	*JavaScript Object Notation*, lihtsustatud andmevahetusvorming
JWT	*Json Web Token*, internetistandard andmete loomiseks koos krüpteerimisenga
LINQ	Language-Integrated Query
Middleware	Vahevara
NPM	*Node Package Manager* – Javascript'i teekide haldaja
RDBMS	Relatsiooniline andmebaas
REST	Representational State Transfer – veebiteenusega suhtlemise liidese arhitektuur
RPA	*Robotic Process Automation*, tarkvararobootika ja tööprotsesside automatiseerimine
SQL	Struktuurpäringukeel
Tarkvararobot	*Software robot* ehk robot või automatsioon
UI	Kasutajaliides
UX	Kasutajakogemus
XML	*Extensible Markup Language* - laiendatav märgistuskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Paljud ettevõtted seisavad manuaalsete protsesside suure osakaalu probleemi ees. Üks viis selle probleemi lahendamiseks on võtta kasutusele RPA[1] [1] [2] [3] [4]. Antud lähenemise puhul ehitatakse virtuaalsed robotid olemasoleva manuaalse protsessi automatiseerimiseks. Nende abiga saab protsesse automatiseerida kas minimaalselt või üldse ilma olemasolevat süsteemi muutmata, mis võimaldab töötajatel keskenduda väärtuslikumatele äriprioriteetidele. Automatiseerimise all mõeldakse näiteks info lugemist erinevat tüüpi failidest nagu Excel ja PDF, nende sisestamist ettevõtete süsteemidesse, andmete muutmist ja palju muud. Robotite arendusel on oluline, et säiliks konfigureeritavus ning paindlikkus. Infovahetus peab olema kiire, äriliste erandite puhul jõuaks veateated määratud isikutele parandamiseks ning tehniliste vigade puhul arendajatele vigade likvideerimiseks. Virtuaalne tööjõud koosneb tarkvararobotitest, mis on installeeritud virtuaalserveritesse ja mis on võimelised võtma üle kõik korduvad administraatori põhised protsessid, millega inimtööjõud igapäevaselt silmitsi seisab. Erinevaid ülesandeid saab täita iseseisvalt ilma inimese sekkumiseta.

RPA-st on kiiresti saamas valiktehnoloogia paljude erinevate organisatsioonide jaoks, mis hõlmavad mis tahes vertikaalset turgu. Ettevõtted, kes kasutavad RPA-d, kogevad järsult äritegevuse tulemuslikkust ja investeeringutasuvust [5]. RPA kasutab manuaalse ülesande automatiseerimiseks tarkvara. Sellel on süsteemi vaatenurgast samad võimalused ja piirangud nagu inimesel. Automatiseerimine ei asenda kunagi täielikult traditsioonilist arendust, vaid võib olla kasulik täiendus tavaarendusele, eriti juhtudel, kus tavaarendus ei ole otstarbekas, sest on ajaliselt või rahaliselt liiga kulukas. Näiteks andmete teisaldamine ühest süsteemist teise, kus teenuseid kasutades ei saa ühendusi luua. Tarkvararobootika on kõige kasulikum suuremahuliste ja väikese keerukusega ülesannete puhul, kus juhtumite vahelised erinevused on minimaalsed. Samas saab tarkvararoboteid kasutada ka pikkades ja keerukates protsessides, kuid sellistel juhtudel

[1] RPA – Robotic Process Automation

on arenduskulud ja veamäärad suuremad [6]. Ülesannet on mõistlik automatiseerida ainult teatud punktini. Tavaliselt on piir 80% juhtudest/töökoormusest. Kõik üle piiri on tõenäoliselt ebamõistlik ja peaks jääma käsitsitööks. Tarkvara valik on väga lai, võimalik on valida üle kümne erineva toote vahel. Samuti ei vaja robotite arendus väga suuri teadmisi programmeerimisest, kuid kindlasti kiirendab see keerulisemate protsesside puhul arenduse kiirust [7].

Näited kasuteguritest:

- Suurem täpsus – Robotid on väga täpsed ja järjepidevad ning ei tee inimlikke apsakaid ega trükivigu.
- Katkematu töövoog – Tarkvararobotid võivad töötada 24/7, sõltuvalt ressurssidest.
- Juhiste järgimine – Arendatud ja seadistatud robotid järgivad ettemääratud juhiseid. Oskavad vigade korral ennast parandada ning vajadusel eelmist ülesannet korrata. Nende töötulemus näitab, et sobivad ideaalselt rangete standarditega protsessides.
- Erinevate rakenduste integratsioon – Tavaliselt kulub palju IT-ressursse, et vananenud rakendused üksteisega ühendada. Robotitega on vastupidi – need töötavad nagu inimesed ja on suutelised navigeerima erinevatel tasanditel ilma, alussüsteemi muutmata.
- Parem töötaja moraal ja töökogemus – Töötajad saavad keskenduda rohkem põnevamasse ja strateegilisemasse töösse, mis toovad ettevõttele rohkem tulu.
- Suurem produktiivsus – Võrreldes manuaalse tööga on kõik protsessi etapid tõhusamad ja saavad kiiremini valmis.
- Tööde näited – Kasutajakontode lisamine või kustutamine, informatsiooni sisestamine ja eemaldamine, süsteemides navigeerimine, failide täitmine, andmete kontroll.

1.1 Metoodika

Käesoleva töö käigus selgitatakse lahti taust, tekkinud probleem ning pakutakse välja lahendus, mille skoop peab mahtuma lõputöö piiresse ja tagama piisavalt väärtust. Analüüsi käigus selgitatakse välja andmestik, mida valmiv tarkvara haldama peab, seejärel täpsustatakse, kes ja milliste kogemustega saab olla tarkvara kasutaja.

Funktsionaalsed nõuded tekkisid nii tarkvararobootika tiimil kui ka osakondade vahelise küsitluse teel. Peale nõuete määramist analüüsiti erinevaid tehnoloogia valikuid, mis oleksid kõige õigemad vahendid kogutud nõuete realiseerimiseks. Lahenduse valmimine on jaotatud erinevatesse osadesse, mis käsitlevad järgnevaid punkte:

- teenusepoolse rakenduse valmimine,
- kasutajaliidese rakenduse valmimine,
- kasutajaliidese näited ja selgitused valminud rakenduse näitel
- andmebaasi disain ja arendus,
- testimine.

Viimasena võetakse kokku valminud rakendus, valideeritakse valminud rakendust ning kirjeldatakse edasiarenduse võimalusi.

2 Probleemi ülevaade ja eesmärgi püstitus

Antud peatüki eesmärk on anda ülevaade probleemist, tutvustada hetkeolukorda ning püstitada projekti eesmärk.

2.1 Probleem

Tarkvararobotite arvu suurenemisel muutub robotiga seonduvate andmete hallatavus ja osapoolte vaheline suhtlus ebaefektiivseks. Igal projektil on spetsiifilised muutujad, konfiguratsioonid, mida erinevad isikud haldavad. Lisaks, robotite poolt tehtud tööde andmed tuleb võimalikult kiiresti logida, et osapooled saaksid vajadusel kontrollida, parandada või töö lõpetada. Hetkel käib kogu infovahetus ja andmete logimine kasutades tabeltöötlusprogrammi Excel. Automatiseerimise teekonna alguses oli tarkvara kasutamine mõistlik, kuna Microsofti litsentsid olid ettevõttel juba kasutuses. Olukorras, kus tarkvararobotite arv tõuseb aastas ~8 võrra, on andmemaht viimaste aastatega kordades muutunud, mis tähendab, et tarkvararobotite haldamiseks kulub väärtuslikku aega rohkem ning vajalikud lisaarendused jäävad tahaplaanile. Erinevate allikate sõnul on probleemide kuhjumisest tulenevalt RPA projektide ebaõnnestumise tõenäosus kuni 50% [8] [9]. Põhjused võivad olla järgnevad:

- Tervikliku platvormi puudumine – RPA eesmärke ei suudeta täita ja suur osa sellest tuleneb tervikliku platvormi ning skaleeritavate moodulite ja teadmiste puudumisest.

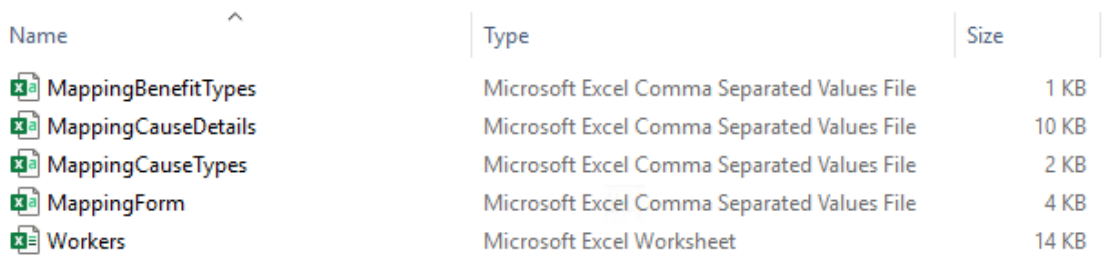
- Ebareaalsed ootused – Organisatsioon seab RPA projekti esmakordsel käivitamisel eesmärgid ja ootused, kuid neid tuleb kontrollida ja viia vastavusse sellega, mis on võimalik, mitte sellega, mida soovitakse.

- Liiga keeruliste protsesside automatiseerimine – RPA töötab kõige paremini standardiseeritud protsessides, mida saab samm-sammult määratleda ja mille töövood muutuvad harva, kui üldse. Ainuüksi RPA abil ulatusliku, kogu süsteemi hõlmava automatiseerimise katsed paljastavad kiiresti tehnoloogilise mittevastavuse.

2.2 Hetkeolukord

Igal projektil on vähemalt üks üldine ja vajadusel mitu väiksemat ärireegleid sisaldavat konfiguratsiooni faili (Joonis 1). Üks suuremaid miinuseid on just ühe tabeltöötlusfaili kasutamine samal ajal, kuna see võib põhjustada erinevate kasutajate vahel konflikte ja vead võivad kergesti tekkida. Lisaks on tabeltöötlusprogrammi kasutamine sageli väga töömahukas, eriti kui on vaja töödelda suuri andmemahte või teostada keerulisi arvutusi. Tihti võib jääda tabel avatuks ning roboti käivitusel ärireeglite kättesaamine muutub keeruliseks kui mitte võimatuks. Sama probleem esineb ka andmete logimisel.

Kõik projektid on arendatud viisil, kus ühte robotit on võimalik käivitada samaaegselt mitmel serveril. Üks projekt koosneb tihti mitmest väiksemast protsessist, mis ühiselt annab kokku ühe terviku. Käivitades ühe roboti samaaegselt erinevatel serveritel, tekib probleem andmete lugemisel ja info salvestamisel, kuna mõlemal käivitatud robotil on samad konfiguratsioonid ja logifailid. Antud olukorras tekib probleem ärireeglite sisse lugemisel ja tehtud tööde salvestamisel tabelisse. Esimene robot käivitub, avab nõutud failid ja alustab töö tegemist, kuid teine identne robot lõpetab tegevuse, kuna tabeltöötlusfailid on *read-only*[1] režiimis, kus vajalikud funktsioonid pole kättesaadavad.



Joonis 1 Näide erinevatest konfiguratsioonidest ühel tarkvararobotil

Andmed logitakse vastavalt ärinõuetele. Olenevalt protsessist salvestatakse kõik vajalikud andmed äripoolele tabelisse, et roboti tehtud tööd kontrollida ja kinnitada. Sealhulgas leidub väga kriitilisi ja tähtsaid isikuandmeid, mis nõuavad rangemat kontrolli andmete ligipääsu ja hoiustamise osas. Hetkel jälgitakse manuaalse tööga tabelite sisu, vajadusel tühjendatakse ja asendatakse uue failiga. Statistika tegemiseks või vea leidmiseks on andmed jällegi vajalikud, mitte kõik salvestatud muutujad ei ole konfidentsiaalsed ja ei vaja kustutamist. Andmete kogumiseks mõeldud faili nimetatakse

[1] *Read-only* ehk lugemisrežiim

masterfile'ks. *Masterfile* tabel on tihti kasutusel ka roboti enda poolt, et täiendada logisid või kontrollida ja vajadusel filtreerida eelnevalt tehtud tööd (Joonis 2). Lisaks sellele on tabeli tulbad struktureerimata kujul, tulbale on identifitseerimiseks küll tüüp määratud, aga selle mittetahtlikul muutmisel tekib roboti loogikas viga ning protsess lõpetab töö.

Lepingu number	Teataja	Building	Movable	Pet	Liability	Status	Error	Robot Finished
15805303	Mati Mets	TRUE	FALSE	FALSE	FALSE	TRUE		05.05.2023 06:45
	Mari Mets	FALSE	TRUE	FALSE	FALSE	FALSE	1) Business Exception - Lepingut ei leitud	06.05.2023 06:46
12321175	Kardi Mets	FALSE	TRUE	FALSE	FALSE	FALSE	2) Technical Exception - Manuste allalaadimine ebaõnnestus	07.05.2023 06:47

Joonis 2 Tabeltöötlusprogrammi exceli näide *masterfile'*st

Tabeltöötlusprogrammidega on kaasnevaks probleemiks veergude järjekord, mille lugemisel robot järgib rangeid reegleid. Uute veergude lisamisel või olemasolevate muutmisel tuleb teha lisaarendusi funktsionaalsuses ning andmete korrigeerimisel tuleb järgida kindlaid protseduure, et vältida vigade tekkimist. See on aeganõudev ja võib kaasa tuua suuremaid riske vigade tekkeks, eriti kui tegemist on keeruliste või suuremahuliste tabelitega. Seda just eriti olukorras, kus versioonihaldus on puudulik. Kui tarkvararobotite eelis on just kiirus ja efektiivsus, siis iga täiendava projektiga tekib palju manuaalset tööd, mis pikemas perspektiivis pole enam hallatav. Kõik robotite poolt kasutatavad failid on paigutatud ettevõtte võrgukettale, mis on turvalisuse eesmärgil piiratud Windowsi turvareeglite abil. See tähendab, et igale projekti kaustale tuleb luua eraldi reeglid ja vajadusel muuta, mis tähendab järjekordselt lisatööd kasutajatoe osakonnale.

2.3 Eesmärk

Töö eesmärk on välja töötada tarkvaralahendus, mis võimaldab automatiseerida erinevate tabeltöötlusprogrammi failide töötlemist, andmete kogumist ja integreerida need tõhusalt tarkvararobotite protsessidesse, tagades seeläbi tööprotsesside sujuvama ja kiirema kulgemise ning vähendades inimliku vea võimalust. Lahendus peab olema lihtsasti kasutatav, kohandatav ja võimaldama statistika tegemist, et aidata arendusotsuste tegemist ja erinevate andmeanalüüside koostamist. Konfidentsiaalsete andmete tõttu peab olema tarkvara turvaline, et vältida kolmandate isikute juurdepääsu volitamata andmetele. Lisaks sellele tagab rakendus andmete ühtsuse, asendab struktureerimata tabeltöötlusprogrammid objektoritenteeritusega, mis muudab kogu protsessi kiiremaks ja efektiivsemaks samal ajal säästes töötajate tööaega.

3 Analüüs

Projekt alustati funktsionaalsete nõuete kogumisega ja põhinõuete määramisega, mida rakendus peab täitma. Toimusid koosolekud erinevate robootika osapoolte vahel, kus tutvustati ideed ja täiustati nõuete kogumit. Pärast funktsionaalsete nõuete kindlaksmääramist, analüüsiti erinevaid tehnoloogiate valikuid, seati üles töökeskkond ja kavandati arhitektuur. Selles jaotises selgitatakse nõudeid ning iga funktsionaalsuse kohta on lisatud kasutuslood ja stsenaariumid.

3.1 Andmetest lähtuv analüüs

Tarkvararobotite poolt kasutatav andmestik on erinev ja sellest lähtuvalt tuleb eelnevalt paika panna olemid. Esmalt on teada, et rakendust hakkavad kasutama erinevate õigustega kasutajad. See tähendab, et andmebaasis peaksid olema välja toodud kasutajad ja nende õigused.

Iga tarkvararobot on oma olemuselt erinev ning selle juhtimiseks on vajalikud erinevad seaded ja andmed. See tähendab, et iga roboti kohta luuakse projekti olem, kuhu kogutakse kokku vajalik info. Sellise lähenemise abil on võimalik kontrollida ligipääsu projektide ja kasutajate vahel ning see aitab vältida andmete lekkimist teistele osapooltele. Kõiki õigusi saab vastavalt lisada, muuta või eemaldada.

Üks olulisemaid andmete kategooriaid on tarkvararobotite poolt tehtud töö tulemuste info, et vastutavad isikud saaksid kiirelt veateateid või jälgida roboti tööprotsessi. See andmekogum on samuti kasulik roboti lisaarenduste ja statistika analüüsimisel. Igal tarkvararobotil võib olla mitu erinevat põhilogi tüüpi andmete kogumiseks. See tähendab, et andmebaasis peab olema kogum erinevatest logide tüüpidest. Olemit nimetatakse põhilogiks. Lisaks sellele on vaja koguda üldiseid andmeid, mis aitavad tuvastada vigasid, hoida järge muudatustel ja andmete liikumisel. Iga põhilogi tüüp vastaks ühele tabeltöötlusfailile.

Et tarkvararobotid oleksid hallatavad erinevate vastutavate poolt, on vaja lisaks erinevaid konfiguratsioone, mis aitavad efektiivselt ja kiirelt muudatusi teha. Igal robotil võib olla

erinevat tüüpi konfiguratsioone, alustades üherealistest kuni mitmerealisteni, mis ei ole piiratud ridade ega veerude arvuga (Tabel 1).

Tabel 1 Olemite nimetused ja selgitused

Olemi nimetus	**Selgitus**
Kasutaja (*Account*)	Kasutaja olem on tarkvararobotite jaoks oluline komponent, mis sisaldab kasutajakontode andmeid ja seadeid. Iga kasutajakonto omab unikaalset identifikaatorit, mis võimaldab eristada ja juhtida erinevate kasutajate õigusi ja juurdepääsu erinevatele funktsioonidele ning ressurssidele.
Konfiguratsioon (*Configuration*)	Konfiguratsioon on tarkvararoboti seadistuste kogum, mis määrab roboti käitumise ja töötingimused. Antud olem on oluline, kuna see võimaldab tarkvararoboti seadeid lihtsasti hallata ja muuta vastavalt vajadusele. See võimaldab äripoolel muutvates tingimustes võimalikult kiirelt kohaneda.
Logi (*Log*)	Logid on kirjed, mis säilitavad teavet rakenduse toimimise kohta, sealhulgas tehtud toimingud, vigade teated, tõrked, hoiatused, kasutajate tegevused. Logide analüüs võimaldab arendajatel, süsteemiadministraatoritel ja teistel spetsialistidel hinnata rakenduse toimivust, leida ja lahendada vigu ning parandada süsteemi tervikuna. Lisaks sellele on logid olulised ka turvakaalutlustel, kuna need võimaldavad jälgida, kes, millal ja millist tüüpi toiminguid rakenduses teinud on.

