TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Timothy Beres 164478IABB

ETTEVÕTTE ARENDUSTIIMIDE VAHELISE ARVELDUSE EFEKTIIVNE AUTOMATISEERIMINE LOGITUD TÖÖTUNDIDE PÕHJAL ETTEVÕTTE HELMES AS NÄITEL

Bakalaureusetöö

Juhendaja: Mart Roost
Magistrikraad

Tallinn 2020

Autorideklaratsioon

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

Autor: Timothy Beres

03.05.2020

Annotatsioon

Antud bakalaureuse töö eesmärk on analüüsida ettevõttes Helmes AS kasutusel olevat pooleldi automatiseeritud üksuste vahelist arveldamise protsessi, nimetusega dünaamiline crosbill ning probleeme selle kasutamisel ja leida neile lahendused. Autor lähtub probleemide tuvastamisel nii funktsionaalsuse kasutajate tagasisidest kui ka puudujääkidest protsessis vastavuses ettevõtte sisestele infosüsteemide arhitektuuritavadele. Töö eesmärgiks on luua efektiivne ehk probleemideta ja automatiseeritud üksuste vaheliste arvete loomise protsess.

Töö käigus on kirjeldatud ära tänane dünaamilise meeskondade vaheliste arvete loomise protsess AS-IS ning kujundatud vastavalt töö käigus tehtud analüüsile TO-BE protsess, mis lahendaks esinenud probleemid ning täidaks lisaks nõutavaid ärilisi eesmärke. Dünaamilise meeskondade vahelise arve protsessi loomise parandamist analüüsides jõudis autor järelduseni, et lisaks on vaja arendada kaks uut mikroteenust, mõlemale teenusele viidi läbi detailne süsteemianalüüs ning seati mõlemad projektid arendusvalmis.

Töö tulemusena loodud analüüsi põhjal teostati planeeritud arendused ning lahendati esinenud probleemid dünaamiliste arvete loomise protsessis. Lisaks loodi lisaväärtusena kaks uut mikroteenust Helmese sisesüsteemide võrku, mida saavad tarbida mitmed süsteemid, luues kasutajatele lisandväärtust ettevõttes olemasolevate andmete põhjal.

Lõputöö on kirjutatud Eesti keeles ning sisaldab teksti 49 leheküljel, 9 peatükki, 20 joonist, 2 tabelit.

Abstract

Effective automation of billing between different development teams within a company based on logged working hours on the example of Helmes AS

The goal of this bachelor's thesis is to analyse and improve the currently semi-automatic process of creating bills between different development teams in the company of Helmes. A bill is formed between different development teams, when the member of team A does work for team B in which case team B will have to compensate the time spent to team A. Users of this function have reported problems with the bills that have been made by the system, either there has been data missing or the data has been incorrect. Since this is a function which includes financial data, it is substantial that all the bills that are automatically formed, are accurate.

In the process of this thesis, the author formed process diagrams of the AS-IS bill creation and modeled the TO-BE process. Improvements in the process were made based on interviews with 15 users of the system and the architectural guidelines that are established for all internal systems in the company. With the aim to improve the current system to match the requirements of the users, the author concluded that two new microservices need to be developed which will both improve the current bill creation process and the central architecture of Helmes' internal systems.

As a result of this thesis, two new microservices were developed and the problems that were mentioned during the interviews were solved. In addition to improving the bill creation process, the new microservices created added value to other systems within the Helmes internal systems cloud, which could provide additional business value to the users based on the data that is provided from these services.

The thesis is in Estonian and contains 49 pages of text, 9 chapters, 20 figures, 2 tables.

Lühendite ja mõistete sõnastik

Jira	Tarkvaraarenduse projektijuhtimise tarkvara, väga kohandatav vastavalt ettevõtte vajadustele.
CIR	*Contract Information and Invoice Request System,* Helmes'e sisemiselt arendatud aplikatsioon.
Crossbill	Nii-öelda ostu-müügi arve kahe Helmese meeskonna vahel, kus müüjaks on meeskond, kelle töötaja tegi tööd teise tiimi projekti ning ostja on meeskond, mille projekti tööd tehti.
API	*Application Programming Interface,* arvuti operatsioonisüsteemiga või rakendusprogrammiga määratud reeglistik, mille alusel rakendusprogramm kasutab operatsioonisüsteemi või teise rakendusprogrammi teenuseid [1].
AD	*Active Directory*, keskne kasutajahalduse süsteem.
REST	*Representational State Transfer,* hulk arhitektuurilisi piiranguid mis moodustavad standardi andmevahetuseks [2].
JSON	*JavaScript Object Notation*, andmevahetusformaat [3].
Jwt	*Json Web Token*, avatud standard, mis pakub kompaktset ja truvalist viisi JSON formaadis andmete edastamseks [4][5].
SQL	*Structured Query Language,* programmeerimiskeel relatsioonilise andmebaasi haldamiseks.
AS-IS	Välja joonistatud protsessimudel enne muudatuste jõustumist.
TO-BE	Välja joonistatud protsessimudel peale soovitud muudatuste jõustumist.
Docker	Konteineritel põhinev rakenduste majutamise tööriist [6].
GUID	*Globally Unique Identifier*, automaatselt genereeritud unikaalne nimetaja [7], kasutusel andmete abstraheerimiseks.
Gateway	Värav, läbi mille liiguvad päringud erinevate (mikro)teenuste vahel. Tagab lisa turvalisuse andmete edastamisel ning lihtsuste teenuste tarbijale [8].
Dynamicsnav2016	Helmeses kasutusel olev raamatupidamisrakendus, mida arendab ja haldab Microsoft.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Käesoleva lõputöö teemaks on „Ettevõtte arendustiimide vahelise arvelduse efektiivne automatiseerimine tehtud ja logitud töötundide põhjal ettevõtte Helmes AS näitel“. Töö eesmärgiks on parandada meeskondade vahelist arveldamise protsessi, analüüsides tänaseid puudujääke dünaamilise arvelduse funktsionaalsuses ning andes vajalik analüütiline sisend arendustöödele. Antud projektis osaleb töö autor nii analüütiku kui ka projektijuhina, see töö keskendub rohkem analüütilisele poolele.

1.1 Taust ja Probleem

Ettevõte Helmes AS on kiirelt laienev Eesti juurtega tarkvaraarenduse firma, mille peakontor asub Tallinnas. Lisaks tarkvaraarendusega tegelevatele kontoritele Tallinnas, Riias ja Minskis kuulub Helmese alla veel mitmeid ettevõtteid, kes tegutsevad ettevõtte nime all iseseisvalt. Tänase seisuga – aprill 2020 – töötab Helmese nime all kokku üle 800 töötaja kuues eri riigis. Töö autor töötab Helmese Tallinna kontoris ettevõtte sisemiste arenduste meeskonnas projektijuhi ning analüütikuna, vastutades sisemiste arenduste tellimise ja arenduse juhtimise eest.

Helmes kasutab maailmas laialt tunnustatud meeskondade struktuuri (täpsemalt peatükis 2.2). Vastavalt peatükis 2.2 mainitud puudustele meeskonnastruktuuris, on Helmese puhul aktuaalne, kus kahe meeskonna vahel on vaja töötaja kulusid arveldada. Tiimide vahelised kulud arveldatakse meeskondade vahel arvena, mille nimetus on ettevõtte siseselt crossbill (täpsemalt peatükis 2.3). Antud protsessi jaoks on loodud vastav liidestus sisemiselt arendatud süsteemi CIR – crossbilli saab teha käsitsi või tekib see dünaamiliselt läbi liidestuse Jira andmebaasiga (juhul kui tööle kulunud aeg on logitud Jirasse). Probleem seisneb dünaamilise crossbilli suures sõltuvuses Jira andmebaasist, sest kaks aplikatsiooni suhtlevad otse ilma vahepealsete kihtideta. Dünaamiline crossbill on seotud vaid ühe ajalogimise rakendusega, kuigi ettevõttes on kasutusel ka teisi rakendusi, näiteks Toggl. CIR poolel on täna uue ajalogimise rakenduse liitmine ajakulukas, sest iga uus rakendus eeldab CIR poolset arendust just selle rakenduse jaoks.

See probleem on eriti aktuaalne kasvavas ja pidevalt laienevas ettevõttes, sest erinevates kontorites ja riikides on levinud erinevad ajalogimise rakendused. Tiimijuhid on raporteerinud ka vigu automaatselt Jira andmebaasi põhjal loodud crossbillides, näidates halba andmete kvaliteeti.

1.2 Töö eesmärgid

Antud töö eesmärkideks on:

1. kirjeldada ära detailselt AS-IS protsessimudel dünaamilise crossbilli tekkimisest;
2. analüüsida koostöös kasutajatega probleeme ja puudujääke tänases protsessis kasutajate jaoks;
3. tuvastada AS-IS protsessimudelis puudused, põhinedes kasutajate tagasisidele, analüüsida erinevaid võimalikke lahendusi tänase protsessi parendamiseks;
4. kirjeldada TO-BE protsessimudel dünaamilise crossbilli tekkimisest, koos kõigi vajalike lisaarendustega;
5. koguda tagasisidet ning analüüsida valminud lahenduse lõpptulemust**.**

1.3 Metoodika

Töö autor kasutab levinud analüüsi tavasid ning modelleerib küsimuse all olevast protsessist AS-IS protsessimudeli, et teha kindlaks, milline on protsessimudel enne muudatuste sisse viimist ning millistele teenustele ja andmeallikatele protsess põhineb.

Vigade tuvastamiseks esialgses protsessis viib töö autor läbi intervjuud Helmese tiimijuhtidega, kes on funktsionaalsuse põhilised kasutajad. Intervjuusid tehakse nii süsteemi aktiivsete kui ka vähem aktiivsete kasutajate, et saada ülevaade erinevate kasutusmustritega osapooltelt ning analüüsida iga vastaja kasutusharjumusi individuaalselt. Igalt vastajalt oodatakse vastuseid vastaja enda vaatepunktist, vältimaks olukorda, kus vastajad proovivad vastata kogu kasutajaskonna nimel, mille ohuks on tuletatud probleemid ning antud tulemus ei annaks reaalselt ülevaadet eksisteerivatest probleemidest. Protsessi analüüsimisel võtab töö autor arvesse kõiki ka kaudselt asjasse puutuvaid faktoreid, nagu näiteks ettevõtte struktuuri, meeskondade eripärasid jms [4].

Detailne analüüs tehakse nii praegusele dünaamilise crossbilli funktsionaalsusele kui ka tulevasele muudetud protsessile põhinedes muutunud funktsionaalsuste nõuetele kasutajate poolt. Autor kasutab UML notatsiooni, modelleerimaks dünaamilise crossbilli tänast ja tulevast protsessimudelit. Diagrammide loomiseks kasutatav tööriist on *draw.io* veebirakendus.

Peale arendustööde valmimist analüüsib autor soovitud tulemusi reaalsete tulemustega põhinedes põhilistele KPI-dele ning kasutajate tagasisidele. Samuti pakub ta välja võimalikke tulevasi arendusi protsessi edasiseks parandamiseks.

2 Teooria

Antud peatükk kirjeldab töös käsitletavaid teoreetilisi teemasid, kirjeldades Helmeses kasutusel oleva meeskonnamudeli omadusi ning puudusi, mis on ka antud lõputöö poolt käsitletava probleemi aluseks. Lõputöös on ka süvitsi käsitletud mikroteenuste mõistet, mistõttu peab töö autor vajalikuks antud teema teoreetilist tausta avada.