Olemi nimetus	**Selgitus**
Projekt (*Project)*	Projekti olem on kogum, mis hoiab endas kõiki projekti komponente, sealhulgas konfiguratsioone, logisid ja teisi andmeid, mis on roboti toimimiseks vajalikud. See võimaldab kasutajatel hallata projekti erinevaid osasid, sealhulgas seadistusi ning logisid, mis aitavad jälgida roboti toimivust, statistikat ja leida tõrkeid. Projekti andmete kuvamine on piiratud õigustega kasutajate vahel, mis tagab andmete turvalisuse ja konfidentsiaalsuse ning võimaldab andmetele juurdepääsu ainult nendele kasutajatele, kellel on selleks vastavad õigused.
Põhilogi (*Masterlog)*	Põhilogid on olulised kirjed, mis koguvad andmeid tarkvararobotite tehtud tööde kohta. Vastavalt vajadusele on võimalik luua erinevat tüüpi põhilogisid, mida saab vastavalt seadistada tulpade nimede ja andmete konfidentsiaalsuse tasemel. Põhilogide abil on võimalik jälgida tarkvararobotite töökulgu ning vigade korral kiirelt reageerida, et vastavalt vajadusele parandada või lõpetada töö. Põhilogid on eriti olulised robotite vastutavatele isikutele, kuna need võimaldavad tööprotsessil silma peal hoida ning tagada, et kõik toimub nii nagu peaks.

3.2 Kasutajakogemus

Kasutajakogemus ehk UX[1] on see, kuidas kasutaja süsteemi või tarkvaraga suhtleb. See koondab inimese arusaama kasutusmugavusest ja tõhususest. Kasutajakogemuse parandamine on enamiku ettevõtete, disainerite ja loojate jaoks oluline samm toodete

[1] UX – *User Experience* [42]

loomisel ja täiustamisel, sest negatiivne kasutajakogemus võib vähendada toote või teenuse kasutamist ja seeläbi ka soovitud positiivseid mõjusid [10].

Kasutaja on ettevõtte sisene klient ehk töötaja, kelle tööülesanne on automatiseeritud protsessi kvaliteedi jälgimine, erandite lahendamine ja vajadusel seadistuste uuendamine. Enamasti on kliendiks arendatud tarkvararoboti tellija esindaja ehk äripool. Ettevõtte sisese kokkuleppe järgi on nemad protsessi üldise toimimise eest vastutavad ka pärast automatsiooni loomist. Ettevõtte näitel võib kasutajaks olla erineva tausta, kogemuse ja erialaga inimesi. Näiteks telemüügi, klienditeeninduse ja -toe osakonnad või erinevad juhtivad positsioonid.

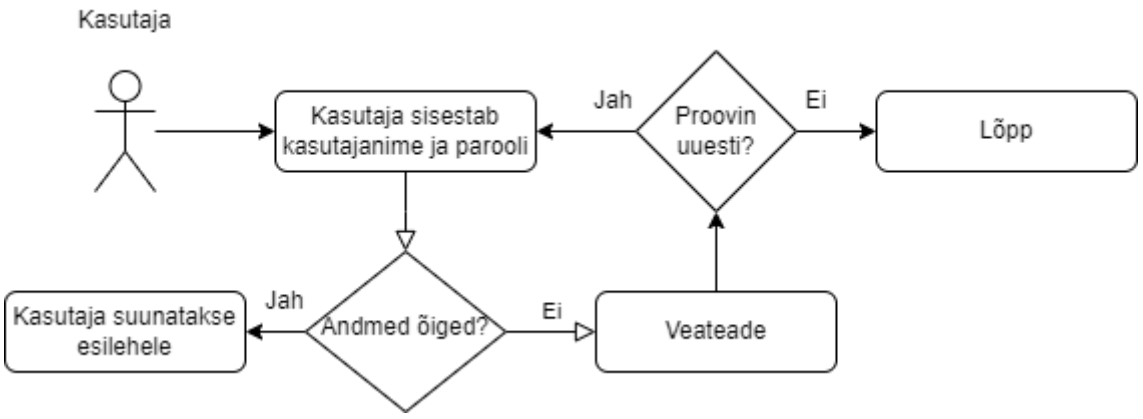
3.3 Rakenduse kasutajad ja projektid

Rakenduse kasutamiseks on esmalt vaja luua kasutajakonto. Seejärel on võimalik lisada uusi projekte, kuhu saab koguda andmeid ja seadistada robotite konfiguratsioone. Projekti loonud kasutaja omab selle üle omanikuõigust ning tal on võimalik kutsuda teisi liikmeid ning määrata vastavad rollid. Tänu sellele funktsionaalsusele saavad kasutajad koostööd teha ning ühiselt projekte hallata ja täiendada.

Iga kasutusjuht eeldab juba kasutaja autentimist ehk sisselogimist kontole, sest ilma selle tegevuseta pole võimalik tarkvara kasutada ega projekte luua, muuta või vaadata. Kasutaja peab esmalt looma konto ja sisse logima, et saada juurdepääs täiendavatele funktsionaalsustele.

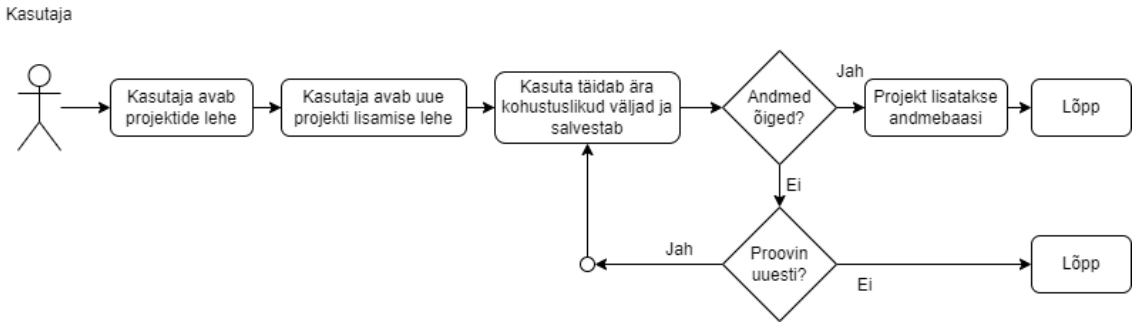
Kasutuslood

- „Kasutajana saan registreerida endale konto, et saada ligipääs tarkvara kasutamisele.“
- „Kasutajana saan tarkvarasse sisse logida kasutades endale mõeldud kasutajanime ja parooli. Kui kasutaja on valideeritud, suunatakse mind esilehele (Joonis 3).“



Joonis 3 Kasutaja sisselogimise plokkskeem

- „Kasutajana saan lisada uue projekti, et luua valmis keskkond tarkvararoboti jooksutamiseks. Projekti lisamiseks avan projekti lehe, avan projekti lisamiseks mõeldud lehe, täidan ära vastavad väljad ning salvestan toimingu (Joonis 4).“



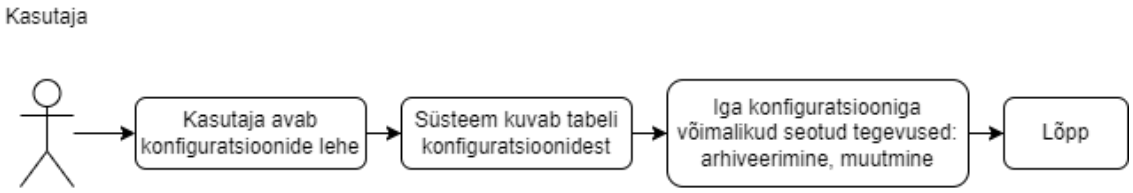
Joonis 4 Projekti lisamise plokkskeem

3.4 Konfiguratsioonide lisamine ja haldamine

Rakenduse üks olulisematest funktsionaalsustest on robotite seadistamisel ja nende haldamisel. Rakenduse kaudu on võimalik luua erinevaid konfiguratsioone, mis visuaalselt on esitatud tabelina, millel on veerud ja read. Iga robot on oma loogika ja sisu poolest erinev, seega nõuab see väga dünaamilist lähenemist muutujate osas. Uue konfiguratsiooni lisamisel tuleb määrata nimi, selgitus ja vastavad muutujad. Kõik tegevused on piiratud õigustega ja neid logitakse turvalisuse ja statistika eesmärgil. Loodud konfiguratsioonid on tarkvararoboti tööprotsessi aluseks. Konfiguratsioonide haldamine käib aktiivses projektis.

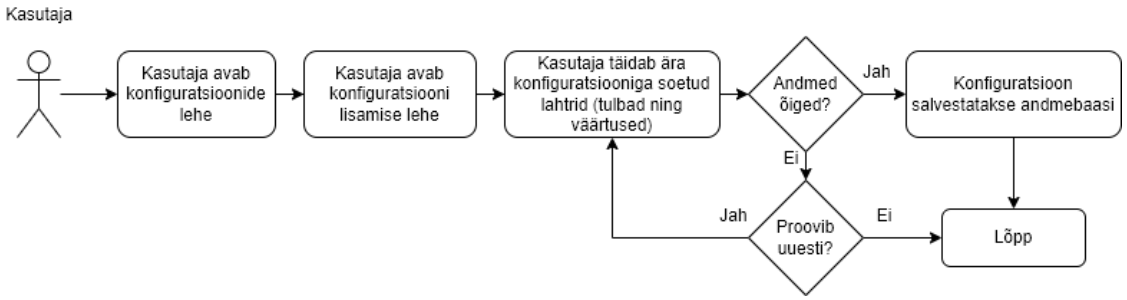
Kasutuslood

- „Projekti kasutajana (edasi kasutajana) saan vaadata projekti konfiguratsioone (Joonis 5). Esmalt on vaja aktiveerida vastav projekt, mille konfiguratsioone soovin vaadata. Järgmisena avan konfiguratsioonide lehe, mis kuvab mulle tabeli antud projekti konfiguratsioonidest. Iga konfiguratsiooniga on võimalik teostada järgmiseid toiminguid: konfiguratsiooni muutmine, konfiguratsiooni arhiveerimine.“



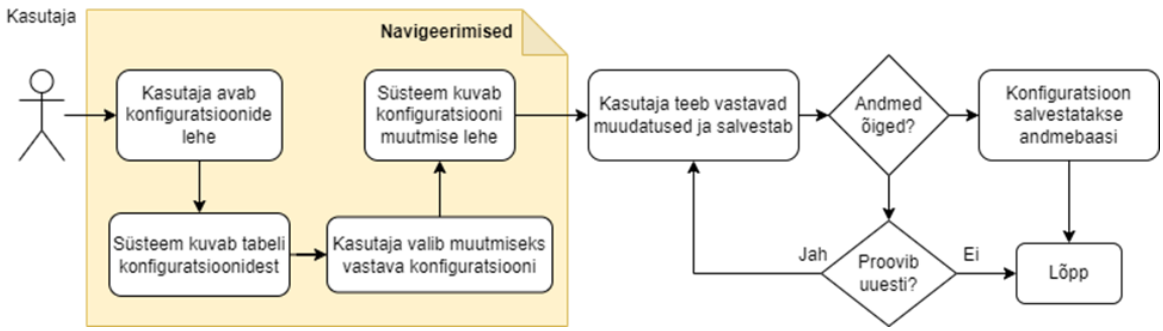
Joonis 5 Konfiguratsioonide vaatamise plokkskeem

- „Kasutajana saan lisada uue konfiguratsiooni. Konfiguratsiooni lisamiseks avan konfiguratsiooni lehe ning uue konfiguratsiooni lisamise lehe. Täidan väljad ja salvestan toimingud“ (Joonis 6)”



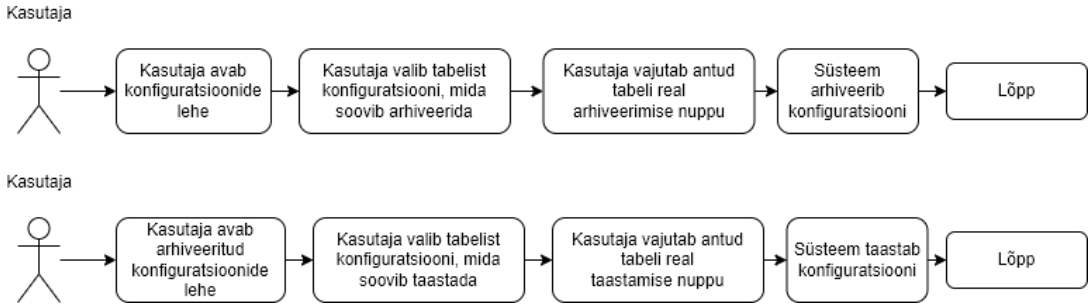
Joonis 6 Konfiguratsiooni lisamise plokkskeem

- „Kasutajana saan uuendada olemasolevaid konfiguratsioone (Joonis 7). Konfiguratsioonide uuendamiseks avan konfiguratsiooni lehe. Tabelisse kuvatavatest konfiguratsioonidest valin muutmiseks vastava konfiguratsiooni. Süsteem kuvab konfiguratsiooni muutmise lehe. Täidan ära vastavad väljad konfiguratsiooni täiendamiseks. Salvestan toimingud“



Joonis 7 Konfiguratsiooni muutmise plokkskeem

- „Kasutajana saan arhiveerida konfiguratsioone. Arhiveerimiseks avan konfiguratsioonide lehe, leian tabelist vastava rea ning vajutan arhiveerimiseks vastavat nuppu. Konfiguratsiooni taastamiseks avan arhiveeritud konfiguratsioonide lehe, leian tabelist vastava rea ja vajutan selleks vastavat nuppu (Joonis 8)„



Joonis 8 Konfiguratsioonide arhiveerimise ja taastamise plokkskeem

3.5 Andmete logimine

Rakenduse teine osa keskendub andmete kogumisele ja statistika koostamisele. Kõik roboti poolt tehtud tööd kuvatakse tabelis, mida kutsutakse *masterlogi'*ks (edaspidi põhilogi). Iga tehtud töö koosneb kindlatest ja dünaamilistest muutujatest. Identifitseerimiseks on igal päringul oma ID. Protsessi käigus võib ilmneda ärilisi või tehnilisi vigu, mis võivad takistada robotil töö lõpetamist. Töö staatus määratakse vastavalt sellele, kas robot suutis töö edukalt lõpule viia või mitte. Kui töö on edukalt lõpule viidud, siis määratakse staatuseks "Completed". Kui aga töö käigus esines probleeme ja töö jäi pooleli, siis määratakse staatuseks "Exception". Selle teabe põhjal saavad vastavad osapooled vea kiirelt likvideerida või töö käsitsi lõpetada.

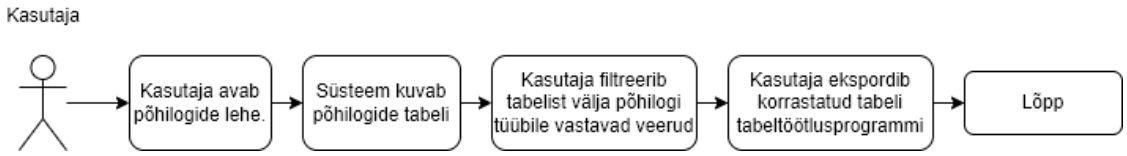
Dünaamilised muutujad võivad olla konfidentsiaalsed ning täpne struktuur määratakse seadetes. Konfidentsiaalsete väljade sisu tühjendamine kindlatel ajavahemikel on vajalik selleks, et tagada andmete turvalisus, kaitsta tundlikku teavet volitamata juurdepääsu eest tagades seadustest tulenevate direktiivide täitmist. Andmete hoiustamine ilma vajaduseta võib tekitada tarbetut riski andmete lekkimiseks või kuritarvitamiseks, seetõttu on oluline järgida konfidentsiaalsusega seotud reegleid ja tühjendada konfidentsiaalsed väljad vastavalt kindlatel ajahetkedel. Samal ajal jäävad mittekonfidentsiaalsed väljad avatuks, et võimaldada analüüsi tegemist ja statistika loomist. Lisaks põhilogile, salvestatakse ka üldiseid andmeid, mis sisaldavad protsessi käivituse ja lõpetamise logisid, metaandmeid ning vigade logisid, mis hõlmavad näiteks autentimisprobleeme ja automatiseeritud keskkonna paroolide aegumist. Need võimaldavad kasutajatel jälgida protsessi käiku ja tuvastada võimalikke probleeme.

Statistika on oluline automatiseeritud protsesside analüüsimiseks ja optimeerimiseks. Logide ja põhilogide põhjal saab luua statistilisi analüüse, mis annavad ülevaate tarkvararobotite töökulust, sealhulgas vigadest ja erinevate tööetappide kulgemisest. Olenevalt tarkvararobotite tööst, saab analüüsida klientide käitumist ja eelistusi. See võimaldab ettevõttel kohandada oma teenuseid vastavalt klientide vajadustele ja soovidele. Lisaks sellele on statistika abiks äriotsuste tegemisel. Andmete analüüs aitab näha mustreid ja trende ning prognoosida tulevikku. Eelnevalt olemasoleva statistika analüüsimisel on Autor leidnud ka alussüsteemides vigu, mis tulid välja tänu roboti erandite detailse analüüsi läbiviimise. Täiendavalt aitab loodud tööriist statistiliselt näidata olemasolevate tarkvararobotite poolt tehtud töö mahtu ja kõrvutada seda tavatöötaja tööajaga.

Kasutuslood

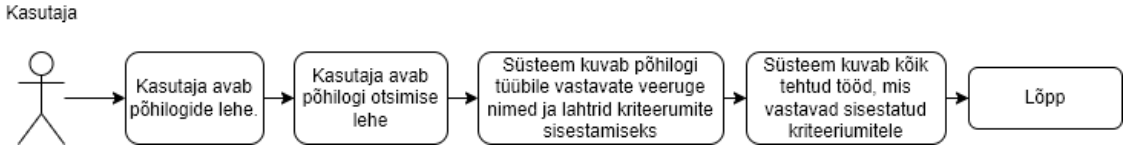
- „Kasutajana saan vaadata robotite poolt genereeritud logisid, et jälgida protsessi käivitumist ja lõpetamist, metaandmeid ning võimalike ootamatute vigade esinemist.“

- „Kasutajana saan vaadata robotite poolt tehtud töid, et jälgida tehtud tulemusi ja nende staatusi. Tabelisse on kuvatud staatilised ja dünaamilised veerud. Staatiliste veergude all on mõeldud järgnevaid veerge: aeg, staatud, erandid. Dünaamilised veerud on seadistatav põhilogi tüübi seadistamisel. (Joonis 9)“



Joonis 9 Põhilogide vaatamise ja tulpade filtreerimise plokkskeem

- „Kasutajana saan filtreerida tabelit erinevate muutujate järgi, et leida vajalikku teavet kiiremini ja lihtsamalt. Kasutajana saan otsida täpsete kriteeriumite järgi tehtud töid. Otsimiseks avan põhilogide otsimise lehe, kus kuvatakse kõik põhilogi tüübile vastavad veerud. Igat veergu saan vastavalt seadistada, et süsteem tagastaks kõige täpsema tulemuse. (Joonis 10)“



Joonis 10 Põhilogide otsimise plokkskeem

- „Kasutajana saan vaadata tehtud tööde statistikat, et saada ülevaade tehtud tööde arvust, nende staatustest ja muudest olulistest muutujatest. Statistika vaatamiseks avan „Statistika“ lehe. Süsteem kuvab esmased statistilised väärtused, milleks on järgnevad: tehtud tööde arv, vigade arv kokku, tehniliste vigade arv, äriliste vigade arv, tehtud tööde õnnestumise tõenäosus, keskmine roboti tööaeg päevas/nädalas/kuus.„

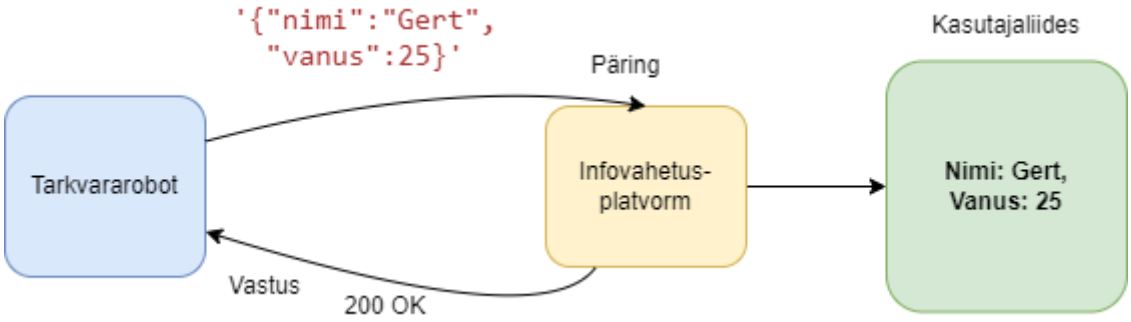
3.6 Tehnoloogiate valik

Iga tarkvaraarenduse projekti käigus tuleb valida sobivad tehnoloogiad nii ees- kui ka tagarakenduse arenduseks. Tehnoloogia valikul on määrav roll projekti edukusele, sest sobivate tehnoloogiate kasutamine võib kiirendada arendusprotsessi ja vähendada potentsiaalsete vigade riski.