2.1 Meeskonnamudel

Autori poolt analüüsitavas ettevõttes on kasutusel ärimudel nimetusega meeskonnamudel (ingl k *dedicated team model*). Antud mudeli näol on tegemist aina rohkem kanda kinnitava ärimudeliga IT ettevõtete seas, olles vastukaaluks varem laialdaselt levinud ressursi jaotuse (ingl k *resource allocation*) mudelile. Meeskonnamudel on aktuaalne ettevõtetele, kes keskenduvad projektide edukatele läbiviimistele ja kliendi rahuolule selle asemele, et panna rõhku iga individuaalse töötaja väljamüügi maksimeerimisele [9]. Meeskonnamudeli põhiliseks tunnuseks on iga töötaja kuulumine kindlasse meeskonda ettevõtte siseselt, selle asemel et otse ettevõtte alla kuuluda [10]. Erinevalt jagatud ressursi mudelile, kus projekti tiime moodustatakse projekti põhiselt vastavalt projekti nõuetele, on meeskonna mudelis juba kokku töötanud meeskonnad, mida projektipõhiselt ei muudeta, vaid projektidesse müüakse iga inimese individuaalse kompetentsi asemel kokku töötanud meeskonna efektiivsust ja kogemust, ehk teenuse ostja ostab sisse kogu tarkvaraarenduse komplekti, alustades arendajatest ja lõpetades tehnilise toe spetsialistiga. Samuti saab klient olla kindel, et tema projektis töötavad inimesed on just sellele projektile orienteeritud ning ei jaga oma koormust mujal, mis võib tekitada takistusi projekti arenemise protsessis [11]. Suur roll meeskonnamudelis on meeskondade moodustamisel, mis on hea meeskonna jaoks võtme tähtsusega. Kuna meeskonnad on tihti teatud valdkondadele spetsialiseerunud, moodustatakse meeskonnad vastavalt sellele ning kliendi jaoks tekib partner meeskonnana, mis on hästi kokku töötanud ning omab teadmisi just temale aktuaalses valdkonnas, mis on aluseks pikkadele ning tugevatele kliendisuhetele [10].

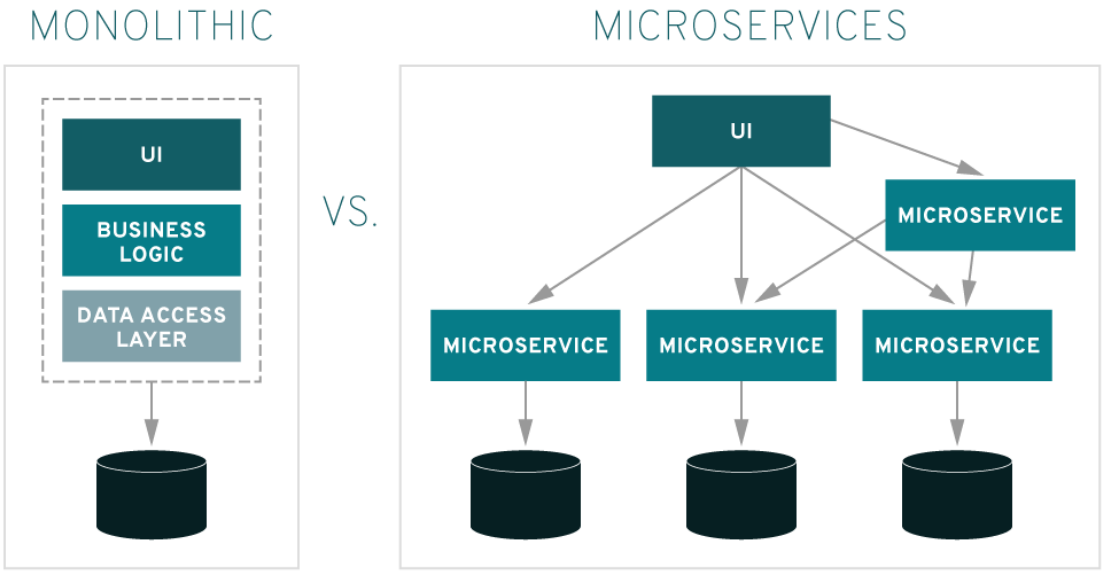
2.2 Meeskondade vaheline arveldus

Meeskonnamudel ei ole aga hõbekuul, vaid sellel on ka puudusi tarkvaraarendusettevõtetes. Üks mudeli tugevusi – töötajate sügavad teadmised oma tegevusvaldkonna kohta – on samal ajal ka selle nõrkuseks. On oht, et töötajate oskused ja teadmised stagneeruvad ning nad kaotavad huvi oma töö vastu, mis väljendub esialgu vähenenud efektiivsuses ning võib lõppeda töötaja lahkumisega ettevõttest [10]. Selle tõttu on meeskonna juhi eesmärk oma töötajaid aktiivsetena hoida ning neis huvi kadumisel see uuesti äratada suunates neid mõne teise meeskonna projekti, mis on uus väljakutse ning hoiab töötajat kaasatuna, arendades samal ajal nende oskusi uues valdkonnas. Teiseks puuduseks on teatud kompetentsi kogunemine kindlatesse meeskondadesse, mille vastu võib huvi olla ka mõnel teisel meeskonnal. Mõlema mainitud negatiivse aspekti juures tekib olukord, kus ühe meeskonna töötaja müüb oma tööd teise meeskonda, mis tekitab olukorra, kus kahe meeskonna vahel on vaja arveldada. Meeskonnastruktuuri järgi vastutab iga meeskond enda kliendi ning seeläbi ka finantside eest, iga tiim on sisuliselt eraldiseisev mikroettevõte koos oma varade ja töötajatega. See tähendab ka seda, et kõik tiimi kulud, sealhulgas töötajate palgakulud, makstakse tiimi tulude arvelt. Antud tiimimudeli omadus tekitab keerukuse olukorras, kus ühe meeskonna töötaja teeb tööd mõne teise meeskonna projekti. Teisele tiimile tehtud töö võrra vähem müüb töötaja oma meeskonna kliendile tunde välja, mis tähendab tema meeskonnale kulu alternatiivkulu vormis. Sellises olukorras peab meeskond, kelle jaoks tööd tehti, töötaja kulutatud aja tema meeskonnale kompenseerima, mis nõuab töötavat ja struktureeritud arveldussüsteemi. Olukord, kus mitme meeskonna vahel toimub arveldamine kellegi tehtud tundide arvelt, on ajalooliselt tekkinud omajagu arusaamatusi ning konflikte, mida ettevõttes siseselt tasub võimalusel vältida. Automatiseerides antud protsessi, on vähem võimalusi arusaamatusteks ning kogu protsess võtab ka meeskondadel vähem aega, mis tähendab lihtsamat inimressursi jagamist meeskondade vahel – mis omakorda on vajalik ärimudeli toimimiseks [12].

2.3 Mikroteenuste arhitektuur

Vastupidiselt ajalooliselt levinud monoliitsele tarkvaraarendusele, on IT-süsteemides aina levinumaks arhitektuurimustriks mikroteenuste kasutamine, mille suurimateks eestvedajateks on suured ettevõtted, nagu näiteks Netflix, Uber, Amazon ja paljud teised. Mikroteenuste arhitektuur seisneb suure ja mahuka rakenduse ehk monoliidi asemel

paljude väikeste rakenduste implementeerimises. Iga väikene rakendus ehk mikroteenus täidab vaid ühte ärilist eesmärki ning on seetõttu kergesti kasutatav ning iseennast seletav kõigile, kellel selle teenuse vastu huvi võib olla. Arhitektuur on aktuaalne IT-süsteemide võrgustikes, kus on palju erinevaid ärifunktsioone ning mitmed lõpptarbijad tarbivad samasugust informatsiooni – tänu mikroteenustele on võimalik üle-rakenduste arhitektuur ning seeläbi ka funktsionaalsuste taaskasutamine üle mitme rakenduse [13]. Joonisel 1 on näidatud põhiline erinevus mikroteenuste ja monoliidi vahel olukorras, kus mikroteenusel on ka kasutajaliides (ingl k *user interface* ehk *UI*). Siinkohal tuleb juurde märkida, et kasutajaliides ei ole kohustuslik tingimus mikroteenusele, vaid on üks komponent antud näites.



Joonis 1. Monoliit (ingl k *monolith*) võrreldes mikroteenustega (ingl k *microservices)* [14].

Paljude väikeste teenuste ülalpidamine on samuti ka vähem kulukas ja arusaadavam, eriti olukorras kus süsteemid arendajad koosnevad väikestest meeskondadest mis tunnevad oma konkreetset teenust ja teavad kuidas seda arendada, kuid ei pruugi teiste teenustega kursis olla. Kuna tegemist on võrgustikuga väikestest rakendustest, ei ole ükski rakendus liiga tugevalt ühegi teisega seotud, sidemed rakenduste vahel käivad vaid läbi iga teenuse andmevahetuskihi. Seetõttu on lihtne ühte süsteemi arendada või välja vahetada ilma kartuseta lõhkuda mõnda teist süsteemi, seni kuna edastatav andmeformaat samaks jääb. Olla mitte liiga tugevalt sõltuv ühest süsteemist on vajalik ettevõtete jaoks, et maandada ärilist riski. Samuti annab teenuste iseseisvus võimaluse rakendada erinevaid tehnoloogiaid vastavalt vajadusele ning arenduse kompetentsile, terviksüsteem ei ole

piiratud ühelegi kindlale andmebaasile või programmeerimiskeelele, luues vabalt areneva ning kergesti ajas kaasas käiva ’süsteemi’. Iga teenus on iseseisvalt arendatav ja reliisitav, mis eeldab ka mingil kujul pidevat tarnet (ing k *continuous delivery*), tänu millele saab vaid muudetava teenuse osa uuendada ilma teistele teenustele seisakuid (ingl k *downtime*) põhjustamata [13].

Mikroteenuste arhitektuur ei ole ka universaalselt õige lahendus igasse olukorda ning tal on omad puudujäägid.

- Esmalt on mikroteenuste haldamine monoliidiga võrreldes tunduvalt kulukam, sest iga teenust tuleb eraldi majutada ning suuremal osal teenustel on ka oma andmebaas, mis on kõige suurem ressursi hõivaja.

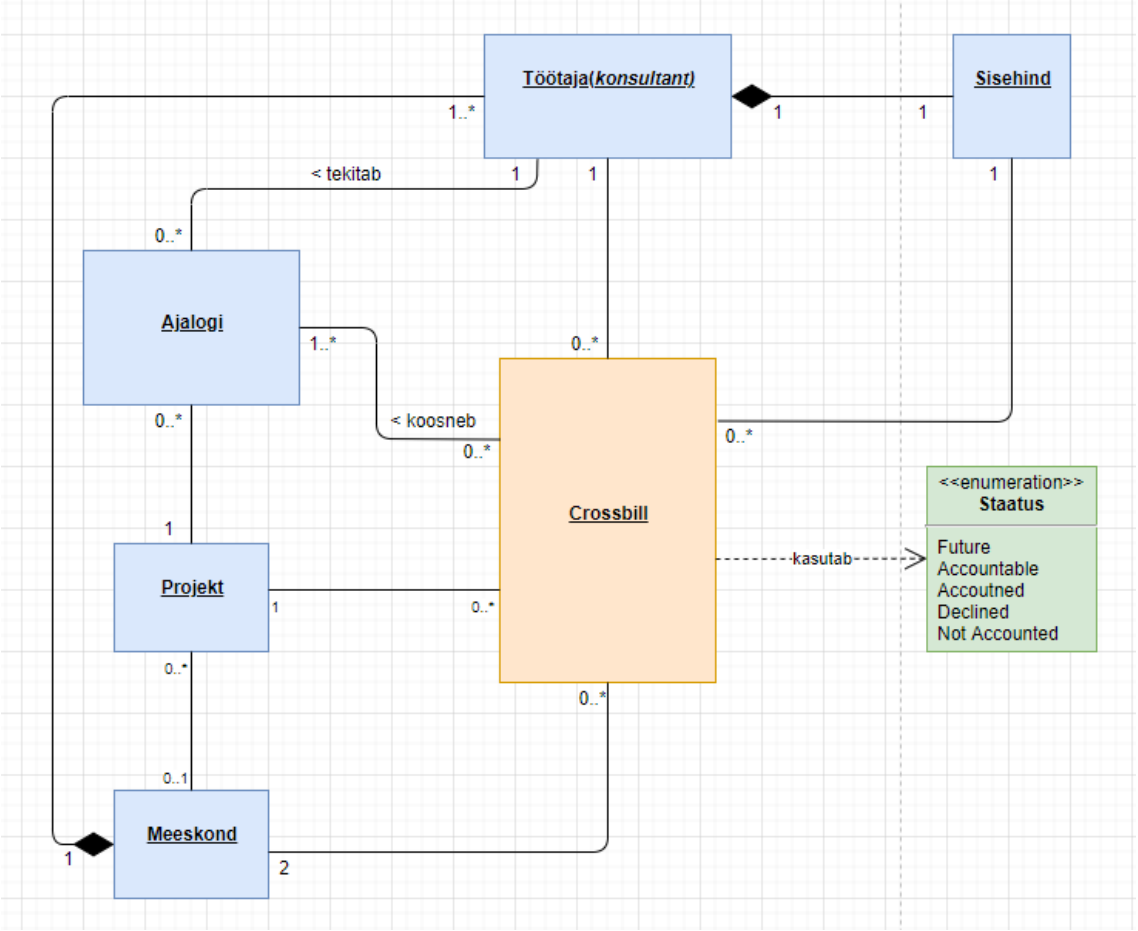
- Raskem on testida ja vigu otsida – kuigi ühe mikroteenuse keskselt on vastupidi lihtsam funktsionaalsusi testida, siis testides olukorda, kuhu kaasatud on mitu mikroteenust, tuleb teada kus asuvad millise teenuses logifailid ning silumine (ingl k *debugging*) on kasutamatu. Iga ühenduse testimise jaoks on tavaks kirjutada just selle kasutusjuhu põhised testid, mis on kulukas.