Selles peatükis uuritakse erinevaid tehnoloogiaid, mis sobivad projekti vajaduste ja eesmärkidega. Eesrakenduste osas vaadeldakse mitmeid populaarseid raamistikke nagu React, Angular ja Vue.js, samuti ka traditsioonilisi tehnoloogiaid nagu HTML, CSS ja JavaScript. Tagarakenduse osas analüüsitakse .NET, Node.js ja Java tehnoloogiaid. Tuleb

meeles pidada, et tehnoloogiate valik sõltub ka ettevõtte vajadustest, meeskonna kogemustest ja projekti ulatusest.

Üheks suuremaks nõudeks tehnoloogia valikul on REST API kasutamise võimalus. REST (*Representational State Transfer*) API on üks viis rakenduste programmeerimiseks ja suhtlemiseks serveriga [11]. See võimaldab andmeid edastada üle veebi standardsete HTTP protokollide kaudu [12]. REST API-ga suhtlemine toimub tavaliselt JSON formaadis, kuid võib olla ka XML või muus vormingus [13]. REST API kasutus on tähtis mitmel põhjusel. Üks olulisemaid põhjuseid on see, et REST API võimaldab suhelda erinevate süsteemidega, sealhulgas tarkvararobotitega (Joonis 11). Näiteks konfiguratsioonide lugemiseks on vaja esmalt teha päring, mis tagastab robotile kasutaja poolt määratud seadistused. Tehtud tööde edastamiseks tuleb kasutada HTTP POST meetodit, et andmed edastada süsteemi andmebaasi salvestamiseks.



Joonis 11 Tarkvararobotite suhtlus loodava tarkvaraga näidis päring

3.6.1 Muud võimalikud tehnoloogiad

Lisaks traditsioonilistele *backend* ja *frontend* raamistikele on olemas ka mitmeid teisi tehnoloogiaid, mis võivad olla kasulikud tarkvara loomisel. Näiteks Microsofti poolt loodud Power Platvormi kuuluvad rakendused Power BI ja Power Apps [14].

Power BI on ärianalüüsi platvorm, mis võimaldab kasutajatel andmeid koguda, analüüsida ja esitada dünaamiliste graafikute kaudu. Power BI toetab erinevate andmeallikate, sealhulgas Exceli, SQL Serveri, SharePointi, Dynamics 365 ja paljude teiste andmebaaside, pilveteenuste ja rakenduste ühendamist ning andmete ühtlustamist ja puhastamist. Lisaks sellele pakub Power BI võimalusi andmete koondamiseks ja esitamiseks aruannete, tabelite, graafikute ja kaartide abil, mida saab jagada

organisatsiooni erinevate osakondade ja kasutajate vahel. Power BI on kasutajasõbralik ja seda saavad kasutada nii algajad kui ka edasijõudnud kasutajad.

Power Apps on Microsofti loodud tööriist, mis võimaldab kiiresti ja lihtsalt luua ettevõttesiseseid rakendusi, sealhulgas mobiilirakendusi. Antud tööriist kasutab madala kooditasemega lähenemist, mis tähendab, et arendajad saavad luua rakendusi ilma, et neil oleks vaja eelnevat programmeerimiskogemust. Selle asemel on võimalik kasutada loogikapõhiseid funktsioone ja visuaalseid elemente, et ehitada rakendusi andmeallikatele, nagu SharePoint, Dynamics 365 ja SQL Server. Power Apps pakub mitmeid funktsioone, sealhulgas valmis komponente, mis võimaldavad arendajatel lisada rakendustele graafikuid, andmetabeleid, vorme ja muid interaktiivseid elemente.

Integreerimine Power BI ja Power Appsi vahel toimub tavaliselt andmeallika tasemel. Andmeallikad, mis on loodud Power BI-s, saab kasutada ka Power Appsi poolt loodud rakenduste jaoks. Seega saavad Power Appsi kasutajad kasutada Power BI-s loodud aruandeid ja graafikuid, et teha paremaid ning kiiremaid otsuseid. Lisaks võimaldab Power BI REST API Power Appsi arendajatel luua spetsiaalseid integreeritud lahendusi, tänu millele on võimalik visualiseerida andmeid otse Power Appsi rakendustes. See tähendab, et andmed, mida kasutajad näevad Power Appsi rakendustes, pärinevad otse Power BI-st, tagades seeläbi, et andmed on alati ajakohased ja täpsed.

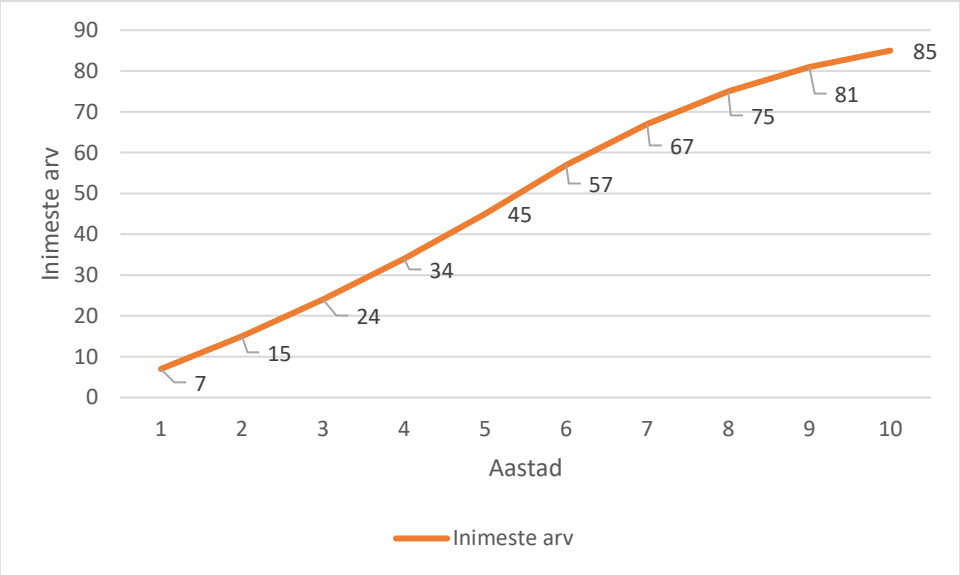
Mõlemal tööriistal on mitmeid erinevaid hinna- ja litsentsimudeleid ning saadaval on ka tasuta versioonid, kuid need piiravad andmemahtu ja funktsionaalsusi. Power platvormide hinnastamine sõltub peamiselt kasutajate arvust, hinnad algavad 10 eurost kuni 20 euroni inimese kohta (Tabel 2). Alljärgnevates näidetes pole arvesse võetud ettevõtetele pakutavaid mahupõhiseid soodustusi.

Tabel 2 Tehnoloogia hinna ja litsentside arvu võrdlus

Tehnoloogia	Litsentsi hind	Inimeste arv	Hind kokku / kuus
Power BI	20€ / inimene*	25	500€
Power Apps	10€ / inimene*	25	250€

Käesoleval ajahetkel on jõutud olukorda, kus tarkvararobootika tiim koos arendajate ja robotite eest vastutavate osapooltega on kasvanud kolme aastaga umbes 25 inimeseni. Osapoolte all on mõeldud nii arendajaid, analüütikuid ja tarkvararobotite eest vastutavaid, kes igapäevaselt robotite tööd kontrollivad ja parandavad. Lisaks tuleb arvestada arendustiimi laienemist, tarkvararobootika nõudluse suurenemist ettevõttes ning võimalikke muudatusi hinnastamise mudelites.

Kui arvestada praegust osapoolte arvu ning eeldada selle tõusu järgmise 7 aasta jooksul, siis võib oletada, et kasutajate arv kasvab 85 kuni 100-ni (Joonis 12). Prognoosis on võetud arvesse seda, et esimestel aastatel on vaja rohkem uusi litsentse ning hiljem hakkavad erinevate projektide osapooled kattuma ja uute litsentside vajadus väheneb. See tähendab, et järgmise seitsme aasta Power BI litsentside kogukulu on praeguste hindade põhjal 106 500 eurot. Arvestades tarkvararobotite kasvavat hulka, on nende tööriistade kasutamine rahalises mõttes ebaotstarbekas.



Joonis 12 Tarkvararobootika osapoolte arvu muutus aastate lõikes

3.6.2 Teenusrakenduse tehnoloogiad

Teenusrakenduse ehk *backend*'i valik on üks olulisemaid otsuseid, mida projekti meeskond peab tegema. Teenusrakendus on osa tarkvararakendusest, mis töötab serveris ja vastutab andmete haldamise, loomise ja kustutamise eest. Tagarakenduse valik sõltub mitmest tegurist, nagu kogemus, projekti ulatus, nõudmised jõudlusele ja turvalisusele, kättesaadavad vahendid ning kulud.

.NET Core Web API - Microsofti välja töötatud avatud lähtekoodiga raamistik, mida kasutatakse veebirakenduste loomiseks ja arendamiseks .NET platvormil [15]. See pakub võimalust luua kõrge jõudlusega, skaleeritavaid ja turvalisi veebirakendusi, mida saab kiirelt integreerida erinevate platvormide ja teenustega. .NET Core Web API on populaarne valik paljude ettevõtete jaoks, eriti neile, kes juba kasutavad .NET tehnoloogiat. Raamistik on saanud palju kiitust oma paindlikkuse ja jõudluse poolest ning seda kasutatakse laialdaselt paljudes valdkondades, sealhulgas e-kaubanduses, panganduses ja tervishoius [16].

Node.js - populaarne tagarakenduse raamistik, mis põhineb JavaScriptil [17]. Node.js töötab serveripoolse rakendusena, võimaldades JavaScripti abil serveri- ja kliendipoolse koodi kombineerimist. See on eriti kasulik veebirakenduste arendamisel, kus suur hulk andmeid liigub pidevalt serveri ja klientide vahel. Node.js peamised eelised on suurepärane jõudlus ja võimekus töödelda suurt hulka paralleelseid ülesandeid.

Java - objektorienteeritud programmeerimiskeel, mis võimaldab luua platvormist sõltumatut tarkvara [18]. See tähendab, et sama koodibaasi saab kasutada erinevates operatsioonisüsteemides, näiteks Windowsis, Linuxis ja MacOS-is. Java on laialdaselt kasutusel serveripoolsete rakenduste arendamisel, sealhulgas veebirakendustes ja andmebaasi töötluses. Java peamised eelised on suur kogukond ja rikkalik valik raamistikke ning tööriistu, mis lihtsustavad arendust ja tõstavad tootlikkust.

Hetkel on välja toodud kolm võimalikku alternatiivi, kuid veebirakenduste arenduseks leidub mitmeid teisi lahendusi. Antud lahendused on valitud analüüsi jaoks autori eelneva kokkupuute ja kogemuse põhjal. Lisaks sellele on vaadatud ka tarkvarade uuenduslikkust ja ajaga kaasas käimist, suurt kasutajaskonda, pidevad uuendused ning vigade parandused. Vaadates tabelit (Tabel 3) on näha, et kõik välja valitud tarkvarad saavad regulaarselt uuendusi. Arvestades rakenduse skoopi ja tarkvarade vahelist kogemust valis diplomitöö teenustarkvaraks .NET Core.

Tabel 3 Teenusrakenduse valikud

Tehnoloogia	Kogemus	Uuenduslikkus
.NET Core	Väga hea	NET 7.0 viimane versioon - 11.04.2023. NET 8.0 preview versioon väljas
Java	Rahuldav	Viimane versioon – 21.03.2023
Node.js	Rahuldav	Viimane versioon – 01.04.2023

3.6.3 Kliendirakenduse tehnoloogiad

Kliendirakenduste tehnoloogiad ehk *frontend*-tehnoloogiad on olulised veebirakenduste arendamisel, kuna need määravad, kuidas rakendus kasutajale esitletakse ning kuidas kasutaja sellega suhtleb. Antud alapeatükk annab ülevaate populaarsetest raamistikest nagu React, Angular ja Vue.js. Tehnoloogiate valik sõltub konkreetse projekti eesmärkidest ja vajadustest ning autori kogemusest.

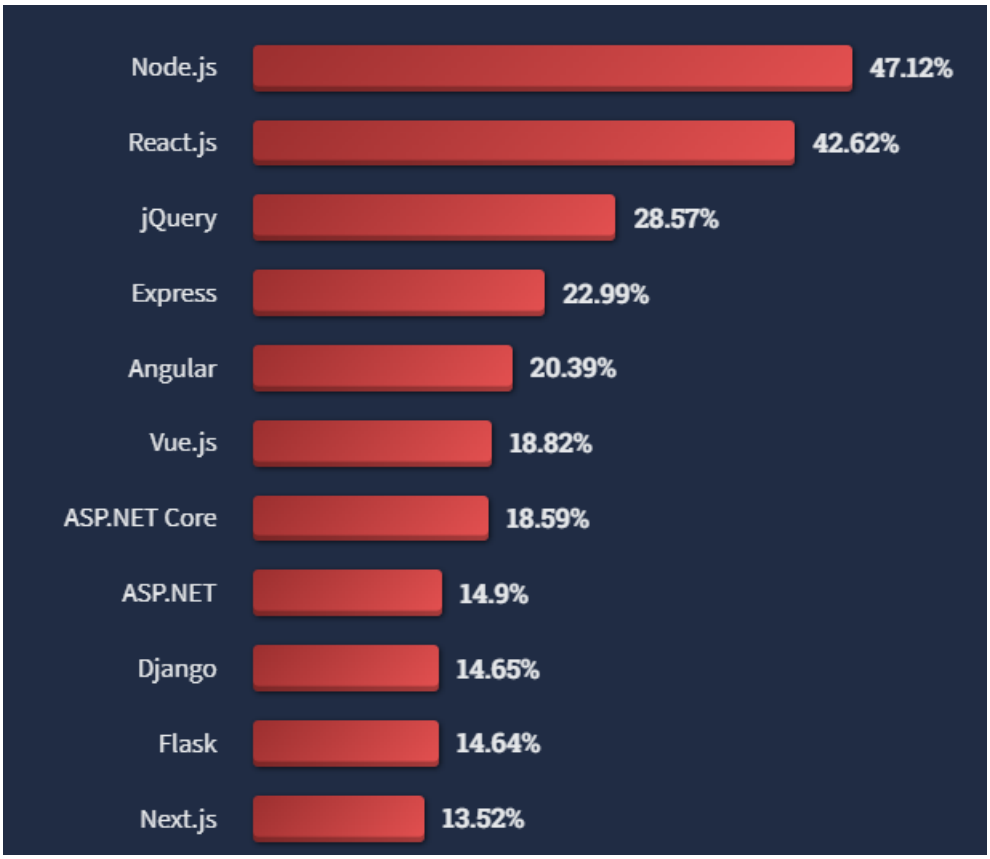
React, Vue.js ja Angular on kõik populaarsed JavaScripti raamistikud, mida kasutatakse laialdaselt veebirakenduste arendamisel. React on loodud Facebooki poolt ning põhineb komponentidel, mis on omavahel seotud ning võimaldavad andmevoo sujuvat haldamist, muutes rakenduse arendamise lihtsamaks ja paindlikumaks [19]. See on väga populaarne raamistik, mida kasutavad paljud suured ettevõtted nagu Facebook, Instagram, Netflix ja Airbnb. Lisaks võimaldab React luua mobiilirakendusi React Native abil, mis on väga populaarne valik mobiilirakenduste arendamiseks.

Vue.js on kaasaegne JavaScripti raamistik, mis on mõeldud veebirakenduste loomiseks [20]. Sarnaselt React'le, kasutab Vue.js samuti komponentpõhist lähenemist, mis muudab rakenduse struktuuri selgemaks ja loogilisemaks. Kuid erinevalt React'ist, keskendub Vue.js rohkem paindlikkusele ja lihtsusele, pakkudes lihtsaid ja intuitiivseid lahendusi keeruliste probleemide jaoks.

Angular on Google'i loodud raamistik, mis sisaldab endas kõiki vajalikke tööriistu, et arendada dünaamilisi veebirakendusi. Angular kasutab TypeScripti ning võib alguses olla

raskem õppida võrreldes Reacti ja Vue'ga, kuid pakub terviklikku ja turvalist raamistikku veebirakenduste arendamiseks.

Stack Overflow on üks kõige populaarsemaid platvorme programmeerimisega seotud küsimuste ja vastuste leidmiseks, kus arendajad saavad küsida küsimusi, jagada oma teadmisi ja lahendada probleeme, mis tekivad tarkvaraarenduses. Antud platvormil on laialdane ja aktiivne kogukond, kes aitab üksteist probleemide lahendamisel. Platvormi 2022 aasta uuringus selgus (Joonis 13), et kõige populaarsemaks kliendipoolse rakenduse raamistikuks on React.js (42.62%), seejärel Angular (20.39%) ja Vue.js (18.82%) [21].

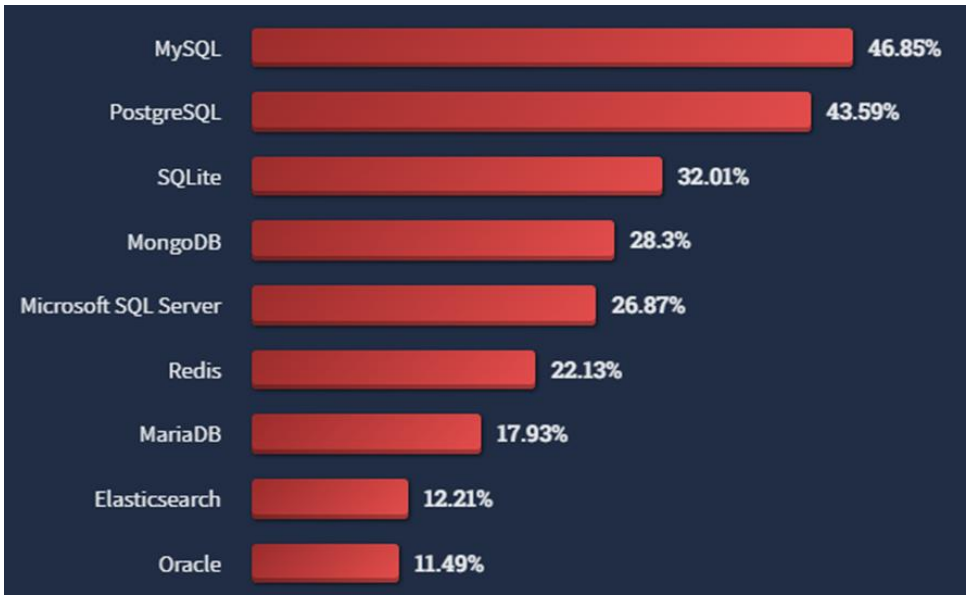


Joonis 13 StackOverflow 2022 uuringu tulemused

Üldiselt sobiksid kõik raamistikud loodava veebirakenduse põhjaks. Antud töös eelistatakse raamistikku React, kuna sellega on autoril eelnev kokkupuude rakenduste arendamisel. Lisaks sellele toetab valikut ka mobiilirakenduste arenduse võimalus React Native abil.

3.7 Andmebaasi tehnoloogiad ja valik

Andmebaasid on oluline osa enamik tarkvarasüsteemidest. Eesmärk on võimaldada andmete tõhusat haldamist, säilitamist ning nende kättesaadavaks tegemist vastavalt vajadusele [22]. Andmebaasid võivad olla erinevat tüüpi, näiteks relatsioonilised, mitte-relatsioonilised, objekt-orienteeritud ja hierarhilised andmebaasid. Igal süsteemil on omad unikaalsed omadused, mis võimaldavad andmeid efektiivselt ja turvaliselt talletada. Stack Overflow 2022 küsitlusest selgus, et populaarsemad andmebaasid on kas relatsioonilised või mitterelatsioonilised andmebaasid. Kõige populaarsemad olid just MySQL (46.85%), PostgreSQL (43.59%), SQLite (32.01%) ja MongoDB (28.3%). (Joonis 14). Nendest valikutest esimesed kolm on just relatsioonilised ja MongoDB mitterelatsiooniline andmebaas. Eraldi uuriti veel statistikat veebilehelt Db-Rankings [23], mis järjestab andmebaasihaldussüsteeme erinevate kriteeriumite järgi nagu otsingumootori tulemuste arv süsteemide nimede otsimisel, Google Trend, Stack Overflow arutelud, tööpakkumised ja profiilide arvud erinevates võrgustikes nagu LinkedIn. Pingerida uuendatakse kord kuus. Antud statistika järgi on populaarsemad andmebaasid järgnevad: Oracle, MySql, Microsoft SQL Server, PostgreSQL, MongoDB.



Joonis 14 Stack Overflow 2022 aasta uuringu andmebaaside tulemused

Toetudes mõlema veebilehe uuringutulemustele, jätkatakse töös relatsioonilise ja mitterelatsioonilise andmebaaside uurimisega.

3.7.1 Relatsiooniline vs mitterelatsiooniline andmebaas

Relatsiooniline andmebaas[1] on andmebaasi tüüp, mille struktuur koosneb tabelitest [24]. Andmed paigutatakse tabelitesse, millel on kindlaks määratud nimi, iga rida esindab konkreetset kirjet ja veerg vastab omadusele või omaduste rühmale, mis on antud kirje kohta salvestatud. Ühes andmebaasis võib olla mitmeid erinevaid tabeleid, mis võivad viidata teistele tabelitele läbi identifitseerimise. Antud veergu nimetatakse primaarvõtmeks. Relatsioonilised andmebaasid kasutavad struktureeritud päringukeelt[2], mille abil saab andmeid lihtsalt salvestada, hallata ja otsida. Relatsioonilised andmebaasid said alguse 1970. aastatel, mil Edgar F. Codd esitas oma artikli kus ta esitles uut andmemudelit [25]. See mudel sai aluseks esimestele relatsioonilistele andmebaasihaldussüsteemidele nagu näiteks Oracle ja IBM DB2. Kõige populaarsemad relatsioonilised andmebaasid on Oracle, MySQL, Microsoft SQL Server, PostgreSQL, SQLite.

Mitterelatsioonilised andmebaasid (NoSQL) on erinevad relatsioonilisest andmebaasist oma mitte-struktureeritud andmehoidla poolest [26]. See tähendab, et oluliste muudatuste tegemine andmebaasis on oluliselt vähem aega nõudev ja sobib rohkem suurte andmemahtude salvestamiseks. Lisaks sellele on antud andmebaasil võimekus käsitelda mistahes struktureerimata andmeid nagu näiteks videod, pildid või vestlused. Antud kontspetsioon sai alguse 1960. aastate lõpust, kuid ei leidnud palju kasutust enne 21. sajandi algust. Kuna NoSQL andmebaasid ei kasuta SQL-i, nõuavad need käsitsi pärinute programmeerimist, mis lihtsamate päringute puhul võivad olla kiired, aga keerulisemate ülesannete puhul aeganõudvad [27]. Näidisena on välja toodud MondoDB ja SQL päringu võrdlus (Joonised 15 ja 16).

```
db.inventory.find( { $or: [ { status: "A" }, { qty: { $lt: 30 }}]})
```

Joonis 15 MongoDB päringu näide [28]

```
SELECT * FROM inventory WHERE status = "A" OR qty < 30
```

Joonis 16 SQL päringu näide

[1] RDBMS - Relational Database Management System

[2] SQL - *Structured Query Language*

3.7.2 Andmebaasi valik

Rakenduse jaoks andmebaasi valimine on üks esimesi otsuseid, mida loodava rakenduse juures tehakse. Valik sõltub erinevatest teguritest, sealhulgas salvestavate andmete tüübist, rakenduse nõuetest ja arendaja teadmistest. Lähtudes autori eelnevale kogemusele ja populaarsusele, jätkatakse relatsiooniliste andmebaasidega. Lisaks sellele on ettevõttes samuti kasutusel RDBMS [1]tüüpi andmebaasid, mis toetab tehtud valikut.

Sobiva valiku tegemiseks leiti esmalt enimlevinud andmebaasid ja lisati need tabelisse (Tabel 4). Lisaks sellele hindas autor igat andmebaasi enda varasema praktilise kogemuse järgi. Kõikide andmebaasidega peale Oracle oli autoril juba eelnev kogemus olemas. Järgmiseks kriitiliseks kriteeriumiks oli hind ning ülalhoidmiskulud. Kõige kallimaks osutusid Microsoft SQL Server ja Oracle, mistõttu jäid need valikust välja [29]. PostgreSQL ja MySQL andmebaasid on mõlemad vabavaralised ja lihtsasti integreeritavad ASP.NET Core[2] rakendusega [30]. Arvestades asjaolu, et autoril on eelnev hea kogemus PostgreSQLiga, hakatakse ka loodava rakenduse juures kasutama antud andmebaasi.

Tabel 4 Andmebaaside ja autori kogemus

Andmebaas	**Kogemus**
Microsoft SQL Server	Hea
MySQL	Keskmine
Oracle	Puudub
PostgreSQL	Hea

[1] RDBMS - *Relational Database Management System*, Relatsioonilised Andmebaasid

[2] ASP.NET Core - tasuta avatud lähtekoodiga veebiraamistik loodud Microsofti poolt

3.8 Arenduskeskkond

Arenduskeskkonna valikust sõltub, millise efektiivsusega valmivat tarkvara arendatakse. See hõlmab erinevaid tööriistu ja tehnoloogiaid, mida kasutatakse tarkvara arendamiseks, testimiseks ja haldamiseks. Antud alapeatükk annab ülevaate võimalikest tööriistades, mida autor oma töös kasutama hakkab.

Versioonihaldussüsteem on tarkvara, mis on mõeldud tarkvaraarenduses lähtekoodi muudatuste jälgimiseks. See tagab efektiivsuse, kiiruse ja andmete terviklikkuse. Seda on eriti näha arendusmeeskondades, kus arendajad saavad tegeleda samaaegselt ühe projektiga. Populaarseimad tööriistad on GitHub, GitLab ja BitBucket.

GitHub on üks kõige populaarsemaid versioonihalduse tarkvarasid, mida kasutab 2023 [31]. aasta seisuga üle 100 miljoni arendaja ja koosneb 372 millionist repositooriumist. Hinnastamine on antud platvormil erinev, olemas on tasuta variant lõpmatute repositooriumite tegemiseks. Samuti on GitHubil väga head võimalused pideva integratsiooni tagamiseks. Töö autor on eelnevalt pealmiselt kasutanud antud tarkvara ja kasutab antud töös.

IDE ehk integreeritud arenduskeskkonna valikul tuleb arvestada, et need toetaksid eelnevalt valitud programmeerimiskeeli. Üks populaarsem C# ja ASP.NET Core toetav IDE on Visual Studio. Lisaks sellele, on kliendipoolse rakenduse arenduseks üks populaarsemaid IDE'sid Visual Studio Code. Mõlemad tarkvarad on loodud Microsofti poolt ja omavad suurt kasutajaskonda.

Draw.IO/Diagrams.net on tasuta avatud lähtekoodiga platvormiülene graafikute joonistamise tarkvara. Selle liidest saab kasutada diagrammide, näiteks vooskeemide, UML-diagrammide, organisatsiooniskeemide ja võrguskeemide loomiseks [32]. Olemi-suhte diagrammi koostamiseks on kasutusel tarkvara Enterprise Architect [33]. See on visuaalse modelleerimise ja disaini tööriist, mis põhineb UML[1]-il. Platvorm toetab: tarkvarasüsteemide projekteerimist ja ehitamist, äriprotsesside modelleerimist ja tööstuspõhiste domeenide modelleerimist [34].

[1] UML - *Unified Modeling Language*, ühtne modelleerimiskeel

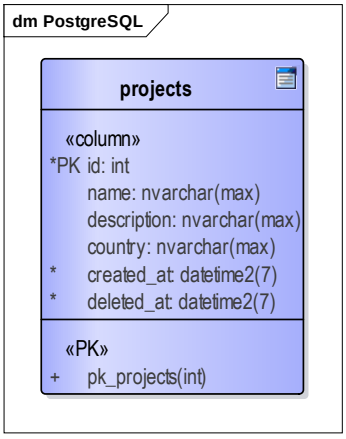
Docker on populaarne avatud lähtekoodiga tarkvara konteinerite virtualiseerimise tehnoloogia, mis võimaldab arendajatel ja süsteemiadministraatoritel arendada, levitada ja käitada tarkvara konteinerites. Tehnoloogia alustas 2013. aastal ja on 2023. aastaks jõudnud 7.3 miljoni kasutajani [35].

3.9 Andmebaasi disain

Andmebaasi disainimisel lähtuti andmestikust, kasutajalugudest ja -juhtudest. Olemi-suhte diagramm loodi tööriistaga Enterprise Architect, mida kasutatakse peamiselt suurte ja keerukate süsteemide modelleerimiseks, projekteerimiseks, dokumenteerimiseks ja juhtimiseks. Andmebaasi disainimise protsessis jälgiti erinevate raamatute nagu „SQL Antipattern“ ja „Practical SQL, 2nd Edition“ välja toodud soovitusi ja reegleid [36] [37]. Välditi andmete dubleerimist, et mitte tekitada lisa keerukust andmete haldamisel. Tabelite nimed koosneksid ainult tähtedest ja väikeste tähtedega, tabelite veerud on väikeste tähtedega ning tühikud on asendatud sümboliga „_“ ehk kasutatud *underscore_name* stiili.

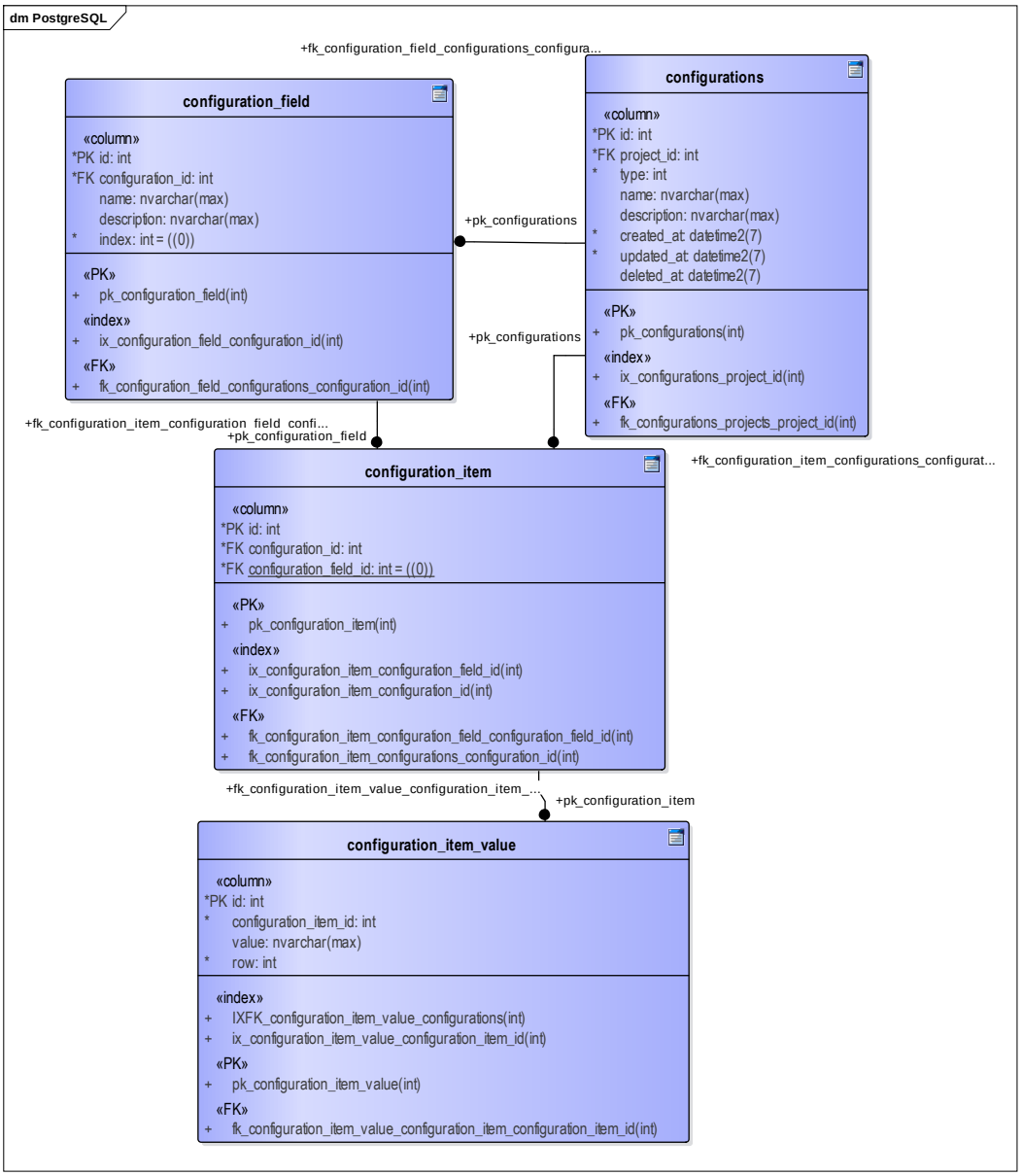
Tabel „Accounts“ sisaldab kõiki kasutajad, mis on autentimise aluseks. Lisaks sellele on vaja tuvastada erinevate olemite vahelisi õigusi, et vältida andmete lekkeid.

Tabelis „Projects“ on kõik projektid. Projektide ja kasutajate vahel on mitu-mitmele seos, seetõttu kasutatakse tabelit „ProjectAccounts“, millel on kaks muutujat, „ProjectId“ ja „AccountId“, millest luuakse primaarnevõti. See väldib olukorda, et üks kasutaja saab olla topelt ühes projektis, kuna primaarvõti tagab unikaalsuse (Joonis 17).



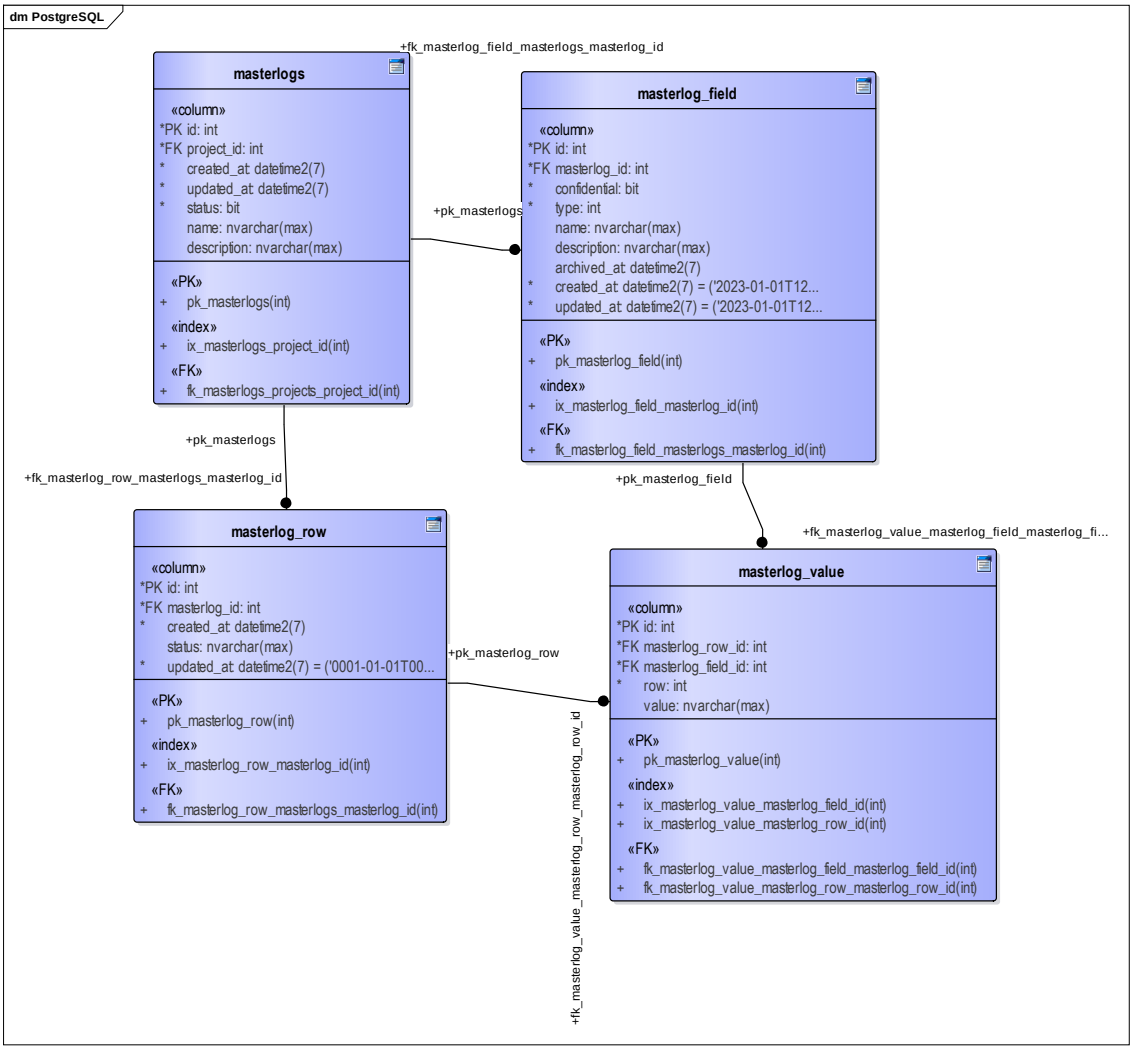
Joonis 17 Projekti olemi näide

Tabelis „Configurations“ on kõik konfiguratsioonid. Sellel on *foreign-key* ehk võõrvõti „ProjectId“, mis ühendab antud olemi kindla projektiga. Siin tekib üks-mitmele seos ehk ühel projektil võib olla mitu konfiguratsiooni. Igal konfiguratsioonil on määratud *field*'id ehk tulbad. Selleks kasutatakse eraldi tabelit „ConfigurationFields“, millel on võõrvõti „ConfigurationId“. Tabelit kasutatakse nime ja koodi hoidmisel. Lisaks sellele on konfiguratsioonil *item*'id ehk detailid, mis ühendavad tulba ja tulbas olevad andmed. Iga detail on lisatud tabelisse „ConfigurationItemValue“, millel on võõrvõti „ConfigurationItemId“. See olem hoiab väärtust ja rea numbrit. Igal „ConfigurationItem“-il on üks tulp ja list väärtustest (Joonis 18).