- Liiga mahukas arendus väikese kasu nimel. Kuigi mikroteenusel on suured eelised olukordades, kus rakendused peavad skaleeruma, ei pruugi see lahendus olla sobiv väiksematele ettevõtetele ning süsteemidele, kus erinevaid andmeallikaid ja protsesse on piiratud koguses. Kuna iga teenust tuleb eraldi arendada, on mikroteenuste arhitektuuri juurutamise protsess pikk ning kulukas, mis tasub ennast alles hiljem ära – väikese süsteemi korral on võimalus, et tasuvuspunkt on liialt kaugel.

Kaaludes mikroteenuste arhitektuuri, tuleks silmas pidada selle positiivseid ja negatiivseid aspekte ning langetada otsus arhitektuuri kasutamise osas vastavalt individuaalsele olukorrale ja rahalisele võimekusele [13].

3 Dünaamilise crossbilli kirjeldus

Dünaamiline crossbill on kahe meeskonna vaheline arve, mis moodustatakse CIR poolt automaatselt ning millel on vajalikud atribuudid, et kirjeldada mõlemale osapoolele tekkinud crossbilli põhjust, kogumaksumust ning kogumaksumuse kujunemist. Antud funktsionaalsus võeti Helmeses kasutusele 2017 aastal ning on juba tõestanud oma vajalikkust Helmese struktuuriga ettevõttes, säästes nii osapoolte tööaega antud protsessis kui ka parandades osapoolt arusaamist omavahelisest arveldamisest. Joonisel 2 on kujutatud crossbilli äriarhitektuuri mudel domeeni klassidiagrammina [15].



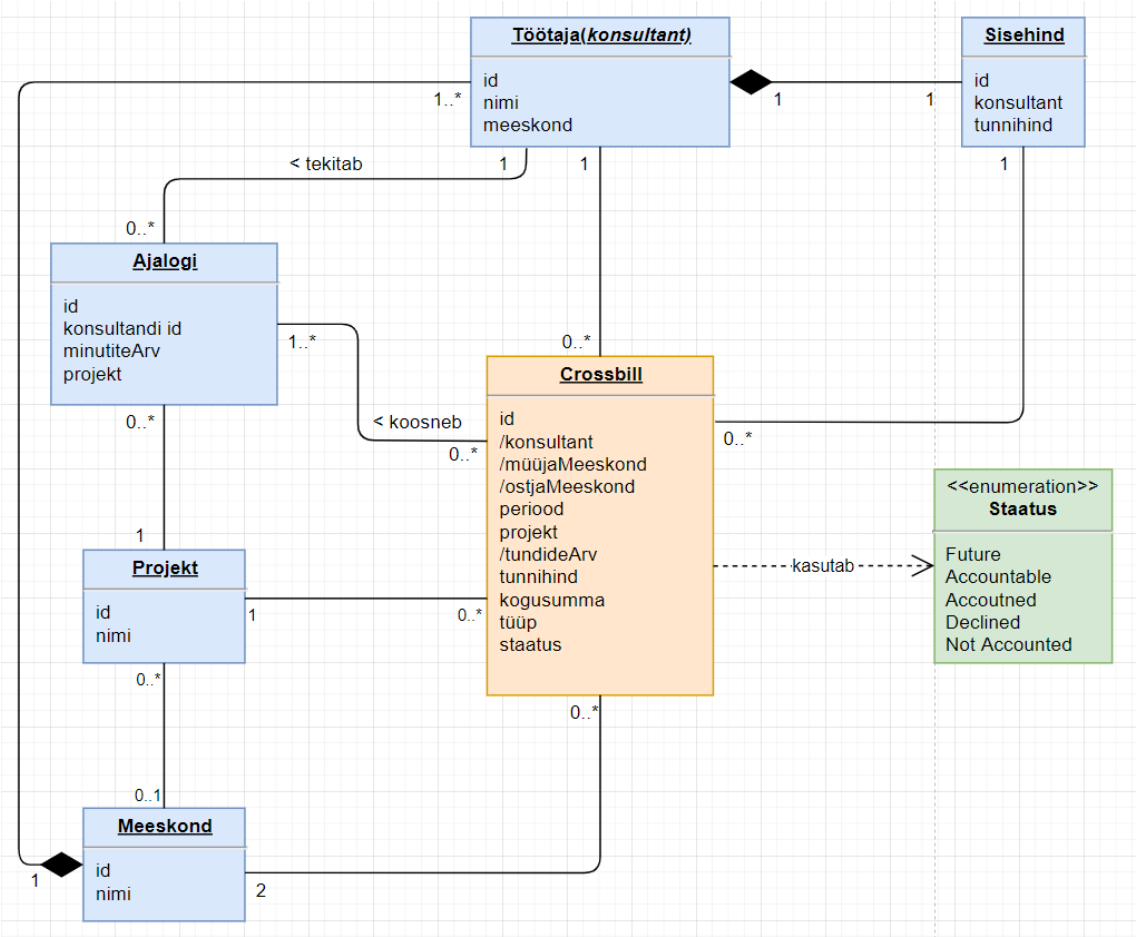
Joonis 2. Domeeni klassidiagramm dünaamilise crossbilli äriarhitektuuri kohta [15].

3.1 Tänased andmeallikad dünaamilise crossbilli tekkimiseks

Dünaamiline crossbill pannakse Helmese sisemiselt arendatud süsteemi CIR platvormil jooksva teenuse *Crossbill Service* poolt kokku, põhinedes mitmele erinevale andmeallikale. Ühe crossbilli komponentideks on:

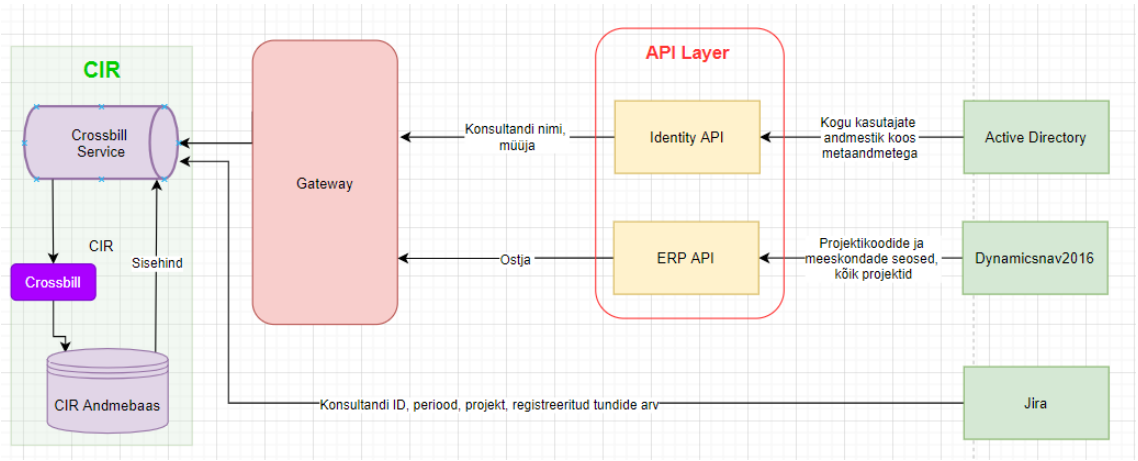
1. töötaja, kes tegi tööd ehk registreeris oma tunnid Jiras (edaspidi konsultant);
2. meeskond, kuhu konsultant kuulub(edaspidi müüja);
3. meeskond, mille alla kuuluvasse projekti konsultant töötunnid registreeris(edaspidi ostja);
4. periood, millal töö tehti – kuise täpsusega ehk näitab kindlat kuud mingil kindlal aastal;
5. projekt, kuhu konsultant oma töötunnid registreeris;
6. registreeritud tundide arv konsultandi poolt;
7. konsultandi eest makstav tunnipõhine summa, mis on määratud igale Helmese töötajale ning mida haldab keskselt Helmese raamatupidamise osakond**(**edaspidi sisehind)**;**
8. crossbilli kogu summa;
9. crossbilli tüüp;
10. crossbilli staatus.

Joonisel 3 on välja toodud kõrgel tasemel crossbilli komponentide andmemudel, mis kirjeldab andmeobjektide seoseid ja rolle crossbilli loomisel. Mudeli loomisel on järgitud UML raamistiku poolt seatud tavasid ettevõtte äriarhitektuuri modelleerimisel [15] .



Joonis 3. Komponentide andmemudel crossbilli tekkimisel. Crossbilli atribuudid, mille ees on '/' märk, on tuletatud põhinedes mõnele teisele väljale [15].

Komponendid crossbilli loomiseks kogutakse neljast erinevast süsteemist: CIR oma andmebaas, Jira väline andmebaas ning kaks API teenust: Identity API ja ERP API. Joonisel 4 on välja toodud, mis rakendustest millised vajalikud komponendid Crossbill Service teenusesse jõuavad.



Joonis 4. Crossbilli loomiseks kasutatavad andmeallikad.

Atribuudid tüüp ja staatus määratakse crossbilli tekkimisel automaatselt. Tüüp on dünaamilisel crossbillil alati Jira ning staatus määratakse vastavalt äriloogilistest nõuetest.

3.1.1 Crossbilli staatused

Igal crossbillil on staatus, mis määrab tema oleku ning asukohta arveldamise protsessis. Kokku on staatuseid viis:

1. *Future* – antud crossbill võetakse arvesse praeguse perioodi lõpus, ehk käib kestva perioodi kohta;
2. *Accountable* – antud crossbill võetakse arvesse eelmise perioodi lõpus, ehk käib eelmise perioodi kohta;
3. *Accounted* – antud crossbill on arvesse võetud, muudatusi teha ei saa;
4. *Declined* – antud crossbill on tagasi lükatud, ei võeta arvesse;
5. *Not Accounted* – antud crossbill tehti perioodi peale perioodi sulgemise kuupäeva, ei võeta arvesse;

Crossbilli periood kestab iga kuu esimesest päevast kuni kuu viimase päevani. Arveid eelmisesse perioodi ehk kuusse peale selle kuu lõppu saab teha kuni järgmise kuu kümnenda kalendripäevani. Iga kuu kümnendal kalendripäeval läheb eelmine periood lukku ning kõik selle perioodi *accountable* staatusega crossbillid võetakse raamatupidamises arvele. Crossbillide staatuseid uuendab süsteem automaatselt vastavalt üles seatud (ingl k *task scheduler*) programmile. Staatuseid uuendatakse kahel erineval kuupäeval

1. iga kuu 1. kuupäeval muudetakse kõikide *Future* staatusega crossbillide staatus *Accountable* staatuseks;
2. iga kuu 10. kuupäeval muudetakse kõikide *Accountable* staatusega crossbillide staatus *Accounted* staatuseks.