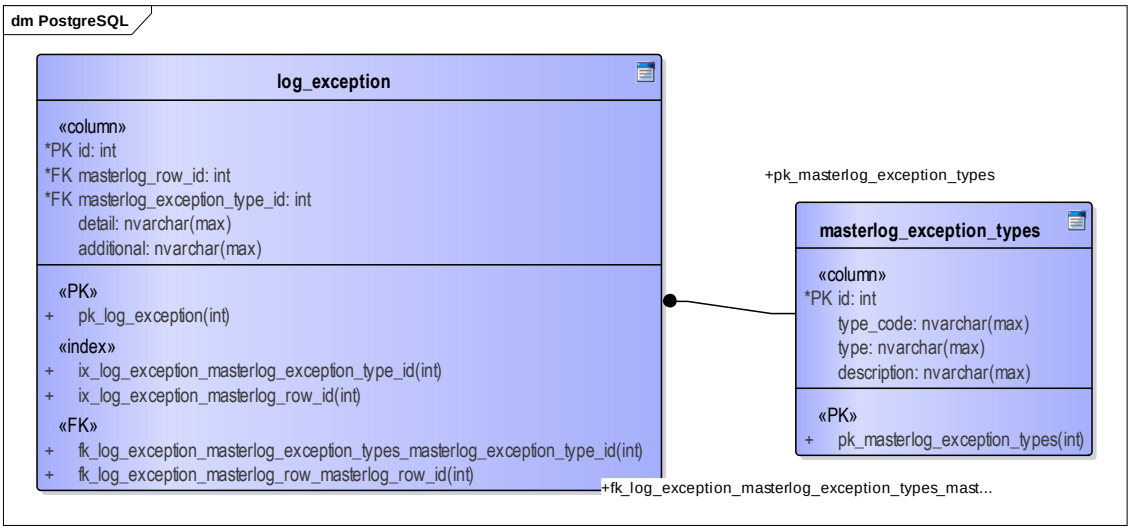


Joonis 18 Konfiguratsioonide olemi diagramm

Tabelis „Masterlogs“ on kõik põhilogide tüübid. Sellel on võõrvõti „ProjectId“, mis ühendab projekti ja põhilogi tüübi. Igal põhilogil on nimekiri tulpadest, mis asuvad tabelis „MasterlogFields“. Selles määratakse ära tulba nimi ja tulba konfidentsiaalsus. Tehtud töid peegeldab tabel „MasterlogRows“, mis on kui üks rida tabeltöötlusprogrammis. Kõik väärtused on eraldi tabelis nimega „MasterlogValues“, millel on kaks võõrvõtit, „MasterlogFieldId“ tulba ühendamiseks ja „MasterlogRowId“ rea ühendamiseks (Joonised 19 ja 20).



Joonis 19 Põhilogide olemi diagramm



Joonis 20 Põhilogide erandi ja tüübi olemi diagramm

3.10 Veebirakenduse disain

Kasutajaliidese disainimisel lähtuti admin-tüüpi paneeli mustritest (Joonis 21). Esiteks on vasakul põhinavigatsioonimenüü, mis haldab järgmisi navigatsioonivalikuid: avaleht, põhilogid, logid, konfiguratsioonid, liikmed, statistika. Menüü ülaosas on aktiivse projekti nimi ning selle kaudu saab navigeerida projekti tabelilehele, kus saab aktiivset projekti vahetada. Navigatsioonireal on täiendavad alalehed, mis täpsustavad navigatsioonimenüüst valitud lehte. Joonisel on valitud „Põhilogi“ leht. Navigatsiooni-reale tekivad põhilogile omased alalehed nagu „Põhilogide nimekiri“, „Põhilogide tüübid“ ja „Lisa uus“. Tabelisse kuvatakse kõik robotite poolt tehtud tööd.

Joonis 21 Admin-paneel tüüpi kasutajaliidese disaini prototüüp

4 Veebirakenduse arendus

Antud peatükkis võetakse ette teenuse- ja kliendipoolse rakenduse arendus. Selgitatakse arenduse lähenemist ning mis viisil eesmärkideni jõuti. Lisaks sellele tuuakse näiteid valminud kasutajaliidesest ja peatükki lõpus selgitatakse testimisest.

4.1 Teenusepoolne rakenduse arendus

Projekti raskustase on kõrge just robotite muutuva struktuuri mõttes. Rakendus ehk REST-API kirjutamisel kasutati C# programmeerimiskeelt ning ASP.NET Core Web API 7.0 raamistikku. Arendusel järgiti *domain-driver-design* põhimõtteid ja back-end jagati väiksemateks projektideks. Päringute haldamiseks kasutatakse kontrollereid, mis vastavalt parameetritele reageerib. Loogika eest vastutavad teenused ehk *service'd*, mis kutsutakse välja kontrollerist. Kõik kontrollerid pärivad baaskontrollerist, mille ülesanne on vajadusel kontrollida kasutaja autentimist. Andmebaasist andmete kätte saamiseks kasutatakse *DataContext* klassi ning repositooriume. Kõik klassid on kõrge struktuuri reeglitega ning selleks kasutatakse *interface*. Andmebaasist pärit ja lõppkasutajateni jõudvad muutujate nimed on turvalisuse ja lihtsuse eesmärgil erinevad, selleks kasutatakse *Data Transfer Object* (DTO). *Entity* ja *DTO* automaatseks kaardistamiseks kasutatakse teeki *AutoMapper*.

4.1.1 Arhitektuur

Tarkvaraarenduses viitab arhitektuur rakenduse või süsteemi üldisele disainile ja korraldusele. See on oluline osa arenduses, kuna see võib oluliselt mõjutada süsteemi hooldatavust, skaleeritavust ja üldist kvaliteeti. Hästi läbimõeldud arhitektuur võib hõlbustada uute funktsioonide lisamist, vigade parandamist ja jõudluse optimeerimist, samas kui halvasti kavandatud arhitektuur võib muuta need ülesanded palju keerulisemaks. Domeenipõhise arenduse lähenemisviis rõhutab tugevat eraldamist rakenduse eri osade vahel, kus iga projekt vastutab konkreetse funktsionaalse valdkonna eest. See muudab rakenduse hooldamise ja aja jooksul skaleeruvuse veelgi lihtsamaks.

API - alamprojekt on kogu teenuserakenduse tuum, mis sisaldab rakenduse käivituse seadistusi, konfiguratsioone ja kontrolleriklasse. Kontrolleriklassid vastutavad päringute vastuvõtmise ja töötlemise eest ning nad pakuvad kliendirakendustele liidestust, mille kaudu saab teenuseid kasutada. Lisaks sisaldab API projekt ka erinevaid laiendusklasse, mida saab kasutada mitmesuguste funktsionaalsuste lisamiseks.

Application - selles alamprojektis on suurem osa äriloogikast ja rakendusespetsiifilisest koodist. See sisaldab mitut alamosa, millest igaühel on oma vastutusalad:

- Authorization: sisaldab autentimise ja autoriseerimisega seotud klasse, nagu autentimisteenused, autoriseerimispoliitikad ja erinevad *middleware*[1] ehk vahevarad.
- DTO: sisaldab andmeedastusobjekte ehk DTO[2]-sid, mida kasutatakse suhtluseks rakenduse erinevate kihtide vahel. Objektide automaatseks kaardistamiseks on kasutatud teeki AutoMapper.
- Services: sisaldab teenuseklasse, mis rakendavad äriloogikat ja sooritavad toiminguid andmekihis.

Domain - see projekt sisaldab domeenimudelite klasse ja liideseid, mis määratlevad olemid ja nende seosed rakenduses. Olemiklassid, mis esindavad äriobjekte nagu näiteks projekt, konfiguratsioon, logid, aga ka repositooriumeid, mis määratlevad toimingud, mida saab nende olemitega teha. Toimingute all on mõeldud CRUD[3] meetodeid nagu lisamine, lugemine, värskendamine ja kustutamine.

Persistance - see projekt vastutab andmebaasiga suhtlemise ja rakenduse andmekihi haldamise eest. See sisaldab klasse, nagu DataContext, mis esindab andmebaasi konteksti ja haldab ühendust, samuti klasse andmebaasi algandmete sisestamiseks ja andmebaasi migratsiooni käivitamiseks.

[1] *Middleware* – Lisalüli, mis asub rakenduse osade vahel, näiteks päringu ja kontrolleri vahel [45]

[2] DTO – *Data Transfer Object*, andmeedastusobjekt [46]

[3] CRUD – *Create, Read, Update, Delete*

4.1.2 REST API

Rakenduse üks põhiosa on päringute loomine. Selles osas tuleb arvestada päringute keerukust, suurust ja andmete seoseid. Liiga keerulised ja mahukad päringud võivad rakenduse kasutamist aeglustada ning nõuda rohkem ressurssi. API päringute defineerimisel on aluseks võetud kasutajalood ja -juhud. Esmalt on loodud kontrollerid (Tabel 5), mis võimaldavad erinevate ressursside kättesaamist. Kontrollerite klassid on vastavalt konfigureeritud, määrates esmalt päringu aadressi, mis on tavaliselt struktuuriga "api/[versioon]/[kontrolleri-nimi]". Versioneerimist kasutatakse, et säilitada tagasiühilduvus ja hallata API muudatusi aja jooksul. See võimaldab arendajatel teha muudatusi ilma olemasolevaid API-le tuginevaid klientrakendusi rikkumata. Päringute loomisel on jälgitud ühtlast joont. Enne päringu koostamist määratakse autentimise nõue ja meetodi tüüp (Joonis 22), iga päring kontrollitakse vastavalt õigustele. Antud näites valideeritakse kasutaja ning kontrollitakse õigusi kasutaja ja projekti vahel. Kasutaja valideerimisest lähemalt alampeatükkis 4.1.5 lk 45.

```
    [Authorize]
    [HttpGet("{id}")]
    public ActionResult<ProjectResponse> GetById(int id)
    {
        var project = _projectService.GetProject(id);
        if(project == null)
            return NotFound(new { message = "Project not found"});

        if (!_projectService.IsAccountConnectedToProject(id, Account.Id))
            throw new ForbiddenException("The client does not have access rights to
the content");

        return _mapper.Map<ProjectResponse>(project);
    }
```

Joonis 22 Päringu näide

Tabel 5 Teenusrakenduse kontrollerid

Kontrolleri nimi	**Sisu**
AccountsController	Kasutajate registreerimine, autentimine, logimine
BaseController	Autenditud kasutaja tagastamine (JWT)
ConfigurationController	Konfiguratsioonide loomine, muutmine, tagastamine, uuendamine
LogsController	Logide sisestamine, tagastamine
ProjectsController	Projektide loomine, otsimine, liikmete uuendamine

Eraldi on välja toodud (Lisa 4) kõik võimalikud päringud, mis on sorteeritud kontrollerite järgi. Päringu parameetrid on esitatud loogeliste sulgude sees. Näiteks konfiguratsiooni päringu ressursis "/configurations/{id}" täpsustab "id" parameeter otsitava konfiguratsiooni identifikaatorit. Päringute tüübid vastavad CRUD reeglitele – Create (loomine), Read (lugemine), Update (uuendamine) ja Delete (kustutamine). Selleks kasutatakse vastavalt HTTP-meetodeid GET, POST, PUT ja DELETE.

4.1.3 Andmebaas

Andmete salvestamiseks ja käsitlemiseks kasutati PostgreSQL andmebaasi. Selleks, et andmebaasi kasutada, oli vaja rakendada teatud täiendavaid meetmeid. Andmebaasi initsialiseerimiseks kasutati Dockerit, mis võimaldab virtualiseerida andmebaasiserveri töökeskkonna.

Kasutades *Entity Framework'*i ja *DataContext* [1]klassi, oli andmebaasi konfigureerimine ja ülesseadmine lihtne. EF[2] võimaldab kirjutada päringuid andmebaasi kui objektorienteeritud andmekogumi kaudu, mida nimetatakse EF-mudeliteks. EF-mudelid kirjeldavad andmebaasi skeemi ja seoseid tabelite vahel ning võimaldavad kasutada

[1] DataContext klass võimaldab andmebaasiga suhelda kasutades LINQ ja SQL. [44]

[2] EF – *Entity Framework,* andmebaasi päringute lihtsustamiseks [47]

LINQ[1]-i päringuid nende andmete käsitlemiseks. Analüüsis väljatöötatud olemitest tehti esmalt klassid ning määrati nende vahelised seosed. Seejärel lisati kõik objektid *DataContext* klassi, mis andmebaasi genereerimisel koostab objektide ehk olemi-klasside põhjal tabelid (Joonis 23).

```
public DbSet<Account> Accounts { get; set; }
public DbSet<Project> Projects { get; set; }
public DbSet<Configuration> Configurations { get; set; }
public DbSet<Masterlog> Masterlogs { get; set; }
```

Joonis 23 DataContext klassis määratud tabelid

Lisaks sellele on kasutatud OnModelCreating meetodit, mida kasutatakse andmemudeli konfigureerimiseks enne selle loomist andmebaasis. See meetod annab kontrolli andmebaasi loomise ja seoste seadmise üle ning võimaldab konfigureerida mitmeid üksikasju, sealhulgas võtmeid, indekseid, suhteid ja piiranguid. Meetodis on seadistatud andmemudeli omadused erinevate olemi-klasside seoste, indeksite ja võtmete osas, sealhulgas kuidas EF Core peaks teatud olukordades käituma, näiteks kui seostatud kirje kustutatakse (Joonis 24).

```
modelBuilder.Entity<ConfigurationItem>()
            .HasMany(x => x.Values)
            .WithOne(x => x.ConfigurationItem)
            .OnDelete(DeleteBehavior.Cascade);
```

Joonis 24 ConfigurationItem objekti konfigureerimise näide. OnDelete määramine

4.1.4 Unit of Work

Andmebaasi toimingute koondamiseks ühte loogilise üksusesse kasutati Unit of Work mustrit [38]. See on realiseeritud abstrakse klassina, mida kasutatakse ühise andmebaasi toimingute kogumina, mis haldab andmebaasi toiminguid nagu salvestamine, kustutamine ja muutmine. Antud muster kogub toimingud ühte loogilisse üksusesse, mida saab seejärel käsitleda kui üht tervikut. Mitu erinevat andmebaasi päringut käsitletakse kui ühte transaktsiooni, mis tähendab, et kas kõik toimingud salvestatakse või ühtegi neist

[1] LINQ - Language Integrated Query [43]

ei salvestata. See aitab tagada andmebaasi terviklikkust ja vähendada vigade tekkimise tõenäosust.

Näites (Joonis 25) initsialiseeritakse projekti repositoorium muutujasse "repository". See muutuja võimaldab välja kutsuda eeldefineeritud meetodeid, mis on seotud Unit of Work'iga ja võimaldavad andmebaasiga suhelda ning projekti andmeid käsitleda. Näiteks uue projekti lisamiseks kasutatakse "repository" muutujaga seotud meetodeid, nagu "Add", et sisestada uued andmed andmebaasi "Project" tabelisse või olemasoleva projekti pärimiseks meetodit „Get“ (Joonis 26).

```
var repository = UnitOfWork.Repository<Project>();
```

Joonis 25 Unit Of Worki kasutamise näide

```
public T Get(Expression<Func<T, bool>> expression)
{
    return _dbSet.FirstOrDefault(expression);
}
```

Joonis 26 Get meetodi koodi näide

4.1.5 Autentimine ja turvalisus

Turvalisus on igasuguse tarkvaraarenduse keskne teema, kuna turvarikked võivad põhjustada tõsiseid tagajärgi nii ettevõttele kui ka kasutajatele. Kui antud rakendus on mõeldud ainult ettevõtte sisevõrgus kasutamiseks, siis ei tähenda see, et turvalisuse aspektidele tähelepanu pööramist saaks eirata. Antud peatükis keskendutakse turvalisusele ning tutvustatakse, kuidas tagatakse kasutajate autentimine ja autoriseerimine ning millised meetodid ja tööriistad aitavad vältida turvaintsidente. Autentimise all on mõeldud kasutaja tuvastamist ning seda, kas antud kasutaja tegevused on õigustega lubatud.

Projektis kasutatakse turvalisuse eesmärkide saavutamiseks JWT ehk *Json Web Tokenit*, mis on turvaline viis andmete edastamiseks serveri ja kliendi vahel [39]. See koosneb kolmest osast: pealkirjast, andmetest ja allkirjast. Kuna JWT-de allkirjad on krüpteeritud, on andmete terviklus tagatud. Arenduse käigus määrati JWT aegumise ajaks üks päev, et kiirendada arenduse ja testimise protsessi. Produktsioonis on kehtestatud 30-minutiline JWT-i aegumisaeg, et tagada andmete turvalisus ja vältida volitamata juurdepääsu.

Eraldi on loodud klass nimega „AuthorizeAttribute“, mis võimaldab kontrollerites täpsustada meetodite ligipääsetavust, lisades meetodi kohale atribuudi „[Authorize]”. Lisaks sellele saab määrata, millistele kasutaja rollidel on antud funktsioonile ligipääs, täpsustades vastavad rollid atribuudi parameetrites - näiteks „[Authorize(Role.Admin, Role.User)]”. Kui kasutajal puuduvad vastavad õigused, tagastatakse veateade HTTP 401 Unauthorized. Esimeses skoobis rollidele tähelepanu ei pööratud. Projekti spetsiifilised rollid ja õigused on loodud eraldi funktsionaalsusena.

4.2 Kliendipoolse rakenduse arendus

Kliendipoolne rakendus on arendatud kasutades React raamistikku koos TypeScriptiga. Parima tulemuse saavutamiseks kasutati taaskasutatavaid komponente, et vähendada vigu ja kuvada neid tõhusalt vastavalt oleku muudatustele. Siinkohal annab TypeScripti kasutamine erinevaid eeliseid nagu näiteks erinevate komponentide vahel ei teki olekutüübi vigu ning see on abiks keerukamate projektide arendamisel. TypeScript kasutab kõiki JavaScripti funktsioone, kuid lisab neile tüübi süsteemi, mille abil saab määrata muutujate, funktsioonide ja objektide tüübid.

Rakenduse loomiseks kasutati NPM[1], mille abil kogu projekt loodi. Lisaks sellele haldab antud paketihaldussüsteem erinevaid teeke ja seda kasutatakse projekti serveri jooksutamiseks. See pakub üle ühe miljoni avaliku teegi, mis sisaldab erinevaid tööriistu ja teisi arendusvahendeid.

4.2.1 Material UI

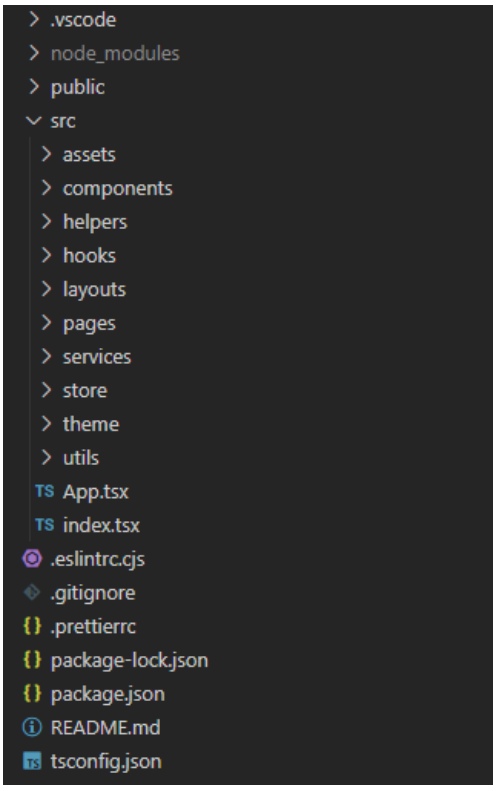
Material UI on populaarne avatud lähtekoodiga Reacti komponentide teek, mis on loodud Google poolt [40]. See sisaldab laiaulatuslikku eelehitatud komponentide kollektsiooni ja erinevaid stiilivõimalusi, mis võimaldab arendajatel kiiresti ja lihtsalt luua kaasaegse välimusega rakendusi.

[1] Node Package Manager - Javascript’i teekide halaja https://nodejs.dev/en/learn/an-introduction-to-the-npm-package-manager/

Material UI põhineb Google'i Material Designil, mis on kasutajaliidese disaini juhend ning rõhutab selget, puhast ja intuitiivset kasutajakogemust. Selle põhieesmärk on luua uus visuaalne keel, mis ühendab disaini põhimõtted tehnilise ja teadusliku uuendusega.

4.2.2 Struktuur

Reacti kasutades on üldiselt hea tava jagada oma kood mitmeks osaks, kasutades selleks kaustade struktuuri. Selleks kasutatakse sageli erinevaid kaustu nagu *components*, *hooks*, *pages*, *services*, *store* ja *utils*. Järgnevalt selgitatakse projekti tähtsamaid osasid lähemalt. *Components* kaust hoiab endas React komponente, mis on väiksed taaskasutatavad osad. *Hooks* kaustas hoitakse erinevaid JavaScript *hook*[1]'e nagu *useEffect*. *Pages* kaust sisaldab kõiki projekti lehti, mis on veebilehe URL-i järgi ligipääsetavad. Näiteks konfiguratsioonide, põhilogide, projektide lehed. *Store* kaust sisaldab rakenduse oleku haldamiseks *Redux*'i, mis on mõeldud suuremahulise rakenduse oleku haldamiseks (Vt joonis 27).



Joonis 27 Kliendirakenduse kaustade struktuur

[1] *Hooks* - https://legacy.reactjs.org/docs/hooks-overview.html

4.2.3 Päringud

Teenusepoolse rakenduse suhtluse saavutamiseks kasutati Fetch API'd[1], mis võimaldab teha HTTP[2] päringuid. Peale päringu tegemist saab saadud andmeid kasutada komponendis kuvamiseks või vea korral kuvada veateate (Joonis 29). Teenusepoolsel rakendusel on palju erinevaid päringuvõimalusi ning selleks loodi eraldi teenuste kiht. See kasutab fetchWrapper'it, mis on HTTP[1] funktsioonide kogum , mis kasutab Fetch API'd. Iga teenuse jaoks loodi eraldi fail nimega „teenuseNimi.service.ts". Näiteks projekti teenuse faili nimeks sai „project.service.ts". Joonis 28 kajastab projekti lisamiseks mõeldud päringut.

```
function addProject(request: AddProjectType) {
    return fetchWrapper.post(`${baseUrl}`, request)
}
```

Joonis 28 Projekti lisamise päringu näide

```
function post(url: string, body: any): Promise<any> {
  const requestOptions: RequestInit = {
    method: 'POST',
    headers: { 'Content-Type': 'application/json',
...authHeader(url) },
    credentials: 'include',
    body: JSON.stringify(body)
  };
  return fetch(url, requestOptions).then(handleResponse);
}
```

Joonis 29 HTTP POST meetodi näide kasutades Fetch API'd

4.2.4 Turvalisus

Maksimaalse turvalisuse saavutamiseks on mõlemad rakendused arendatud arvestades maksimaalse turvalisuse põhimõtteid. See tähendab, et mõlemal rakendusel on samad kontrollid veendumaks, et kasutaja ei kuritarvitaks või aeglustaks veebirakendust. Esmalt on igalt kasutajalt nõutud autentimine, et vältida ligipääsu andmetele, millele õigused puuduvad. Kasutaja tuvastamiseks kasutatakse JWT (JSON Web Token'it), mis on salvestatud veebilehel liikumisel brauseri mälusse ja antakse igal päringul kaasa. Päringu saates kontrollitakse autendi nõuet ja vastavat ligipääsu nõutud andmetele. Andmetele,

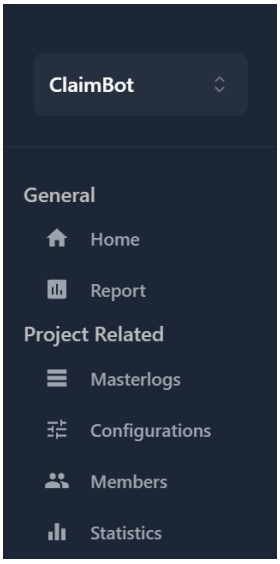
[1] Fetch API - JavaScripti API, mis võimaldab teha võrgupäringuid ja saada vastuseid serverilt.

[2] HTTP - Hypertext Transfer Protocol, hüperteksti edastusprotokoll

millele kasutajal ligipääs puudub kuvatakse HTTP 403 Forbidden veateade. Kui *token* ei kehti või on aegunud, kuvatakse kasutajale HTTP 401 Unauthorized veateade ja suunatakse sisse logimise lehele. Igal sisselogimisel läheb käima taimer, mis enne *token*'i aegumist selle ära uuendab. See väldib pidevat sisse logimist ja hoiab kasutaja rakenduses sees.

4.3 Kliendirakendus

Kliendirakenduse arendusel järgiti üldiseid kasutajakogemuse ja disaini mustreid [41]. Kliendirakendus põhineb *admin*-tüüpi disainil (Lisa 3). Vasakul pool lehte on põhinavigatsiooni osa, mis vastutab erinevate lehtede navigatsiooni eest (järgnevalt navigatsiooni menüü) (Joonis 30). Samuti on lisatud aktiivse projekti kuvamine ja selle vahetamine. Projekti osa all on erinevad navigatsioonielemendid, mis on jaotatud kaheks. Esmalt on välja toodud üldised navigatsioonielemendid – esileht ja raportid. Esileht avaneb peale sisselogimist ning vastutab esmase info kuvamise eest nagu näiteks üldine statistika ja viimased logid. Sama kehtib ka admin-tüüpi paneeli kohta. Täpsema statistika ja andmete kuvamiseks on raportite leht, kus saab teha juba detailsemaid analüüse. Raportite leht on navigatsiooni menüüs välja toodud, aga jäetud esimesest skoobist välja. Need lehed ei sõltu aktiivsest projektist ning kuvavad kõikide projektide koondandmeid, kuhu autenditud kasutajal õigused on. Teine grupp on just aktiivse projektiga seotud navigeerimised. Esmalt on põhilogid, seejärel konfiguratsioonid, projekti liikmed ja statistika leht.



Joonis 30 Vasaku paneeli navigeerimise näidis

Teine navigatsioon on lisatud lehe päisesse ning on spetsiifiline iga lehe jaoks (järgnevalt navigatsiooni rida) (Joonis 31). Navigatsiooni real, vasakul on kuvatud erinevad alamnavigatsioonid ja paremal pool kasutajaspetsiifilised, mis on igal lehel ühised. Põhilogide lehe näitel kuvatakse esmalt logid ning navigatsiooni realt on võimalik liikuda veel logide spetsiifilistele lehtedele nagu statistika ja põhilogide tüübid. Navigatsiooni rea all on välja toodud filtrite rida, mis on samuti lisatav ja konfigureeritav vastavalt lehe vajadustele.

Joonis 31 Ülemise navigatsioonirea näidis

Enamik lehtedest kasutab samu stiililahendusi. Juhul, kui tegu on nimekirjaga näiteks põhilogid, siis kuvatakse kasutajale tabel koos esmase info ja erinevate funktsionaalsete valikutega. Näiteks antud kirje muutmine, arhiveerimine, kustutamine ja aktiivseks määramine. See sõltub täpsemalt juba kirjest endast.

4.3.1 Autentimise leht

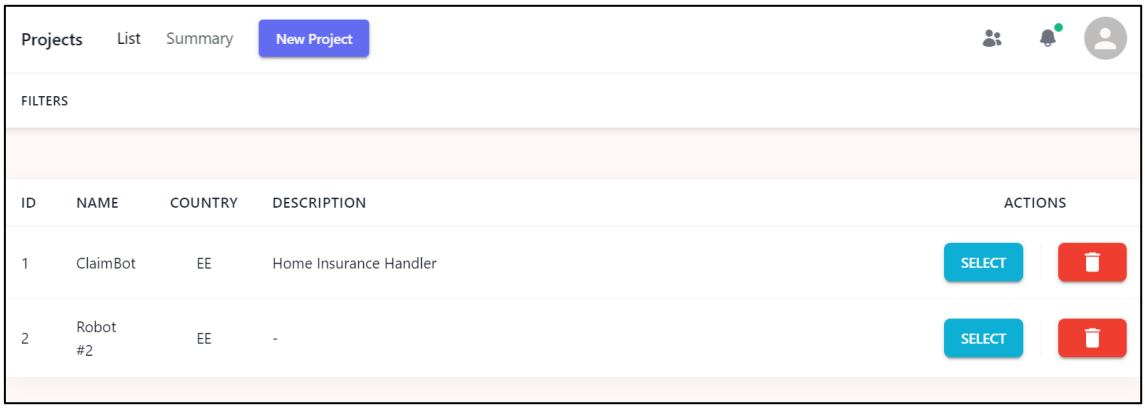
Autentimise leht on hetkel loodud kasutajanime ja parooli baasil. Eraldi registreerimisleht puudub, see oleneb juba täpsemalt ettevõtte vajadustest ja nõuetest – kas ja kuidas kasutajaid juurde lisatakse. Võimalikud variandid on järgnevad: registreerimine ettevõtte sisevõrgus, läbi admin paneeli uute kasutajate lisamine või ettevõttespetsiifilised autentimise võimalused, mida käsitletakse järgmises skoobis.

4.3.2 Esileht

Antud leht on esimene rakenduse poolt kuvatav leht, mis *admin*-tüüpi paneelis üldiselt kuvab esmast statistikat või viimaseid juhtunud sündmuseid. Esimest lähenemist on kasutatud ka antud veebilehe disainimisel. Tehtud tööde osas on välja toodud viimase nädala numbrilised väärtused: tehtud tööde arv, õnnestunud tööde arv, tehniliste vigade arv, äriliste vigade arv ning protsentuaalsed väärtused graafikuna. Iga väärtuse juurde on välja toodud protsentuaalne muutus võrreldes eelmise nädalaga. Lisaks tehtud tööde statistikale kuvatakse olenevalt projektist kas kõige uuemaid põhi- või üldlogisid. Üldlogidest on välja toodud just kriitilisemad nagu protsessi katkestamised või muud logid, mis vajavad kohest sekkumist vastava osakonna vastutavade poolt.

4.3.3 Projektide leht

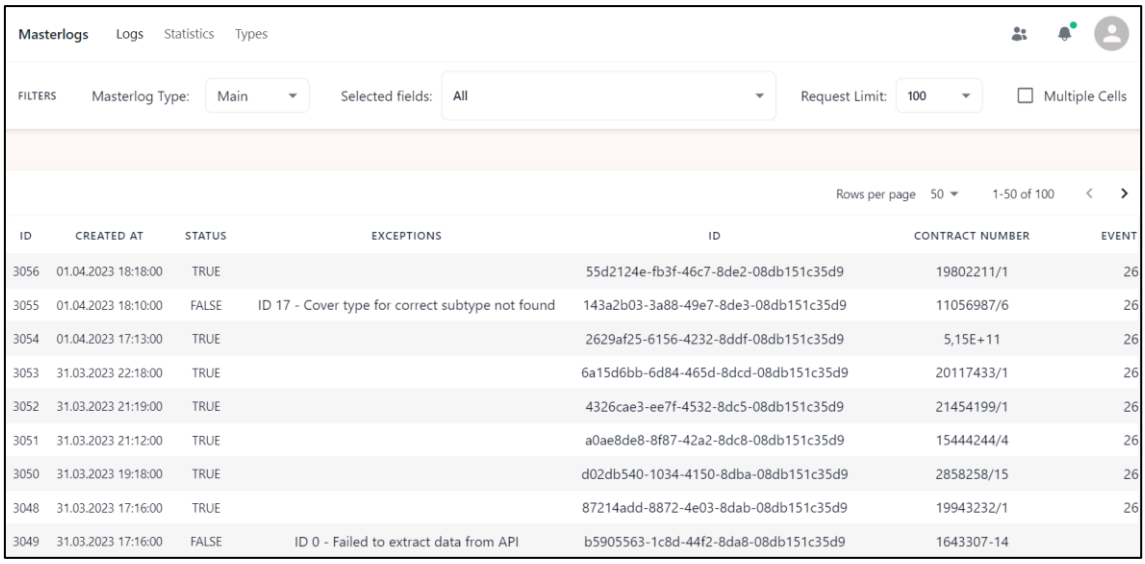
Projektide lehel on esmalt kuvatud tabel kõikidest projektidest, kuhu autenditud kasutaja kuulub. Lisaks saab antud lehel lisada uue projekti, vajutades ülemise navigatsiooni realt „New Project“. Projekti aktiveerimiseks tuleb tabelis leida vastav projekt ja seejärel vajutada nuppu „SELECT“. Projekti kustutamiseks tuleb vajutada punast prügikasti ikooniga nuppu. Kui kasutajal aktiivne projekt puudub, teeb kliendirakendus selle otsuse esmalt kasutaja eest, valides esimese olemasoleva projekti. Samuti salvestatakse otsus brauseri mälusse, et järgmistel sisse logimistel kiirendada protsessi.



Joonis 32 Projekti lehe näidis

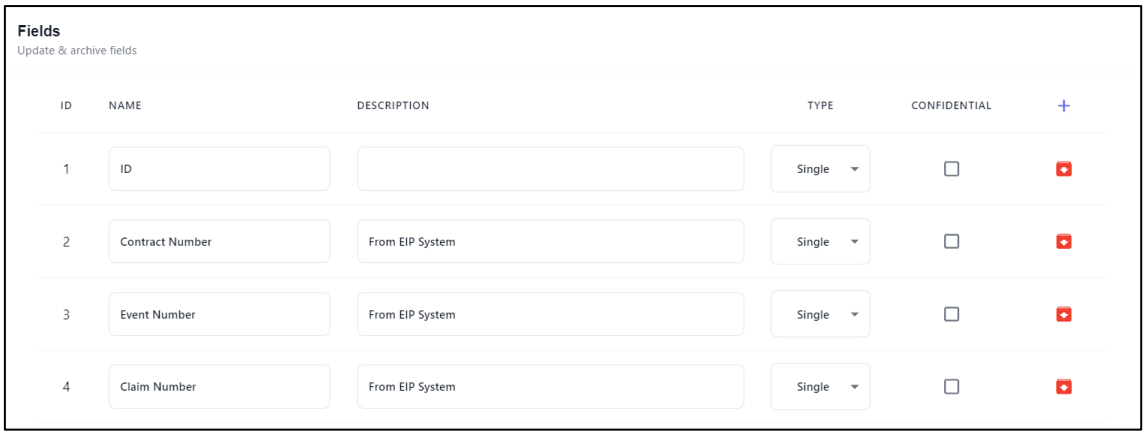
4.3.4 Põhilogide leht

See leht on kliendirakenduse üks tähtsamaid osasid. Siin kuvatakse kõige uuem info viimastest tarkvararoboti tehtud töödest. Siin kohal arvestatakse juba aktiivset projekti. Navigatsioonirea alla on lisatud filtreerimiseks rida, mis võimaldab antud tabeliga seotud täpsustusi teha. Näiteks on võimalik valida aktiivne põhilogide tüüp, tabelist eemaldada dünaamilised väljad, mida kuvada ei soovita, maksimaalsete ridade arvu ühes päringus ja nimekirja kuvamist ühes lahtris. Viimast juhul, kui lahter seda lubab. See tähendab, et ühte tabeli lahtrisse tekib nimekiri alamlahtritest, mida tabeltöötlusprogramm teha ei võimaldanud. Tabelisse kuvatakse tehtud tööd uudsuse järjekorras ning tabel on vertikaalselt keritav, see sõltub veergude ehk dünaamiliste väljade arvust. Väljade konfigureerimine toimub põhilogi tüübi lehel (Joonis 33).



Joonis 33 Põhilogide lehe näidis koos filtreerimise ja tabeliga

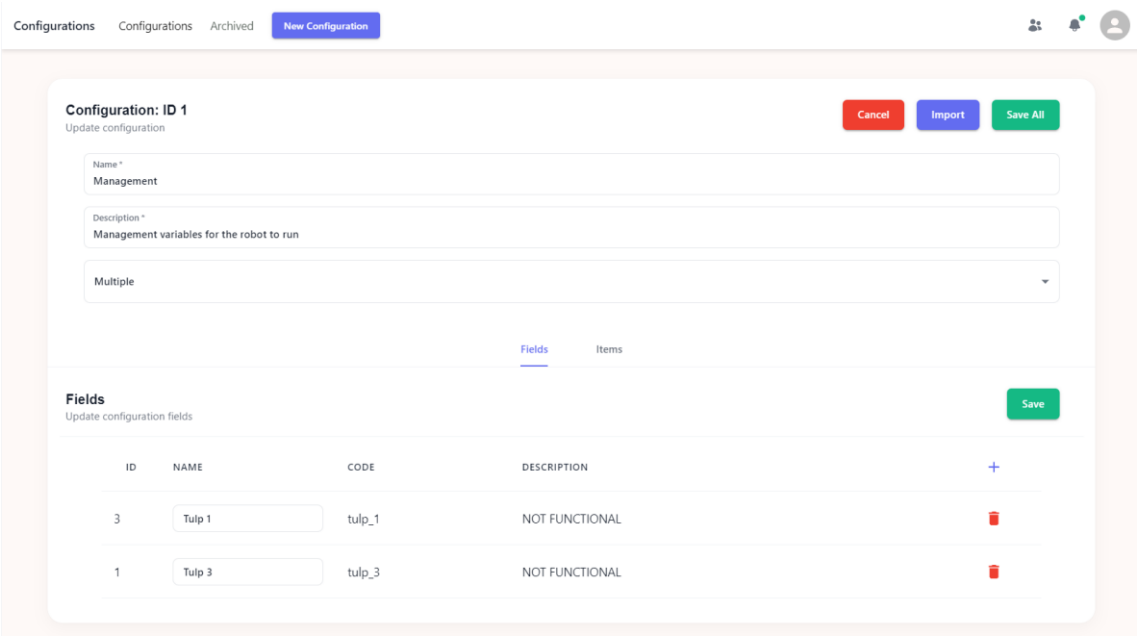
Põhilogi tüübi lisamisel tuleb esmalt määrata selle nimi ja lühikirjeldus. Lisaks sellele tuleb määrata kõik veerud, mis andmeid tabelisse koguma hakatakse. Igal veerul on nimi ja lühikirjeldus, mis täpsustab veeru sisu. Veeru tüüpe on kaks – „Single“ ja „List“. „Single“ tüüpi veergudel on ainult üks väärtus, „List“ tüüpi veerul võib olla mitu väärtust, mis teeb ühest lahtrist mitme alamväärtusega nimekirja. Viimasena on võimalik määrata veeru konfidentsiaalsus – kas antud veergu salvestatakse konfidentsiaalsed andmeid või kogutakse neid vaid kontrollimise ja statistika eesmärgil. Konfidentsiaalsete andmetega on võimalik piirata ligipääsu ja vajadusel teatud ajavahemiku möödudes andmed kustutada. Viimases lahtris on välja toodud veeru arhiveerimise võimalus, mis tähendab, et põhilogide lehel arhiveeritud veergu ei kuvata. Kõik muudatused, sealhulgas veergude muudatused ja arhiveerimised jõustuvad peale salvestamist (Joonis 34).