Crossbilli loomisel vaadatakse crossbilli loomise kuupäeva ning perioodi kuhu see crossbill kuulub, selle alusel määratakse sellele staatus vastavuses eelpool mainitud äriloogikale.

3.2 Dünaamilise crossbilli loomise protsess AS-IS

Uute dünaamiliste crossbillide loomiseks alustatakse CIR poolelt kasutaja poolt crossbillide sünkroniseerimise protsess. Esmalt pöördub *Crossbill Service* Jira andmebaasi poole ning pärib sealt kõik registreeritud ajalogid, mis asuvad soovitud ajavahemikus. Järgnev protsess toimub *n* korda, *n* tähistab erinevate ajalogide arvu. Iga ajalogi on unikaalne kolme tunnuse poolest, mille kombinatsioon ei kordu:

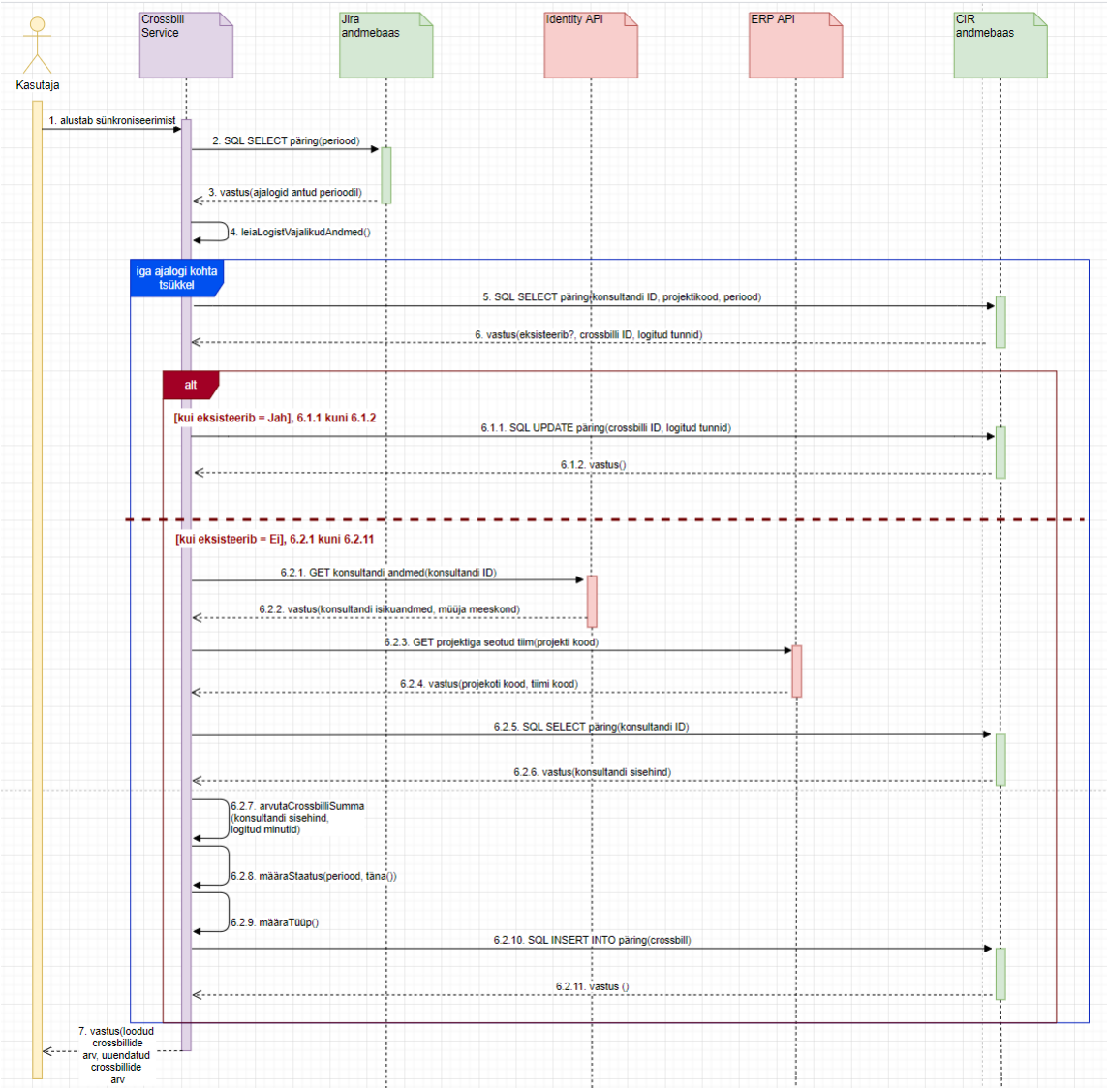
- konsultandi ID;
- projektikood;
- periood.

Iga ajalogi omab esmalt selle loonud konsultandi ID'd, töö perioodi, projekti koodi, kuhu aeg logiti ning logitud minutite arvu. Siin jaguneb loomis protsess kaheks, vastavalt saadud ajalogi sisule:

a) Kui leitakse, et antud ajalogi põhjal on juba crossbill loodud, uuendatakse juba olemasoleva crossbilli mahtu ehk tehtud tundide summat ning ka crossbilli kogusummat. Ajalogile vastab crossbill on olemas juhul, kui eelpool mainitud 3 tunnuse kombinatsioon eksisteerib,

b) kui antud ajalogile ei ole juba andmebaasis olevat vastet, luuakse täiesti uus crossbill.

Ajalogide põhjal luuakse crossbill vastavalt peatükis 3.1 kirjeldatud andmete liikumise loogikale, pöördudes erinevate teenuste vastu. Crossbilli staatus määratakse vastavalt peatükis 2.1.1 kirjeldatud äriloogikale. Tüüp on dünaamilise crossbilli korral alati *Jira*. Joonisel 5 on kujutatud järgnevusdiagramm, mis kirjeldab crossbilli loomise loogikat AS-IS, lähtudes UML pool seatud tavadest järgnevusdiagrammi (ingl k *sequence diagram*) loomiseks [16].



Joonis 5. Järgnevusdiagramm crossbillide sünkroniseerimise kohta [16].

3.3 Toimingud peale crossbilli tekkimist

Peale seda kui crossbill on platvormile lisatud – kas läbi sünkroniseerimisprotsessi või käsitsi – käsitletakse seda tõesena seni kuni see on tagasi lükatud. Ehk igal tiimijuhil on võimalus endale esitatud arveid tagasi lükata kuni perioodist järgmise kuu kümnenda kuupäevani, peale seda võetakse crossbill olemasoleval kujul arvesse. Kui crossbill on tagasi lükatud, ehk staatused *Declined*, seda arvesse ei võeta .

Iga kuu kümnendal päeval, peale eelmise perioodi lukku minekut, eksporditakse CIR süsteemist kõik crossbillid Exceli failina, mille järel laetakse see üles raamatupidamise

aplikatsioon *Dynamicsnav2016*. Exceli põhjal jaotatakse kulud ja tulud juba selles programmis automaatselt meeskondade vahel.

3.4 Probleemid tänases protsessis

Probleemide tuvastamiseks küsitles autor 15 Helmese tiimijuhti, kes on aktiivsed crossbill funktsionaalsuse kasutajad. Intervjuud viidi läbi vabas vormis ning iga intervjuu oli ligikaudu 45 minuti pikkune, mille käigus lasi autor küsitletavatel kirjeldada nende kogemust dünaamiliste crossbillidega viimase 7 kuu jooksul. Tagasisidet koguti vaid viimase 7 kuu kohta, et vastajad põhineksid värsketel kogemustel, eesmärgiga välistada üksikjuhtumeid, millel ei pruugi jätkuvat ärilist mõju olla [17]. Vastajaid oli kahest Helmese kontorist, 11 vastajat Tallinna ning 4 vastajat Minski kontorist. Lisaks on autoril jooksvalt kogutud tagasiside CIR süsteemi kohta piletite vormis, mida hoiustatakse Jira projektis.

3.4.1 Kasutajate tagasisides mainitud probleemid

Allpool loetletud probleeme nimetasid vastajad kõige sagedamini. Peale esimest intervjuude ringi palus autor jätkukohtumisel terve valimiga järjestada nimetatud punktid prioriteedi järgi kasvavas järjekorras, prioriteedi määramisel arvestati probleemi ärilist mõju vastajate meeskonnale – olgu see siis alternatiivkulu, otsese kulu või protsessile kulutatud lisandunud aja vormis. Sama tulemust toetasid ka viimase 7 kuu jooksul raporteeritud probleemid CIR kohta.

1. kui meeskonna töötaja vahetab arveldamisperioodi jooksul meeskonda, lähevad antud perioodi tulud valesse meeskonda;

2. dünaamiliselt loodud crossbillide tundide maht erineb tegelikkusest;

3. Jirasse logitud ületunde crossbillide moodul ei arvesta, kuigi Jira pool on ületundide märkimise võimalus olemas;

4. ajalogisi sünkroniseeritakse vaid Jira’st, kasutusel on ka teisi ajalogimise rakendusi;

5. kasutajatel läheb meelest õigeks kuupäevaks tehtud crossbillid üle kontrollida;

6. kohati on arusaamatu, millest täpsemalt dünaamiline crossbill on tekkinud, tekitades arusaamatust kahe tiimijuhi vahel;

7. mõnes olukorras tekib crossbill meeskonna siseselt, ehk töötaja logib aega oma meeskonna projekti, mille tagajärjel crossbill tekkida ei tohiks;

8. on tekkinud duplikaat crossbille;

Viieteistkümnest vastajast 13 olid kokku puutunud olukorraga, kus nende meeskonna töötaja liikus teise meeskonda ning selle tulemusena sai automaatselt valmistatud crossbilli müüjaks määratud vale meeskond. Vaadates joonist 5 saab selgeks, et probleem on tekitatud asjaolust, et müüja määratakse selle järgi, mis meeskonda kuulub konsultant crossbillide sünkroniseerimise ajahetkel ja mitte töö tegemise ajal. Antud olukorras on protsessi kitsaskohaks sõltuvus sünkroniseerimise ajahetkest, samal ajal pole sünkroniseerimise alustamine kasutajate jaoks mitte mingil viisil piiratud.

Kõik vastajad olid märganud, et CIR'i automaatselt tekkinud crossbill ei sisalda sama töö mahtu, mis oli logitud Jira süsteemi. Tegemist on kõrge ärilise riskiga, sest kasutajatel on lihtne märkamata jätta puudu olevaid tunde olemasolevas crossbillis, mis on paljudel juhtudel lõppenud osaliselt puuduliku arvega. Autor tõdes selle punkti puhul võimalust, et puudu olevate tundide põhjuseks võib olla ka õigel ajal tegemata jäänud sünkroniseerimine – probleemiks on vajadus sünkroniseerimine käsitsi alustada. Lisaks on ohuks veel otse Jira andmebaasi vastu käiv SQL päring, mis on väga mahukas ning ei kohane, kui Jira poolel peaks toimuma muudatusi. Probleemi edasi uurides tõdes autor, et dünaamilised crossbillid on kohati puudulikud.