Joonis 34 Põhilogi tüübi lisamise / muutmise lehe näidis

4.3.5 Konfiguratsioonide leht

Sarnaselt teistele lehtedele, kuvatakse ka konfiguratsioonide lehel esmalt tabel. Kasutades ülemist navigatsioonirida on võimalik vaadata aktiivseid ja mitteaktiivseid ehk arhiveeritud konfiguratsioone. Konfiguratsioone on võimalik muuta, lisada ja arhiveerida. Samuti on võimalik arhiveeritud konfiguratsioone tagasi aktiivseks muuta. Konfiguratsiooni muutmise ja lisamise leht kasutab samu komponente. Lisamise või muutmise leht on jagatud kolmeks, esimeses osas on üldine info – nimi, lühikirjeldus ja tüüp. Teises osas saab vastavalt lisada või muuta välju ehk tulpasid (Joonis 35). Tabelis on välja toodud kõik veerud, mida saab vastavalt muuta või kustutada. Tulba kustutamisel kustuvad ka kõik antud tulba väärtused. Kõik muudatused ja toimingud jõustuvad alles salvestusel.

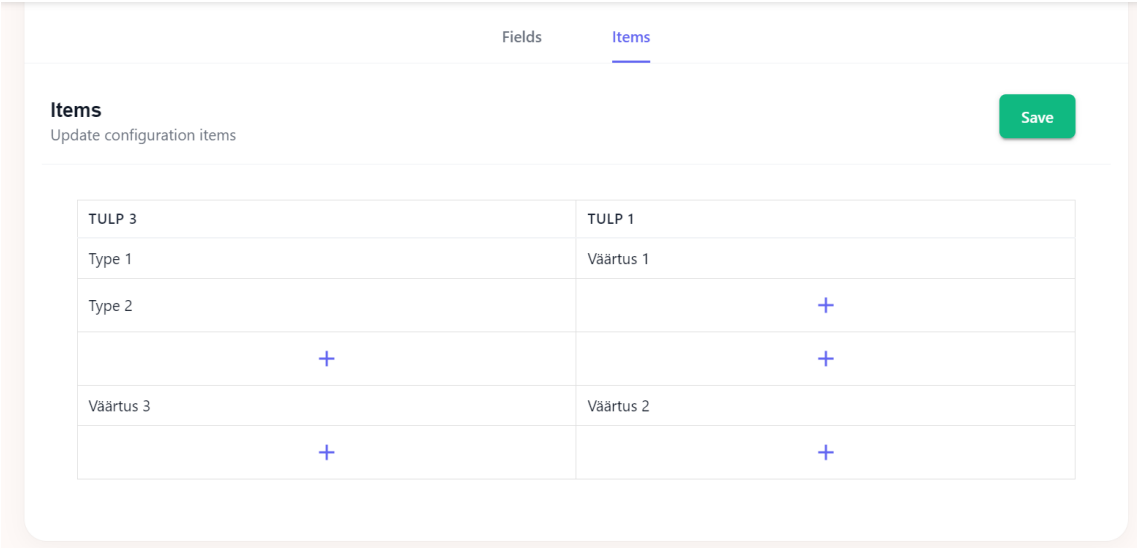


Joonis 35 Konfiguratsiooni ja tulpade lisamise lehe näidis

Kolmas osa keskendub tabeli väärtustele (Joonis 36). Eelmises osas määratud veergudele on võimalik sisestada väärtused. Tabeli suurus oleneb veergude ja konfiguratsiooni tüübist – üherealised või mitmerealised konfiguratsioonid. Üherealised on mõeldud just *key-value*[1] tüüpi seadistustele ning on nõutud robotite arenduse tarkvaras. Mitmerealised konfiguratsioonid on mõeldud just suurtemate andmete kaardistamisel või seadistamisel.

[1] *key-value* - programmeerimistermin, mis viitab andmete struktuurile, mis koosneb kahest osast: võtmest (*key*) ja väärtusest (*value*).

Tabelis kuvatakse viimasel real tühi rida, mis võimaldab mugavalt uusi ridu lisada. Igat lahtrit on võimalik lisada, muuta või kustutada.



Joonis 36 Konfiguratsiooni leht. Tabeli väärtuste muutmise / lisamise lehe näidis

4.4 Testimine

Testimine on üks oluline osa tarkvaraarenduse protsessis. Selle eesmärk on tagada rakenduse töökindlus ning vastavus ettemääratud nõuetele. Korralikult testitud tarkvara aitab vähendada vigu ja probleeme, mis tagab parema kvaliteedi kasutamisel.

Teenusrakenduse testimiseks kasutati NUnit raamistikku. Lisaks sellele võeti testide loomiseks kasutusele FakeItEasy ja AutoFixture paketid. NUnit on C#-i üks populaarsemaid testiraamistikke, mis võimaldab luua ja käivitada automaatseid teste. NUnit pakub ka võimalusi testandmete genereerimiseks, testide järjestamiseks, ajastamiseks ja katvuse mõõtmiseks. FakeItEasy on avatud lähtekoodiga raamistik .NET keele jaoks. See võimaldab lihtsalt ja loogiliselt testida oma rakenduse komponente, kasutades võltsobjekte, mis teeb testimisprotsessi tunduvalt kiiremaks. AutoFixture on .NET platvormile loodud teek, mis aitab lihtsustada testimist, genereerides automaatselt juhuslikke andmeid objektide ja andmetüüpide jaoks.

Esmalt loodi eraldi projekt nimega „Tests“, kuhu installeeriti eelmainitud paketid. Testitavate klasside sisu oli jagatud vastavalt kaustadesse – kontrollerite testid kaustas „Controllers“ ning faili nimi näiteks „ProjectsControllerTests.cs“, teenuste testid kaustas

„Services“, olemite testid kaustas „Entities“, andmeedastusobjektide testid kaustas „DTO“. Samuti loodi päritavad klassid, mis sisaldavad klasside ühisosa, näiteks kontrollerite testimiseks loodi päritav klass, kuhu sisestati kõikide kontrollerite ühine koodi osa (Joonis 37).

```
[TestFixture]
public class CommonControllerTests
{
    protected IProjectService _projectService;
    protected IConfigurationService _configurationService;
    protected ILogService _logService;
    protected IMapper _mapper;
    protected IFixture _fixture;

    [SetUp]
    public void SetUp()
    {
        _fixture = new Fixture();
        _projectService = A.Fake<IProjectService>();
        _configurationService = A.Fake<IConfigurationService>();
        _logService = A.Fake<ILogService>();
        _mapper = A.Fake<IMapper>();

        _fixture.Behaviors.OfType<ThrowingRecursionBehavior>().ToList()
            .ForEach(b => _fixture.Behaviors.Remove(b));
        _fixture.Behaviors.Add(new OmitOnRecursionBehavior());
    }
}
```

Joonis 37 Koodinäide päritavast testi kontrolleriklassist

Testikontrolleri sees määrati sellele vastav kontrolleriklass ning päriti „CommonControllerTests“ klass. Konstruktoris initsialiseeriti kontroller ning testimiseks määrati ka autenditud kasutaja ID (Joonis 38).

```
    private ProjectsController _controller;

    [SetUp]
    public void SetUp()
    {
         _controller = new ProjectsController(_projectService,
              _configurationService, _logService, _mapper);
        _controller.ControllerContext.HttpContext = new
               DefaultHttpContext();
        _controller.ControllerContext.HttpContext.Items["Account"] = new
               Account { Id = 1 };
    }
```

Joonis 38 Projekti kontrolleri testiklassi näide

5 Hinnang loodud veebirakendusele

Loodud veebirakenduse esimene skoop lahendas püstitatud probleemi. See tähendab, et valminud projektiga on võimalik asendada kõik tabeltöötlusfailid ning integreerida kogu tarkvararobotite suhtlus. Lisaks sellele koostab valminud tarkvara juba üldist statistikat automaatselt, mis eelnevalt oli tülikas ja aeganõudev töö. Kui esimene skoop lahendab üldiselt ainult probleemi, siis järgnevad edasiarendused efektiivistavad ja kiirendavad protsessi. Valminud tarkvara on veel arendusjärgus ja sellepärast ei ole seni ettevõttes kasutusele võetud. Veebirakenduse esmast skoopi on osakondade töötajatele esitletud ning saanud positiivset tagasisidet valideerides magistritöö eesmärki.

5.1 Majanduslik pool

Magistritöö teema valikul oli oluline roll RPA juurutamisel ja kasvamisel. Aastate jooksul loodud tarkvararobotid ja RPA lähenemine on oma kasumlikust igati tõestanud ja õigustanud. Sellest tulenevalt kasvas välja protsessis puudulik külg, mida antud uurimuse käigus lahendatakse, et ettevõttele rohkem väärtust luua ja kogu protsessi haldus efektiivsemaks muuta. Eelduste kohaselt hoiab lahenduse kasutusele võtmine kokku orienteeruvalt 1-2 tundi nädalas iga projekti manageerimise ja monitoorimise juures sõltuvalt roboti töökoormusest ja protsessi ärikriitilisusest. Magistritöö majanduslik väärtus kasvab eksponentsiaalselt tarkvararobotite arvu suurenemisega.

5.2 Edasiarendused

Kasutaja õigused ja rollid – Projektispetsiifiliselt oleks võimalik määrata erinevatele andmetele ligipääse nii lugemise kui muutmise vaatepildist. See tähendab, et kasutaja õigust saab määrata vastavalt konfiguratsiooni või põhilogile ning täpsustada mis funktsionaalsust kasutada saab. Lisaks projektispetsiifilistele rollidele on tarkvaralised rollid ehk administraatori paneeli võimalused, võimalus muuta ja seadistada veebirakendust.

Administraatori paneel – Tagab terve veebirakenduse kontrolli, võimalik näha kõikide andmete statistikat, teha kasutajatega seonduvaid muudatusi, sealhulgas kasutajate eemaldamine ja/või lisamine.

Statistika – Detailsema statistika tagamine erinevate graafikute ja diagrammide abil. Iga projekti spetsiifiliselt võimalik konfigureerida vastavalt muutujale graafikud ja need salvestada järgmisteks kasutusteks.

Andmete import ja eksport – Andmete importimise ja eksportimise võimalused. Näiteks põhilogide ehk tehtud tööde tabeli eksportimine tabeltöötlusprogrammi arhiveerimiseks. Samuti konfiguratsiooni importimine kasutades tabeltöötlusprogrammis eelnevalt täidetud andmeid.

6 Kokkuvõte

Antud töö eesmärk oli leida lahendus tarkvararobotite manageerimiseks ja andmete haldamiseks. Lahenduse esimene skoop hõlmas endas prototüüpi, mis asendab kõik robotitega seonduvad tabeltöötlusprogrammid ja on infovahetuse platvormiks erinevate osapoolte vahel. Töö analüüsis toodi välja erinevad arenduse võimalused ning loodi ülevaade potentsiaalsetest tehnoloogiatest, sealhulgas viimaste aastate jooksul ilmunud uudsetest lähenemistest. Lõputöö arenduskäigu peatükk annab ülevaate teenus- ja kliendirakenduse arhitektuurist, turvalisusest ja disainist ning loodud prototüübist.

Töö käigus valmis veebirakendus koos funktsionaalsusega lahendades püstitatud probleemi, milleks oli tabeltöötlusprogrammide kasutamine. Lisaks sellele võimaldab loodud rakendus tarkvararoboteid käivitada samaaegselt mitmes virtuaalmasinas, mis varasemalt oli tabeltöötlusprogrammi piirangute tõttu keeruline. Kuigi kõik analüüsitud nõuded ei valminud magistritöö raames, on rakendusel suur potentsiaal ning uute funktsionaalsuste lisamine toob tulevikus ettevõttele rohkem väärtust. Lisaks analüüsitud nõuetele on rakendusse planeeritud edasiarendusi, mis annab märku sellest, et autor on pühendunud rakenduse täiustamisele ja parandamisele. Projekti saab lugeda õnnestunuks, sest tänu sellele vähendab ettevõte püsikulusid litsentside arvelt ja hoiab kokku märkimisväärselt töötajate aega tagades seeläbi robootika tiimi jätkusuutlikkuse. Mugavalt kättesaadavaks muutunud andmete süvaanalüüs võimaldab luua olulist ärilist väärtust tarkvararobotiga seotud protsessi kontekstis.

Kasutatud kirjandus

[1] S. Madakam, „The Future Digital Work Force: Robotic Process Automation (RPA),“ *Journal of Information Systems and Technology Management,* 2019.

[2] T. Taulli, The Robotic Process Automation Handbook, Apress; 1st ed. edition (February 29, 2020), 2020.

[3] Robotic Process Automation: Management, Technology, Applications, De Gruyter (May 10, 2021), 2021.

[4] M. C. Lacity ja L. Willcocks, Robotic Process and Cognititve Automation: The Next Phase, SB Publishing, 2018.

[5] T. Chakraborti, From Robotic Process Automation to Intelligent Process Automation, 2020.

[6] Capco Institute, „Avoiding pitfalls and unlocking real business value with RPA,“ *JOURNAL 46: AUTOMATION,* p. 19, 2017.

[7] A. Aavik, „11 levinumat RPA rakendust aastaks 2021,“ 04 12 2020. [Võrgumaterjal]. Available: https://flowit.ee/11-levinumat-rpa-rakendust/.

[8] J. Ahvenainen, „Why do RPA projects fail? Automation is what we make of it,“ Robocorp, 19 4 2021. [Võrgumaterjal]. Available: https://robocorp.com/blog/why-do-rpa-projects-fail-automation-is-what-we-make-of-it.

[9] Xenith, „Top 4 Reasons For The Failure Of Robotic Process Automation (RPA),“ 29 06 2020. [Võrgumaterjal]. Available: https://www.xenith.co.uk/blog/top-4-reasons-for-the-failure-of-robotic-process-automation.

[10] D. Norman, The Design of Everyday Things, 1988.

[11] „IBM,“ [Võrgumaterjal]. Available: https://www.ibm.com/topics/rest-apis.

[12] „Cloudflare,“ [Võrgumaterjal]. Available: https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/.

[13] „JSON,“ [Võrgumaterjal]. Available: https://www.json.org/json-en.html.

[14] [Võrgumaterjal]. Available: https://powerplatform.microsoft.com/en-us/power-bi/.

[15] „Micosoft,“ [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-7.0&tabs=visual-studio.

[16] H. K. Dhalla, „A Performance Analysis of Native JSON Parsers in Java, Python, MS.NET Core, JavaScript, and PHP,“ %1 *International Conference on Network and Service Management,* 2020.

[17] „Node.js,“ [Võrgumaterjal]. Available: https://nodejs.org/en/about.

[18] „Java,“ [Võrgumaterjal]. Available: https://docs.oracle.com/javase/tutorial/.

[19] A. Insignares, „React Pros and Cons: What are the Advantages and Disadvantages of ReactJS?,“ [Võrgumaterjal]. Available:

https://www.koombea.com/blog/react-pros-and-cons-what-are-the-advantages-and-disadvantages-of-reactjs/.

[20] Vue, „Vue.js Introduction,“ [Võrgumaterjal]. Available: https://vuejs.org/guide/introduction.html.

[21] „Stack Overflow Survery,“ [Võrgumaterjal]. Available: https://survey.stackoverflow.co/2022/.

[22] L. Harkushk, D. Omelchenko ja Y. Panasenko, „Vital things to consider when choosind a database yor your app,“ [Võrgumaterjal]. Available: https://yalantis.com/blog/how-to-choose-a-database/.

[23] DB-Engines, „DB-Engines Ranking,“ [Võrgumaterjal]. Available: https://db-engines.com/en/ranking.

[24] Simplilearn, „What is a Relational Database Management System?,“ 12 12 2022. [Võrgumaterjal]. Available: https://www.simplilearn.com/relational-database-management-system-article.

[25] E. F. Codd, „A Relational Model of Data for Large Shared Data Banks,“ *Communications of the ACM,* kd. 13, nr 6, 1070.

[26] C. S. Mullins, „NoSQL (Not Only SQL database),“ [Võrgumaterjal]. Available: https://www.techtarget.com/searchdatamanagement/definition/NoSQL-Not-Only-SQL.

[27] N. Leavitt, „Will NoSQL Databases Live Up to Their Promise?,“ 2010.

[28] „MongoDB Docs - Query Documents,“ [Võrgumaterjal]. Available: https://www.mongodb.com/docs/manual/tutorial/query-documents/.

[29] S. Agrawal, „Adoption of database technology: A comparative study,“ *Control Theory and Informatics,* kd. 3, nr 5, 2013.

[30] M. Drake, „A Comparison Of Relational Database Management Systems,“ 14 2 2014. [Võrgumaterjal]. Available: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresql-a-comparison-of-relational-database-management-systems.

[31] „Github,“ Github, Inc., 2022. [Võrgumaterjal]. Available: https://github.com/about.

[32] J. Battersby, „How to download Draw.io Diagrams,“ 2020. [Võrgumaterjal]. Available: https://www.tomsguide.com/how-to/how-to-download-drawio-diagrams.

[33] R. Ansyori , N. Qodarsih ja B. Soewito, „A systematic literature review: Critical Success Factors to Implement Enterprise Architecture,“ *Procedia Computer Science,* kd. 135, pp. 43-51, 2018.

[34] S. Systems, „ENTERPRISE ARCHITECT,“ Sprax Systems, [Võrgumaterjal]. Available: https://sparxsystems.com/products/ea/index.html.

[35] D. Berkholz, „Docker Index Shows Continued Massive Developer Adoption and Activity to Build and Share Apps with Docker,“ 10 2 2021. [Võrgumaterjal]. Available: https://www.docker.com/blog/docker-index-shows-continued-massive-developer-adoption-and-activity-to-build-and-share-apps-with-docker/#:~:text=There%20are%20now%207.3%20million,%25%20year%2Dover%2Dyear..

[36] B. Karwin, SQL Antipatterns, Volume 1, Pragmatic Bookshelf, 2022.

[37] A. DeBarros, Practical SQL, 2nd Edition: A Beginner's Guide to Storytelling, No Starch Press, 2022.

[38] M. Prajapati, „Generic Repository & Unit Of Work Patterns in .NET,“ *B.E Computer Engineering.*

[39] S. Ahmed, „An authentication based scheme for applications using JSON web token,“ %1 *International Multitopic Conference (INMIC)*, 2019.

[40] M. UI, „Material UI,“ [Võrgumaterjal]. Available: https://mui.com/core/.

[41] J. Nilsen, „Heuristic evaluation of user interfaces,“ p. 8, 1990.

[42] „A Study on Understanding of UI and UX, and Understanding of Design,“ *International Journal of Applied Engineering Research,* kd. 12, 2017.

[43] Microsoft, „Language Integrated Query (LINQ) (C#),“ 09 03 2023. [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/.

[44] Microsoft, „DataContext Class,“ [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/dotnet/api/system.data.linq.datacontext?view=netframework-4.8.1.

[45] „Middleware,“ [Võrgumaterjal]. Available: https://www.techtarget.com/searchapparchitecture/definition/middleware.

[46] „Data Transfer Object,“ [Võrgumaterjal]. Available: https://martinfowler.com/eaaCatalog/dataTransferObject.html.

[47] Simplilearn, „What Is C# Entity Framework?,“ [Võrgumaterjal]. Available: https://www.simplilearn.com/tutorials/asp-dot-net-tutorial/entity-framework-in-c-sharp.