Kuigi tegemist on üsna harva kasutusjuhuga, oli üle poolte vastajate mureks logitud ületundide mitte arvestamine crossbillis, ehk kõik ületunnid arvestatakse tavalise tunnihinnaga. Antud viga on autorile ka varem teada olnud, kuid on jäänud parandamata kõrge arenduse maksumuse tõttu – ajalogid tulevad Jira kaudu otse läbi mahuka SQL päringu ning muudatused selles protsessis ei ole vaid ületundide jaoks ennast õigustanud. Antud probleemi põhjus on aluseks ka prioriteedilt neljandale probleemile ehk dünaamilise crossbilli sõltuvus vaid Jira andmebaasist. Kuigi selline liidestus oli süsteemi loomise hetkel piisav lahendus, ei skaleeru see tänases olukorras edasi ning kahe süsteemi otsene liidestamine saab takistuseks edasistele arendustele ja piirab crossbilli funktsionaalsuse kasutusjuhtusid. Antud liidestuse keerukus on aluseks ka teistele

probleemidele, mida kasutajad mainisid – probleemi korral on kulukas vea koht tuvastada ja arendusega viga parandada, näiteks duplikaat crossbillide ja meeskonna siseste crossbillide korral. Ka probleemi parandades on oht, et alati tekib uus probleem, sest tänane arendus ei ole läbipaistev ning kergesti loetav uute arendajate poolt.

Viimase murena toodi välja arusaamatusi ja kommunikatsiooni probleeme. Näiteks on kasutajatel lihtne unustada perioodi crossbille üle kontrollida, et tuvastada vigu tekkinud crossbillides. Täna ei ole neil selle kohta ka ühtegi meeldetuletust. Samuti on arusaamatusi, millest on crossbill tekkinud. Aega logitakse kindlatesse Jira piletitesse, kuid CIR crossbilli liidestusse see info ei jõua – kõik vastajad näeksid selles väärtust, kui crossbill kuvaks mingil moel pileti infot, kuhu aega logiti.

3.4.2 Vastuolud ettevõtte sisesüsteemide arhitektuuriga

Helmese sisesüsteemide arenduses on kehtestatud arhitektuuri nõuded kõigile loodavatele süsteemidele ning teenustele. Eesmärk on luua pilv API teenustest, mille kaudu erinevad rakendused suhtlevad, ilma otse üksteise poole pöördumata. Antud olukorras suhtlevad omavahel kaks eraldi seisvat rakendust, kasutamata selleks ühtegi vahekihti. See tekitab ärilise riski ettevõttele – oht on olla liiga tugevalt seotud ühe süsteemiga, eriti süsteemiga mis ei ole Helmese enda ülal peetud. Täna oleks väga keerukas Jira välja vahetada mõne teise süsteemi vastu ilma suurte arendusteta temaga seotud süsteemides, seega on teoreetiline risk, et ettevõte on sunnitud kasutama endale enam mitte sobivat või tasuvat süsteemi [18].

4 Protsessi parandamise analüüs

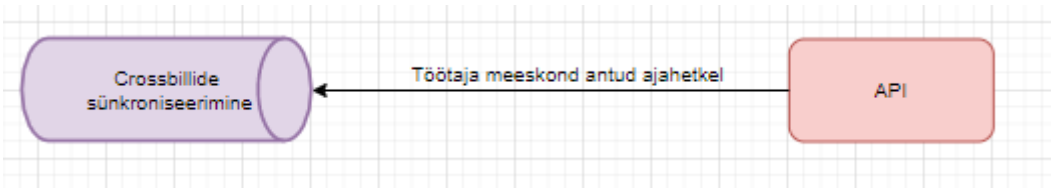
Antud peatükis viib autor läbi detailanalüüsi olemasolevatele probleemidele koos vajalike muudatustega dünaamilise crossbilli protsessis.

4.1 Probleemide kategoriseerimine

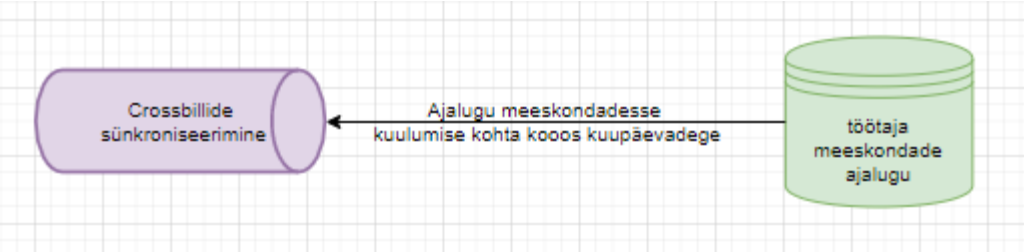
Autor koondas kasutajate mainitud probleemid kokku põhinedes nende probleemide aluseks olevatele puudustele süsteemis. Antud hetkel jäeti kõrvale kõik probleemid, mille aluseks ei saa määrata kindlat puudust tänases protsessis, näiteks mured mis tulenevad süsteemi puuduliku juurutamise põhjustatud arusaamatustest. Sellised probleemid on lahendatavad funktsionaalsuse dokumentatsiooni täiustamise või vajadusel uue juurutustsükli organiseerimisega.

4.1.1 Töötajate tiimi kuuluvuse ajalooline andmestik

Vastajate poolt kõrgeima prioriteediga märgitud probleem tänases dünaamilise crossbilli protsessis on puuduv informatsioon töötajate tiimi kuuluvuse ajaloo kohta. Seetõttu määratakse süsteemi poolt ekslikult crossbilli müüja meeskonnaks vale meeskond juhul, kui töötaja on perioodi jooksul meeskonda vahetanud. Antud olukord ei ole ka Helmese siseselt harva esinev. Selgelt tuleb välja, et dünaamilise crossbilli funktsionaalsuse jaoks on vaja hoiustada töötajate tiimi kuuluvuse ajalugu, millega crossbilli koostamisel arvestada saaks. Joonisel 6 on kujutatud väga kõrgel tasemel esmalt tänane müüja leidmise loogika ning joonisel 7, kuidas see tulevaselt peaks toimima.



Joonis 6. Tänane müüja leidmise loogika dünaamilise crossbilli loomisel.



Joonis 7. Tulevane müüja leidmise loogika dünaamilise crossbilli loomisel.

Joonise 7 puhul pöördub protsess crossbilli müüja meeskonna leidmiseks ajalooliste andmete poole ning määrab meeskondliku kuuluvuse ajalogi registreerimise ajahetkel. See kinnitab, et töötaja logitud töö eest saab tasu meeskond, kuhu töötaja töö tegemise ajal kuulus.

4.1.2 Ajalogide pärimise loogika

Suurem osa vastajate muresid on seotud ajalogide andmete puudulikkuse või ebatäpsusega. Kasutajate probleemid sellel teemal jagunevad kaheks – vale info ning puudlik info. Kuigi iga probleemi on võimalik tänases protsessis iseseisvalt käsitleda, koondab autor need kokku vigadeks ja puudusteks tänases ajalogide pärimise protsessis. Võttes lisaks arvesse antud protsessi vastuolusid Helmese sisesüsteemide arhitektuuriga ja piiratust ühele ajalogimise rakendusele, tuleb terve protsess ümber modelleerida ning mõistlik on kõik probleemid lahendada uues protsessis, selle asemel et neid ükshaaval lahendama hakata.

4.1.3 Meeldetuletuste lisamine

Seoses kiire ajalise graafikuga ning mitmete erinevate sisesüsteemide kasutamisega on tiimijuhtidel oht unustada kuu alguses kontrollida eelmise kuu loodud crossbille nende tiimile, mis on tekitanud olukorras, kus vigased või ebatäpsed crossbillid on jõudnud raamatupidamisse. Meeldetuletus iga kuu alguses oleks piisav, et tiimijuhtide tähelepanu antud protsessile juhtida.

4.2 Ettevõtte sisesüsteemide arhitektuur

Helmese sisesüsteemide arenduses on kehtestatud arhitektuurilised juhendid, mida tuleb järgida iga uue süsteemi või funktsionaalsuse arendamisel. Juhendid on kõrgel tasemel määratud head tavad, mida järgides on tagatav iga arendatava süsteemi kvaliteet ning korrektne ja turvaline andmekäitlus. Töö autor lähtub antud tavadest planeerides

muudatusi olemasolevale süsteemile. Sisesüsteemide arenduses kasutatav arenduse ressurss tuleb sisesüsteemide arenduse meeskonnale Helmese seest – tähendab arendust 'ostetakse' mõnel majasiseselt arendusmeeskonnalt vastavalt sellele, kellel on ressurssi üle. Tänu sellele eripärale muutuvad ka projektide arendusmeeskonnad tihedalt ning iga arendust planeerides peetakse silmas ohtu, et süsteemi arendaja võib süsteemi elu jooksul mitu kord vahetuda. Selle tõttu on tähtis puhas ja arusaadav süsteemide arhitektuur [18].

Kõikide uute süsteemide arendamisel on esmatähtis kinni pidada mikroteenuste arhitektuuri põhimõttest, sellest täpsemalt peatükis 2.3.

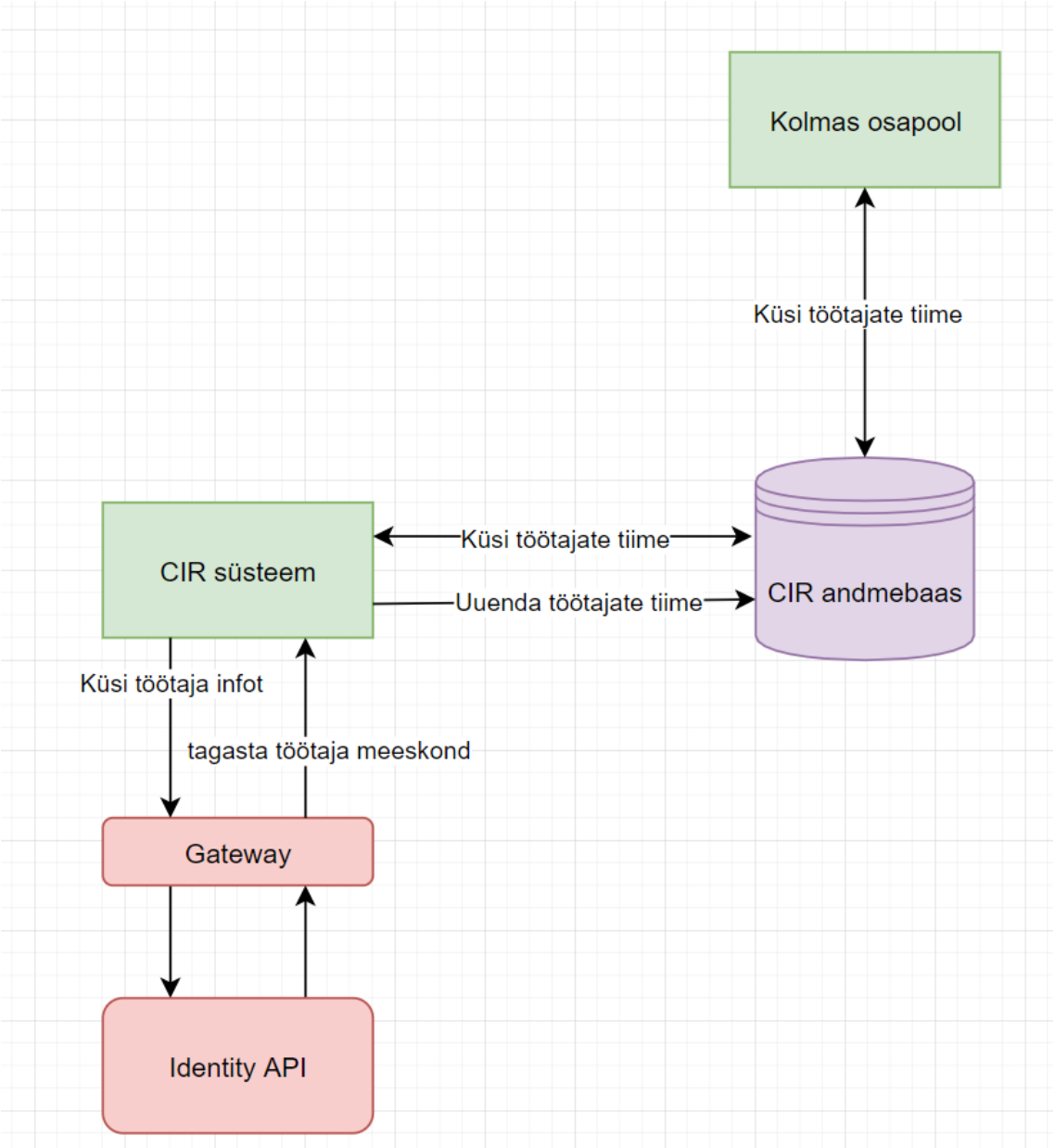
Suur osa Helmese sisemisest IT arhitektuurist on turvaline andmekäitlus ning keskselt hallatavad õigused erinevatele andmestikele. Keskse kasutajahaldustööriistana on kasutusel *Active Directory*, läbi mille toimub ka kogu õiguste haldus süsteemide vahel. Iga sisemine rakendus või mikroteenus vajab teise süsteemi poole pöördumiseks teenuskasutajat, millele on vastav õigus antud. Õigused tulevad teenuskasutaja küljes olevast rolligrupist, mida hallatakse Helmese sisemise IT meeskonna poolt. Igal süsteemil on vaja enda nimelist rolligruppi, formaadis *Role – API GW – {süsteemi nimi}*. Selleks, et hoida korda olukorras, kus erinevaid teenuseid on palju ning süsteemide arendajatel ja kasutajatel võib esineda raskusi soovitava teenuse leidmisega, on vaja keskset pidepunkti, läbi mille kogu rakenduste vaheline suhtlus käiks. Kasutades *gateway* teenust, ei pea lõpp-kasutajad teadma iga eraldi seisva teenuse aadressi, vaid saab sinna pöörduda läbi *gateway*'s asuva alam-aadressi. See annab võimaluse ühes kohas autentida teenuste poole pöördujaid, ilma vajaduseta sama loogikat igas mikroteenuses eraldi kasutada, mis teeb iga teenuse arenduse vähem kulukaks keeruka õiguste haldamise äriloogika rakendamise arvelt. Samuti säästab antud lahendus aega lõpp-kasutajate jaoks, kelle jaoks piisab ühekordsest autentimisest, et pääseda ligi kõikidele autoriseeritud süsteemidele [8]. *Gateway* poole pöörduvad kasutajad autenditakse nende kasutajanime ning parooli järgi – peale edukat autentimist tagastatakse automaatselt genereeritud *jwt*, mis sisaldab andmeid, millistele süsteemidele on kasutaja autoriseeritud ligi pääsema [5]. Kasutades autentimise päringust saadud *jwt*, saab kasutaja pöörduda läbi *gateway* kõigi autoriseeritud süsteemide poole. Lisaroll *gatewayl* on jagada koormust ühe teenuse erinevate instantside vahel. Täna on Helmeses kasutusel *Zuul Gateway*, mis on Netflixi poolt arendatud ning tänaseks avatud lähtekoodiga rakendus, mis on mõeldud kasutamiseks paljude API teenustega ettevõtetes, et reguleerida andmeliiklust rakenduste vahel [19].

4.3 Töötajate tiimi kuuluvuse andmestiku loomine

Vastavalt vajadusele on vaja hoiustada keskselt Helmese töötajate meeskonda kuuluvuse ajalugu. Autor analüüsis kahte varianti andmete hoiustamiseks ning hindas mõlema variandi tugevusi ning puudusi. Mõlemad variandid kujutavad endast automaatset andmete kogumist ning töötlemist, põhinedes olemasolevatele andmekanalitele. Autor välistas töötajate ajaloo käsitsi uuendamise, mille suureks puuduseks on protsessile kuluv tööaeg ning samuti läheks käsitsi uuendamine aina tülikamaks, mida suuremaks ettevõte kasvab. Samuti eeldaks see vähemalt ühte määratud töötajat igas Helmese kontoris, kes konkreetselt antud infot uuendama peaks ning antud töötaja vahetumisel põgusat koolitamist uuele töötajale, et süsteemi tutvustada.

4.3.1 Andmete salvestamine CIR süsteemi

Esimese variandina on võimalik töötajate ajalugu salvestada CIR süsteemi uude tabelisse. Töötajate tiimi kuuluvust tuleks uuendada perioodiliselt, pöördudes Identity API poole ning salvestades muudatused töötajate meeskondades. Andmeid tuleks uuendada iga kord, kui töötaja kohta saadud meeskond Identity API kaudu ei vasta samale meeskonnale, mis talle on viimasena CIR pool märgitud. Andmeid tuleks uuendada vähemalt kord päevas, et tagada õiged muudatuste kuupäevad. Joonisel 8 on kujutatud protsessi andmete liikumine pealiskaudse näidisena, lisatud on ka olukord, kus kolmas osapool soovib ligi pääseda töötajate ajaloolistele kuuluvustele meeskondades.



Joonis 8. Töötajate ajalooliste andmete hoiustamine CIR andmebaasis.

Antud variandi tugevused on:

- esmane arendus oleks üsna lihtne ning toimuks juba tuttaval platvormil;
- funktsionaalsuse lisamine oleks vähe kulukas;
- CIR süsteemis on juba varasemalt rakendatud protsesse, mis asünkroonselt töötavad, osasid funktsionaalsusest saaks taaskasutada;

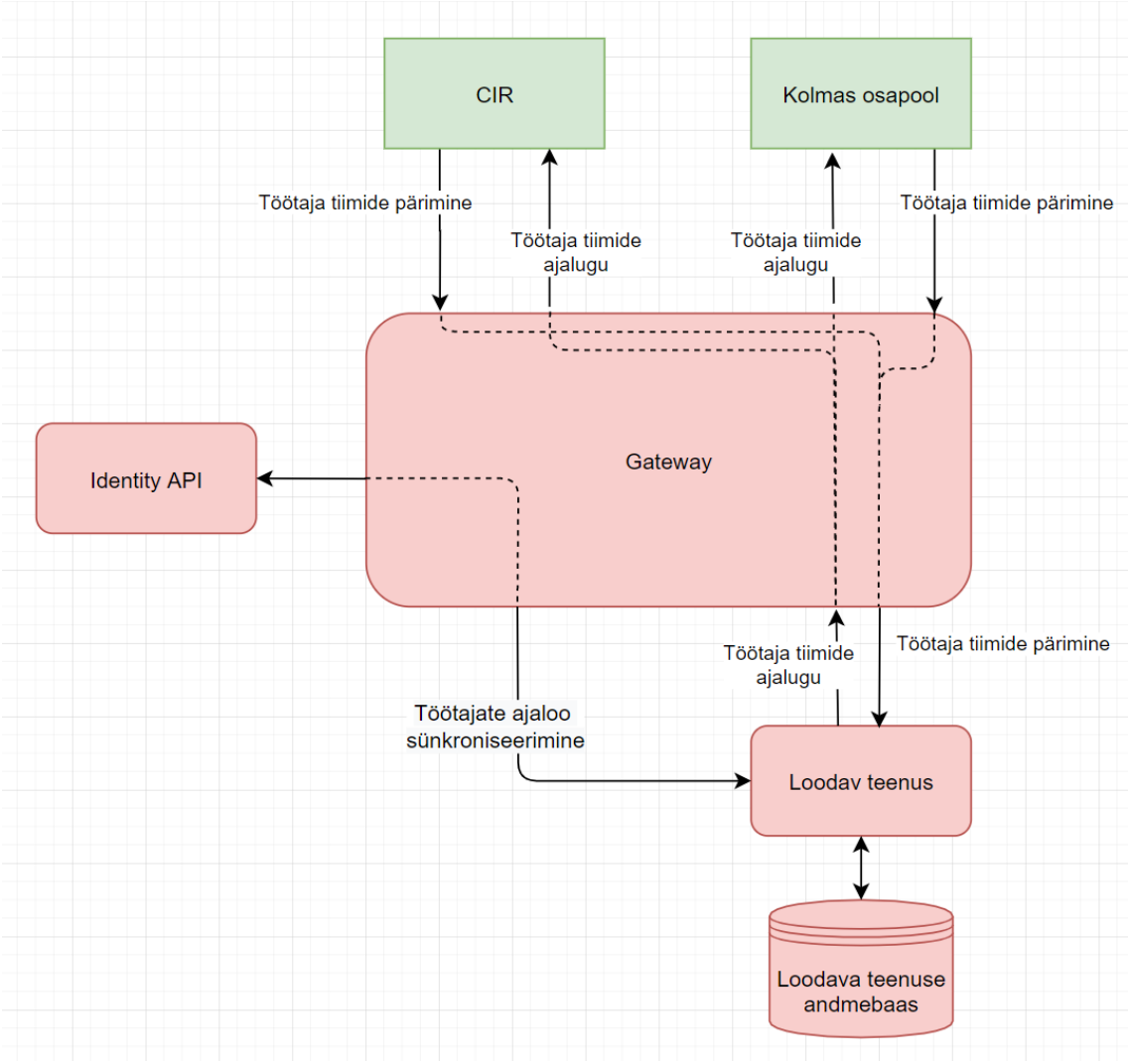
Andmete hoiustamisel CIR süsteemis on samas ka tuntavad puudused:

- ajalooliste töötaja andmete kogumine pole süsteemi äriline funktsioon;
- lahendus on konkreetselt CIR jaoks ning ei skaleeruks tulevikus - kogutud andmestiku vastu on suure tõenäosusega tulevikus huvi ka teistel süsteemidel ja teenustel, CIR poole pöördumine töötaja ajalooliste andmete saamiseks oleks segane tulevaste arenduste jaoks;
- CIR süsteemil puudub täna andmevahetuskiht, mille kaudu andmeid välja anda;
- kuigi esmane arendus oleks odav ja lihtne, oleks kõik tulevased arendused protsessile keerukad ning nõuaks terve süsteemiga töötamist;

Kuigi esmapilgul tundub kõnealuse protsessi lisamine CIR rakendusse kõige lihtsama ja odavama variandina, tuleb arvestada ka antud andmestiku väärtust suuremas pildis. Töötajate ajalooline info ei piirdu suure tõenäosusega vaid nende meeskondadesse kuuluvusega, vaid vastavalt ettevõtte ja pakutavate funktsionaalsuste kasvamisele kasvab ka vajadus töötajate ja üldiste ajalooliste andmete vastu. Sellisel juhul kasvab CIR külge kogumik funktsionaalsusi, mis arhitektuuriliselt süsteemi külge kuuluda ei tohiks ning mis teevad tulevased arenduse keerulisemaks, kasvatades süsteemi mahukaks monoliidiks. Ettevõtte eesmärk on liikuda uute arenduste puhul rohkem mikroteenuste arhitektuuri poole [13].

4.3.2 Andmete salvestamine välisesse teenusesse

Alternatiivse variandina toob töö autor välja uue teenuse loomise, mis oleks keskendunud just ajalooliste andmete kogumisele. Kuigi antud olukorras on kõne all töötajate ajaloolised andmed, tuleb arvestada ka suurema pildiga ning analüüsides ettevõtte ärilisi vajadusi, on tõenäoline, et tulevikus on vajadus erinevate ajalooliste andmete kohta, mida täna kuskil ei hoiustata. Uue teenuse esimene osa oleks töötajate meeskonda kuulumise ajaloo kogumine ning vastavalt päringule andmete tagastamine. Tegemist oleks mikroteenusega, millel on andmevahetuskiht ehk API, mis tagaks struktureeritud ning turvalise andmevahetuse. Sarnaselt eelmisele variandile, pöörduks teenus perioodiliselt Identity API poole ning uuendaks töötajate tiimi kuuluvust juhul, kui töötaja meeskond on muutunud võrreldes eelmiste andmetega. Joonisel 9 on kujutatud esialgne andmete liikumise struktuur olukorras kus CIR ja ka kolmas osapool soovivad ligi pääseda töötajate ajaloolistele andmetele.



Joonis 9. Töötajate ajalooliste andmete hoidmine selleks ette nähtud teenuse andmebaasis.