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

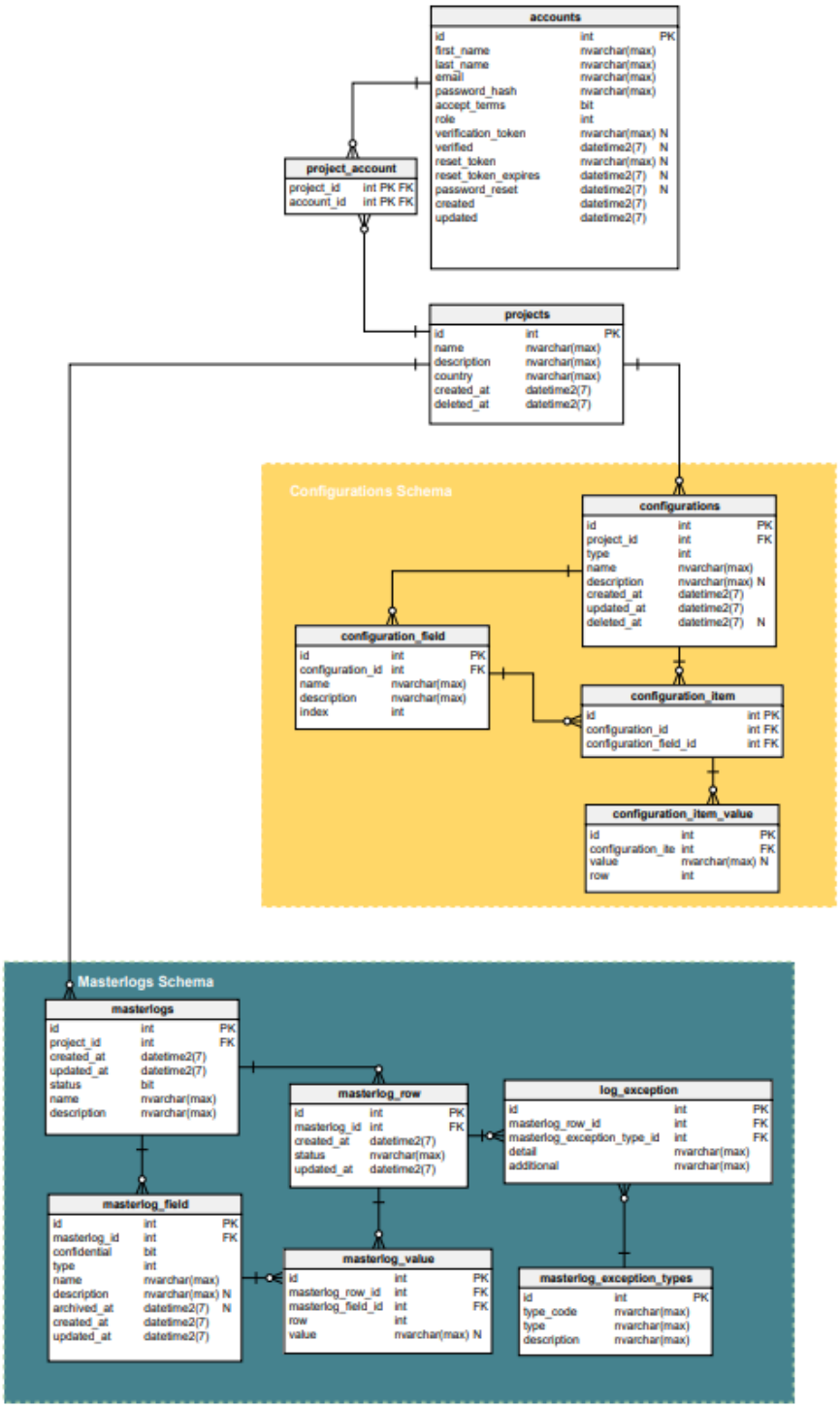
Mina, Gert Mänd

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Automatsioonide monitoorimise ja konfigureerimise platvormi disain ja prototüüpimine kindlustusettevõtte baasil“ mille juhendaja on Meelis Antoi
   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.

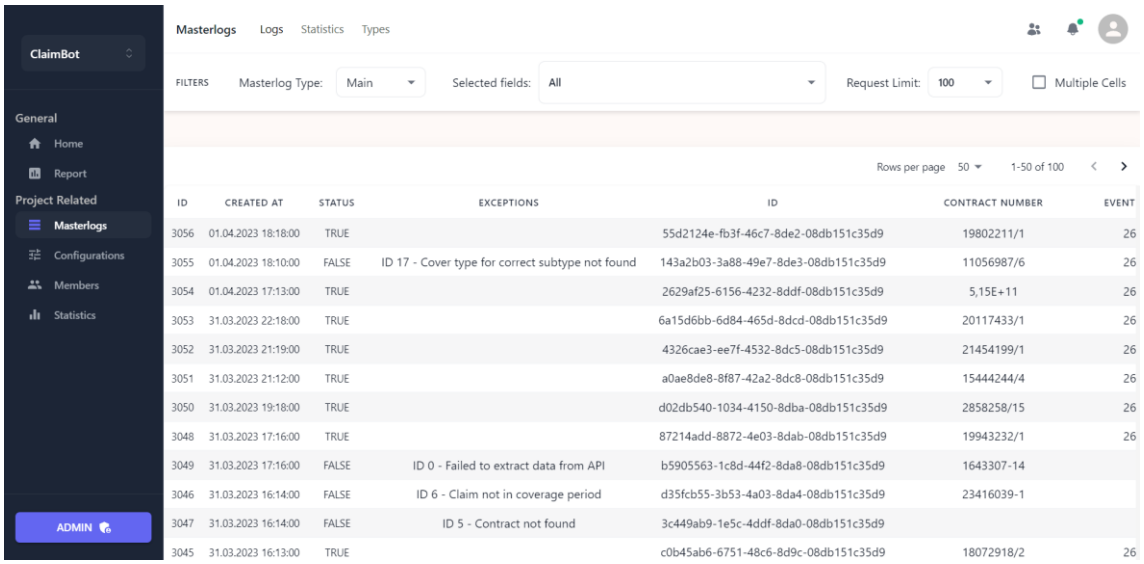
10.05.2023

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.

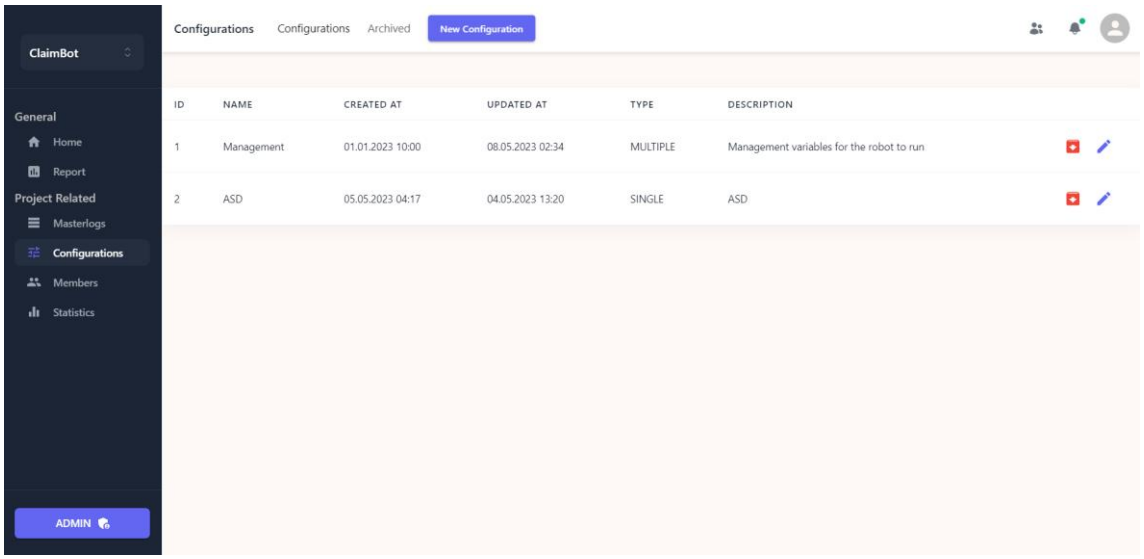
Lisa 2 – Olemi-suhte diagramm



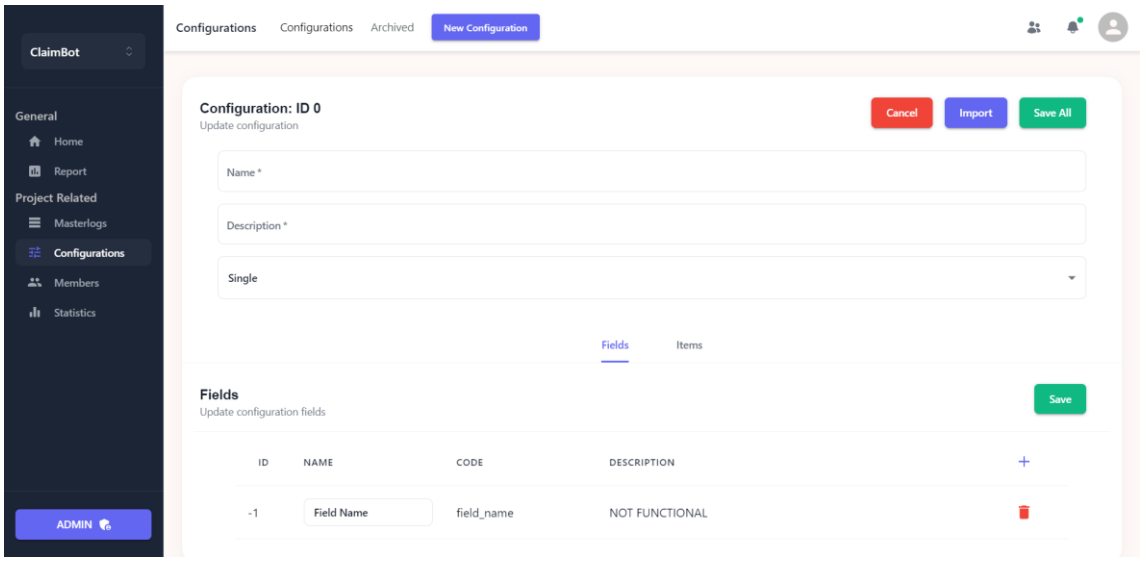
Lisa 3 – Kliendirakenduse vaated

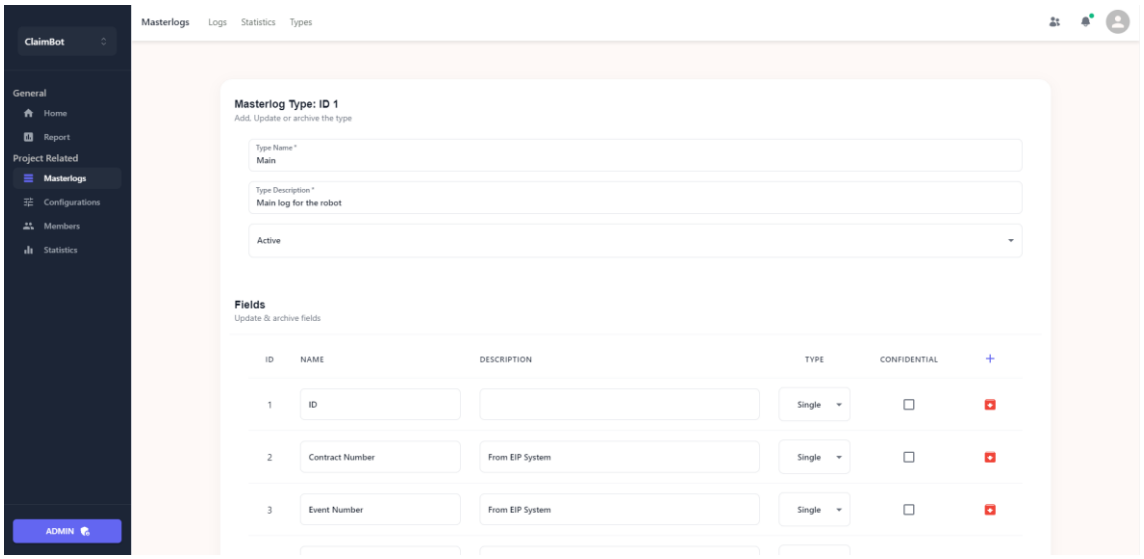


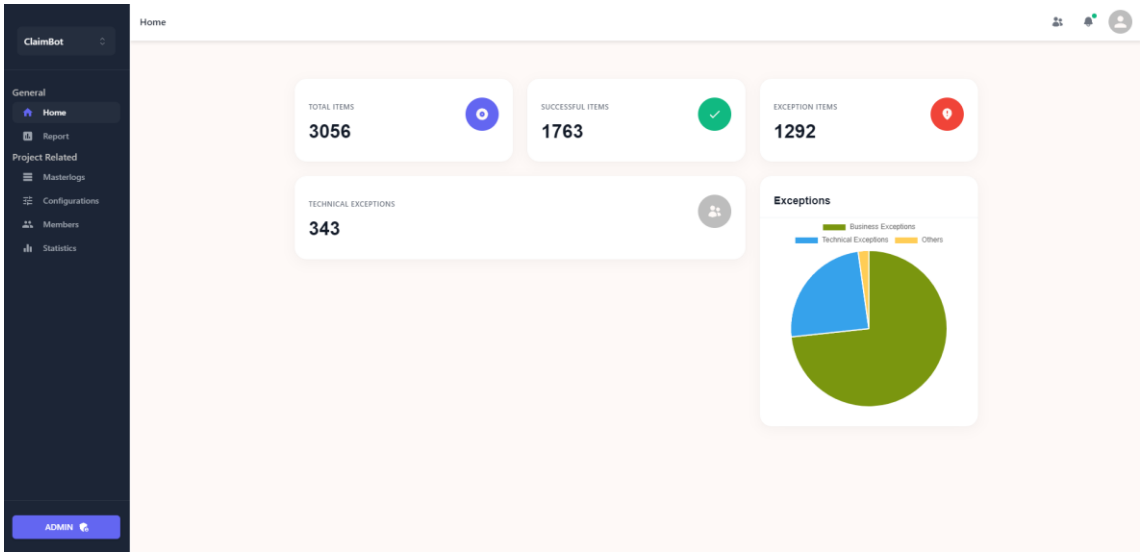
ID	CREATED AT	STATUS	EXCEPTIONS	ID	CONTRACT NUMBER	EVENT
3056	01.04.2023 18:18:00	TRUE		55d2124e-fb3f-46c7-8de2-08db151c35d9	19802211/1	26
3055	01.04.2023 18:10:00	FALSE	ID 17 - Cover type for correct subtype not found	143a2b03-3a88-49e7-8de3-08db151c35d9	11056987/6	26
3054	01.04.2023 17:13:00	TRUE		2629af25-6156-4232-8ddf-08db151c35d9	5,15E+11	26
3053	31.03.2023 22:18:00	TRUE		6a15d6bb-6d84-465d-8dcd-08db151c35d9	20117433/1	26
3052	31.03.2023 21:19:00	TRUE		4326cae3-ee7f-4532-8dc5-08db151c35d9	21454199/1	26
3051	31.03.2023 21:12:00	TRUE		a0ae8de8-8f87-42a2-8dc8-08db151c35d9	15444244/4	26
3050	31.03.2023 19:18:00	TRUE		d02db540-1034-4150-8dba-08db151c35d9	2858258/15	26
3048	31.03.2023 17:16:00	TRUE		87214add-8872-4e03-8dab-08db151c35d9	19943232/1	26
3049	31.03.2023 17:16:00	FALSE	ID 0 - Failed to extract data from API	b5905563-1c8d-44f2-8da8-08db151c35d9	1643307-14	
3046	31.03.2023 16:14:00	FALSE	ID 6 - Claim not in coverage period	d35fcb55-3b53-4a03-8da4-08db151c35d9	23416039-1	
3047	31.03.2023 16:14:00	FALSE	ID 5 - Contract not found	3c449ab9-1e5c-4ddf-8da0-08db151c35d9		
3045	31.03.2023 16:13:00	TRUE		c0b45ab6-6751-48c6-8d9c-08db151c35d9	18072918/2	26



ID	NAME	CREATED AT	UPDATED AT	TYPE	DESCRIPTION
1	Management	01.01.2023 10:00	08.05.2023 02:34	MULTIPLE	Management variables for the robot to run
2	ASD	05.05.2023 04:17	04.05.2023 13:20	SINGLE	ASD


ClaimBot
Configurations
Configurations
Archived
New Configuration
General
Home
Report
Project Related
Masterlogs
Configurations
Members
Statistics
ADMIN
Configuration: ID 0
Update configuration
Cancel
Import
Save All
Name *
Description *
Single
Fields
Items
Fields
Update configuration fields
Save
ID
NAME
CODE
DESCRIPTION
-1
Field Name
field_name
NOT FUNCTIONAL


ClaimBot
Masterlogs
Logs
Statistics
Types
General
Home
Report
Project Related
Masterlogs
Configurations
Members
Statistics
ADMIN
Masterlog Type: ID 1
Add, Update or archive the type
Type Name *
Main
Type Description *
Main log for the robot
Active
Fields
Update & archive fields
ID
NAME
DESCRIPTION
TYPE
CONFIDENTIAL
1
ID
Single
2
Contract Number
From EIP System
Single
3
Event Number
From EIP System
Single


ClaimBot
Home
General
Home
Report
Project Related
Masterlogs
Configurations
Members
Statistics
ADMIN
TOTAL ITEMS
3056
SUCCESSFUL ITEMS
1763
EXCEPTION ITEMS
1292
TECHNICAL EXCEPTIONS
343
Exceptions
Business Exceptions
Technical Exceptions
Others

Lisa 4 – Päringute tabel

Kirjeldus	Tüüp	Ressurss
Autentimine		
Kasutaja autentimine	POST	/accounts/authenticate
Tokeni uuendamine	POST	/accounts/refresh-token
Tokeni tühistamine	POST	/accounts/revoke-token
Kasutaja registreerimine	POST	/accounts/register
Parooli lähtestamine	POST	/accounts/forgot-password
Konfiguratsioonid		
Konfiguratsiooni päring	GET	/configurations/{id}
Konfiguratsiooni loomine	POST	/configurations/
Konfiguratsiooni uuendamine	PUT	/configurations/
Konfiguratsiooni tulpade eemaldamine	DELETE	/configurations/{id}/fields
Konfiguratsiooni tulba eemaldamine	DELETE	/configurations/{id}/fields/{fieldId}
Konfiguratsiooni väärtuse eemaldamine	DELETE	/configurations/{id}/items/{id}
Konfiguratsiooni arhiveerimine	POST	/configurations/{id}/archive
Konfiguratsiooni arhiveerimise tühistamine	POST	/configurations/{id}/unarchive
Logid		
Põhilogi päring	GET	/logs/masterlog/{masterlogId}
Põhilogi andmete päring	GET	/logs/masterlog/{masterlogId}/rows
Põhilogi uuendamine	PUT	/logs/masterlog/{masterlogId}
Põhilogi tulpade päring	GET	/logs/masterlog/{masterlogId}/fields
Põhilogi tüübi loomine	POST	/logs/masterlog/
Põhilogi tulba lisamine	POST	/logs/masterlog/{masterlogId}/field
Põhilogi tulba arhiveerimine	POST	/logs/masterlog/field/{fieldId}
Põhilogi andmete lisamine	POST	/logs/masterlog/{masterlogId}/row
Põhilogi andmete uuendamine	PUT	/logs/masterlog/{masterlogId}/row/{rowId}
Logide päring	GET	/logs/
Logide lisamine	POST	/logs/
Projektid		
Projekti päring	GET	/projects/{id}
Projekti loomine	POST	/projects/
Projekti kustutamine	DELETE	/projects/{id}
Projekti konfiguratsioonid	GET	/projects/{id}/configurations
Projekti baas statistika	GET	/projects/basic-statistics
Projekti põhilogid tüübid	GET	/projects/{id}/masterlogs
Projekti põhilogide andmed	GET	/projects/{id}/masterlogs/rows