Uue teenuse loomisel on tugevusteks:

- mikroteenuste arhitektuur, tänu millele on kogutavate andmete formaati tulevikus lihtne muuta ning vajadusel lisaarendusi sooritada;
- uus teenus on selge ärilise eesmärgiga ja läbipaistev, mistõttu on teda lihtne arendajatel kasutada;
- väliste süsteemide arendus, mis nõuavad töötajate ajaloolisi andmeid, on odav ning kiire;
- lahendus skaleerub probleemideta, ei olene ühestki teisest süsteemist välja arvatud Identity API, kust töötajate andmed pärinevad;

Antud lahenduse negatiivseteks külgedeks on samal ajal:

- kallis esmane arendus, nõuab täiesti uut süsteemi;
- uue teenuse sisse toomine tänasesse rakenduste võrku, mis tähendab lisasüsteemi, mida hallata ning millel on vaja tooteomanikku;
- mikroteenustega kaasnevad ohud, näiteks dubleeritud funktsionaalsus erinevatel süsteemidel;

Lahenduse puuduse positiivseid ning negatiivseid külgi kokku võttes jõuab autor järelduseni, et tegemist on funktsionaalsuse poolest puhtama ning arhitektuuriliselt õigema lahendusega mis vastukaaluna toob kaasa kallima arenduse maksumuse.

4.3.3 Variantide võrdlus ning otsus

Variantide võrdlusel põhines autor arenduste maksumusest, variantide skaleeruvusest ning sisesüsteemide arhitektuurilistest tavadest. Arenduse maksumuse leidmiseks lasi autor mõlema lahenduse arenduse töömahud ära hinnata. Hinnangud töömahtudele on umbkaudsed ning omavad väärtust suhtarvudena võrdluses. Tabelis 1 on välja toodud erinevad faktorid mõlema variandi kohta.

Tabel 1. Kahe töötaja meeskonda kuuluvuse ajaloo variandi võrdlus põhinedes olulistele faktoritele.

Atribuut	**Uus teenuse loomine**	**CIR osana kasutamine**
Esialgne arenduse maksumus	Ligi 70 tundi arendust	Ligi 15 tundi arendust
Skaleeruvus	Skaleerub lõputult, ei olene andmeid tarbivatest süsteemidest, 0 tundi	Iga uues süsteemi liidestamine andmebaasile on keerukas arendus, 20 tundi
Mikroteenuste arhitektuur	Jah	Ei

Järgnevate arenduste maksumus (lisaks töötajate tiimide ajaloole)	Iga uue arenduse implementeerimine on odav ning iseseisvalt rakendatav	Uued arendused on mahukad ning nõuavad arendust tervele rakendusele

Võrdlusest tuleb välja, et välja arvatud esialgse arenduse maksumus, mis on CIR arenduse puhul ligi 80% odavam, soosivad ülejäänud otsustavad faktorid uue teenuse arendamist. Võttes arvesse, et iga uue süsteemi lisaarenduse maksumus on ligi 20 arenduse tundi, tasub uue teenuse loomine rahaliselt ära ennast olukorras, kus vähemalt 3 rakendust vajavad ligipääsu töötajate ajaloo andmestikule.

Lisaks oleks arenduse loomine CIR rakenduses vastuolus Helmese sisesüsteemide arhitektuuriga ning liiguks vastassuunas majasiseselt implenteeritava mikroteenuste arhitektuuriga. Tekiks olukord, kus aine rohkem süsteeme on konkreetselt CIR süsteemist ja selle andmebaasist sõltuvad ning üks eraldiseisev rakendus saaks liialt suure ärilise vastutuse. Eraldi seisvat teenust on lihtne arendada ning kasutada arendajatel [13].

Teenuse vastu räägib asjaolu, et uut eraldiseisvat teenust peab ülal hoidma ning haldama aktiivselt, see nõuab lisa tähelepanu võrreldes CIR süsteemiga, millel on juba tooteomanik olemas. Samuti tuleb silmas pidada, et liialt suurte süsteemide koguse juures kannatab tavaliselt süsteemide ning andmete kvaliteet olukorras, kus neid ei administreerita [13][7].

Võttes arvesse eelpool mainitud punkte, otsustas töö autor, et antud olukorras on vajalik luua uus teenus, mis kogub perioodiliselt töötajate meeskondadesse kuuluvuse ajalugu ning hoiab neid kindlalt formaadil. Investeerides arendusse rohkem ressurssi, tagab Helmes enda jaoks töökindla ning kergelt skaleeruva teenuse, mis annab väärtust erinevates ärilistes protsessides. Teenus tagastab läbi API kihi andmeid neid pärivatele süsteemidele ning on üheks arusaadavaks andmeallikaks Helmese siseste ajalooliste andmete jaoks.

4.4 Ajalogide liikumine süsteemide vahel

Suur osa probleemidest, mida vastajad nimetasid, olid seotud vigades ajalogide andmetes. Täna kogub CIR ajalogisi otse Jira andmebaasist läbi keeruka ning mahuka SQL päringu. Selle tõttu on esiteks raske leida tekkivate vigade põhjuseid ning neid parandada, ilma et vaja oleks suuremat arendust CIR poole peal. Samuti piirab see uute ajalogimise rakenduste lisamist dünaamilise crossbilli loomise protsessi, uue süsteemi lisamiseks või muutuste korral vanas süsteemis, on vaja CIR poolel teha mahukat arendust ning iga arendus on konkreetse ajalogimise rakenduse põhine, tehes lahenduste taaskasutamise keeruliseks või võimatuks. Autor analüüsib võimaliku alternatiivina tänasele ajalogide pärimisele uue teenuse loomist, mille eesmärk oleks ajalogisi vahendada lõppkasutaja ja erinevate ajalogimise rakenduste vahel.

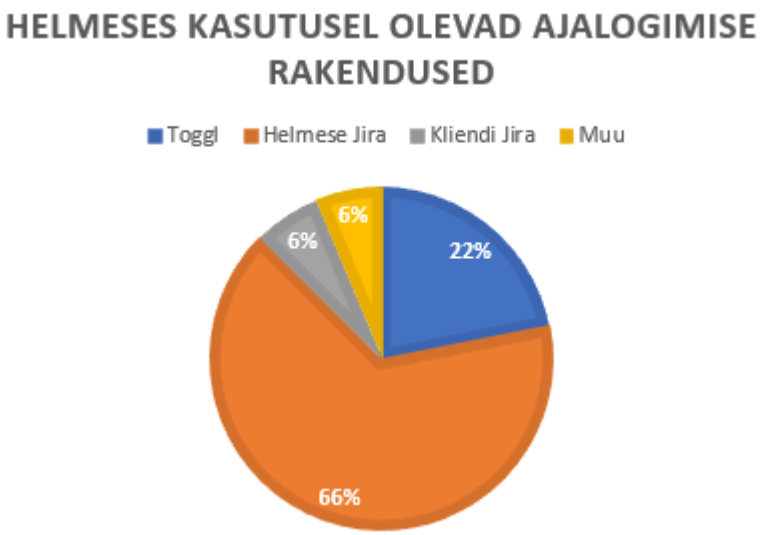
4.4.1 Ajalogide vahendamise teenuse loomine

Mikroteenuste eelised ettevõttes on välja toodud peatükis 2.3, sama kohandub ka siin. Lisaks aitaks uue teenuse arendamine terve dünaamilise crossbilli tekkimise protsessi algusest peale läbi vaadata ning seeläbi lahendada tänased probleemid ja luua võimalusi uuteks lahendusteks. Mikroteenuse struktuur on antud olukorras aktuaalne seetõttu, et ajalogisid pakkuvad rakendused võivad pidevalt muutuda, mis ei tohiks andmeid tarbivaid süsteeme mõjutada. Samuti oleks õige ajalogide kogumise funktsionaalsus tõsta CIR süsteemist välja, sest tegemist on andmetega, mida tulevikus kasutavad ka teised rakendused. CIR kaudu nad andmetele täna ligi ei pääseks.

4.4.2 Ajalogimise liidestamiste võrdlus

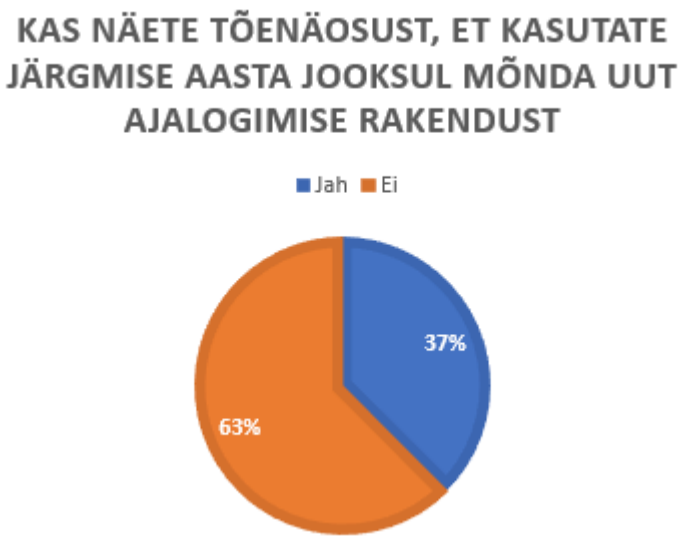
Kuigi mikroteenuste arhitektuur on ennast tõestanud vajalikuna suurtes ettevõtetes kust andmetel on palju tarbijaid, ei pruugi see ennast Helmese kontekstis antud näitel õigustada, juhul kui ajalogide ainsaks tarbijaks jääb CIR ning ainsaks ajalogimise rakenduseks jääb Jira. Sellisel juhul on kuluka mikroteenuse arendamisel vähe ärilist mõju ning arendus ei pruugi ennast ära tasuda. Autor viis olukorra analüüsimiseks läbi lisa intervjuud kuue tiimijuhiga, et tuvastada vajadus töötajate ajalogide järele ka teistes kontekstides ning korraldas küsitluse kõikide Helmese tiimijuhtide seas, et tuvastada, milliseid ajalogimise rakendusi kõige rohkem kasutatakse ning millised võivad olla muutused selles vallas tulevikus. Küsimustikule vastas 32 tiimijuhti ning joonisel 10 on

nende märgitud variandid. Kõik vastusevariandid, millel oli kuni 1 hääl, koondas töö autor vastuse ''muu'' alla.



Joonis 10. Sektordiagramm töötajate poolt kasutusel olevate ajalogimise rakenduse osakaalu kohta.

Küsitlusel on eristatud Helmese Jira ning kliendi Jira – kliendi Jira ei asu Helmese infrastruktuuris ning selle andmetele turvalist ligipääsu ei ole, seetõttu jäeti see variant esmasest analüüsist välja. Vastajate seast selgub, et peale Jira on suur kasutajaskond veel Toggl rakendusel. Lisaks täna kasutusel olevatele rakendustele, küsiti ka vastajate arvamust, kas nad näevad ennast aasta jooksul kasutamas mõnda ajalogimise rakendust, mida täna vastusevariantides ei olnud. Autor ei pidanud vajalikuks hetkel täpsustada kaalutavaid rakendusi, mis näeks ette eraldi analüüsi iga liidestatava rakenduse jaoks. Joonisel 11 on vastajate vastused, kas nad näevad ennast kasutamas muid ajalogimise rakendusi.



Joonis 11. Sektordiagramm töötajate arvamuse kohta, kas aasta jooksul tuleb kasutusele uusi ajalogimise rakendusi.

Vaadates küsitluse vastuseid on selge, et Helmeses on laialdaselt kasutusel rohkem ajalogimise rakendusi kui Jira ja tulevikus kasutatavate rakenduste arv kindlasti kasvab. Põhinedes kasvavale huvile erinevate rakenduste vastu, on vajalik tekitada võimalus uusi ajalogimise rakendusi hõlpsalt dünaamilise crossbilli loogikasse lisada. Antud küsimustiku põhjal on selleks selge vajadus olemas.

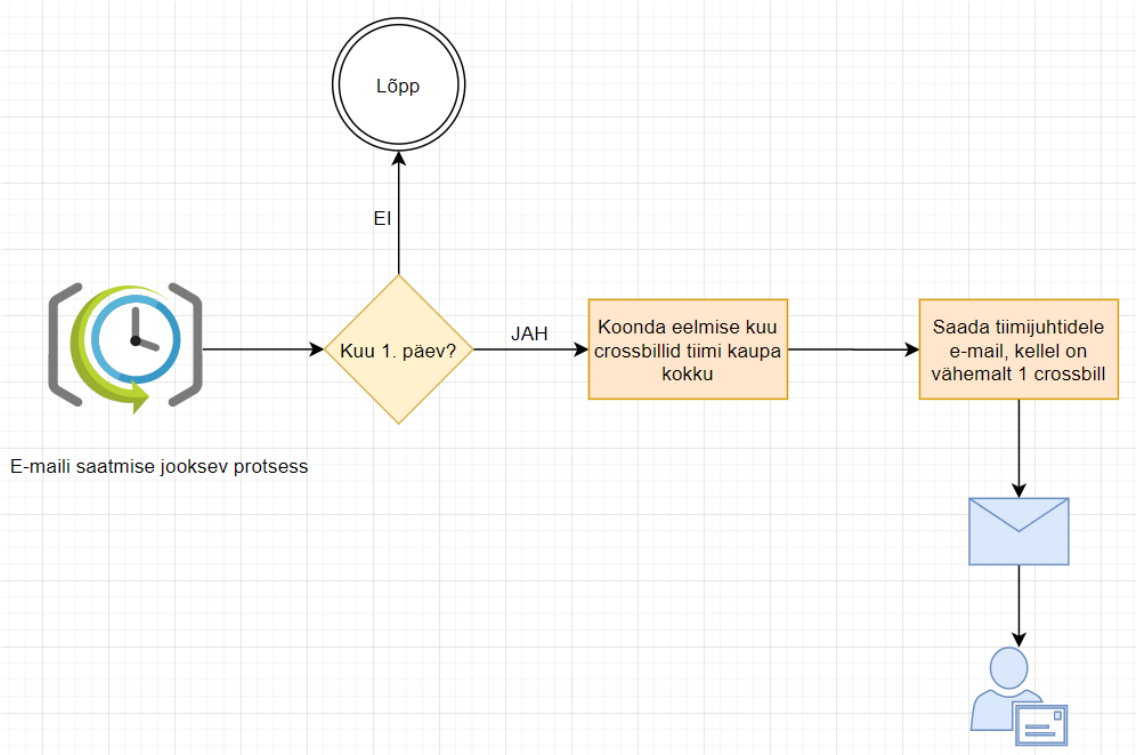
Intervjuude käigus kuue vastajaga, selgus et tugev huvi ajalogide andmete vastu on igal vastajal ning andmestik annaks väärtust ka süsteemidele peale CIR. Näitena toodi üles tulude arvestamist töötaja kohta, töötaja väljamüüki ja palju muud.

Põhinedes Helmese töötajate vastustele ning huvile ja sisesüsteemide arhitektuurilistele juhistele, on vajadus eraldi ajalogide teenuse järele olemas. Vastuargumendiks saaks lugeda, et vajadus antud teenuse järele ei ole aktuaalne veel tänases olukorras, vaid tulevikus, kui rohkem ajalogimise rakendusi on juba kasutusel. Sellegipoolest on sisesüsteemide roll luua võimalus liidestada erinevaid rakendusi enne kui nad on kasutusel, et vältida olukorda, kus väline rakendus jääb kasutamata sisesüsteemide puuduste tõttu.

4.5 Meeldetuletuste lisamine dünaamilise crossbilli protsessile

Eraldi murena tõid vastajad välja andmete kontrollimise unustamist olukorras, kus neile on crossbill loodud. Töö autor pakub probleemi lahendamiseks välja meeldetuletuse

lisamise, mis saadab meeskonnajuhtidele iga kuu alguses e-kirja, kus on kirjas kõik nende meeskonnaga seotud crossbillid, jagatud kaheks – sissetulek ja väljaminek. Sellises olukorras oleks tiimijuhil võimalik põgusalt andmed üle vaadata ilma vajaduseta CIR süsteemi minna ning arusaamatuse korral. Rakenduse CIR protsessi, mis perioodiliselt e-maile välja saadab, lisatakse uus teavitus, mis käivitatakse iga kuu esimesel päeval. Teavituses on 2 tabelit – ostetavad crossbillid ja müüvad crossbillid. Joonisel 12 on protsess kõrgel tasemel välja joonistatud.



Joonis 12. Crossbillide ülevaatamise teavituse saatmise tulevane protsess.

5 Töötajate ajaloo mikroteenuse analüüs

Antud peatükk kirjeldab detailset analüüsi uue mikroteenuse loomiseks, mis salvestaks perioodilist töötajate tiimi kuuluvust ning võimaldaks läbi API tagastada töötaja kuuluvust kindlasse tiimi mingil perioodil. Antud teenus nimetatakse esialgu nimega *History API*. Mikroteenuse nimetusel on tähtis roll selle teenuse elu jooksul, nimi peab olema võimalikult lühike, et ta oleks mugav kasutada nii arendajatel kui ka teistel rakendustel – samal ajal peab nimi olema piisavalt sisukas, et anda ülevaade, mis on teenuse peamine äriline eesmärk [20][17]. Antud teenus keskendub ajalooliste andmete kogumisele, töötlemisele ning esitamisele – seega sobib talle ka ingliskeelne nimi, mis otsetõlkes tähendab ''ajalugu''. Seda peetakse meeles ka tulevikus, kui antud teenusele kaalutakse lisaarendusi – kõik funktsionaalsused peavad olema seotud ajalooliste andmete kogumise ja/või esitamisele [17]. Samuti on ettevõtte sisemiste rakenduste eest vastutava isiku ülesanne tagada, et kõik funktsionaalsused, mis on seotud ajalooliste andmete kogumise ja/või esitamisega, tuleks võimalusel ehitada antud teenusele.

5.1 Töötajate ajaloo teenuse komponendid

Mikroteenusele saab liidestatud andmebaas. Baas on esialgu lihtsa struktuuriga ning omab vaid ühte andmetabelit, millesse hakkab teenuse perioodiliselt salvestama töötaja kuuluvust meeskonda ning kuuluvuse perioodi alguse kuupäeva ning lõppkuupäeva. Antud andmete kogumiseks on teenusel perioodiliselt jooksutatav protsess, mis uuendab andmetabeli andmeid vastavalt vajadusele. Viimase komponendina on teenusel HTTP protokolliga API kiht, mille kaudu on välistel rakendustel võimalik teenuselt töötajate tiimi kuuluvuse andmeid pärida.

5.2 Töötajate tiimi kuuluvuse andmebaas

Baasi esmaseks eesmärgiks on iga töötaja kohta määrata kindlaks, mis tiimi see töötaja on ajalooliselt kuulunud. Seega tabelile on minimaalse eesmärgi täitmiseks vaja viite veergu:

1. id – tabeli kande unikaalne nimetaja, tabeli *primary key*;

2. töötaja guid – töötaja unikaalne guid formaadis nimetaja, näitab millise töötaja kohta antud kirje käib;

3. meeskonna kood – millise meeskonna kohta antud kirje käib;

4. töötamise alguskuupäev – näitab, alates mis kuupäevast töötaja antud tiimi on kuulunud;

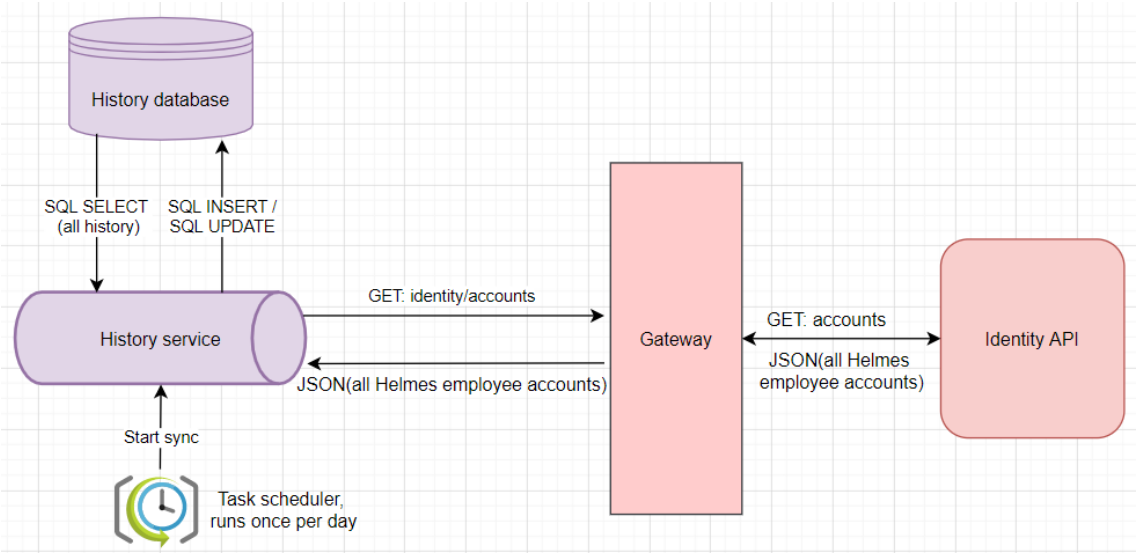
5. töötamise lõppkuupäev – näitab, kuni mis kuupäevani töötaja antud tiimi on kuulunud;

Ainuke unikaalne nimetaja tabelis on *id*, ehk iga töötaja ja tiimi kohta saab olla 0...* kirjet, piirangut kirjete arvule ei ole. See tähendab, et töötaja saab olla olnud mitmes meeskonnas eri aegadel ning ühes meeskonnas saab olla piiramatu arv töötajaid. Ühel töötajal ei saa olla korraga mitu kirjet, millel puudub töötamise lõppkuupäev ehk ei saa olla mitmes meeskonnas korraga. Andmestiku koostamisel peetakse silmas *Privacy by design* metoodikat, mille tõttu ei ole andmebaasi põhjal võimalik andmeid ühegi konkreetset töötajaga kokku viia, ilma tema guid koodi teadmata [21]. Antud andmetabeli struktuur on minimalistlik kuid piisav soovitud ärilise funktsionaalsuse saavutamiseks. Kasutades tööriista pgAdmin 4, tegi töö autor näidisandmebaasi töötajate andmete jaoks, kasutades selleks postgreSQL andmebaasi keelt. Töö lisana (Lisa 1 – Töötaja meeskondade ajaloo demo) on kujutaud näidisandmebaas koos kahe kirjega – *id 1* kirjeldab kuuluvust eelmisesse meeskonda *OLDTEAM* ning kuulumise perioodi ning *id 2* kirjeldab kuuluvust tänasesse meeskonda *NEWTEAM*.

5.3 Töötajate tiimi kuuluvuse andmete uuendamine

Selleks, et töötajate tiimi kuuluvuse andmeid uuendada, peab loodud teenus pöörduma varem loodud teenuse Identity API vastu, et sealt vastusena saada kõiki Helmese töötajate andmed. Identity API tagastab andmeid, mis paiknevad Active Directory kasutajahalduse programmis – API tagastab andmeid pärimise hetke seisuga, andmete ajalugu ei salvestata. Protsessi jaoks vajalikud andmed on töötaja unikaalne nimeta ehk guid formaadis identifikaator, töötaja meeskond antud hetkel ja märge, kas töötaja kasutaja on aktiivne või mitte. Joonisel 13 on illustreeritud sünkroniseerimise struktuur kõrgel tasemel – kord päevas alustab automaatne protsess andmete sünkroniseerimise töötajate ajaloo teenuses, peale mida pöördub teenus läbi Helmes'e gateway Identity API poole

päringuga kõigile töötajate andmetele. Suhtlus teenuste vahel toimub REST raamistikus ning andmete formaadiks on Json andmevahetusvorming [3]. Alloleva protsessdiagrammi komponendid on nimetatud inglise keeles, sest antud diagramm on kasutusel ka arendusprotsessis, millel on nõue ingliskeelsele dokumentatsioonile.



Joonis 13. Töötajate meeskondadesse kuuluvus ajaloo sünkroniseerimise protsess.

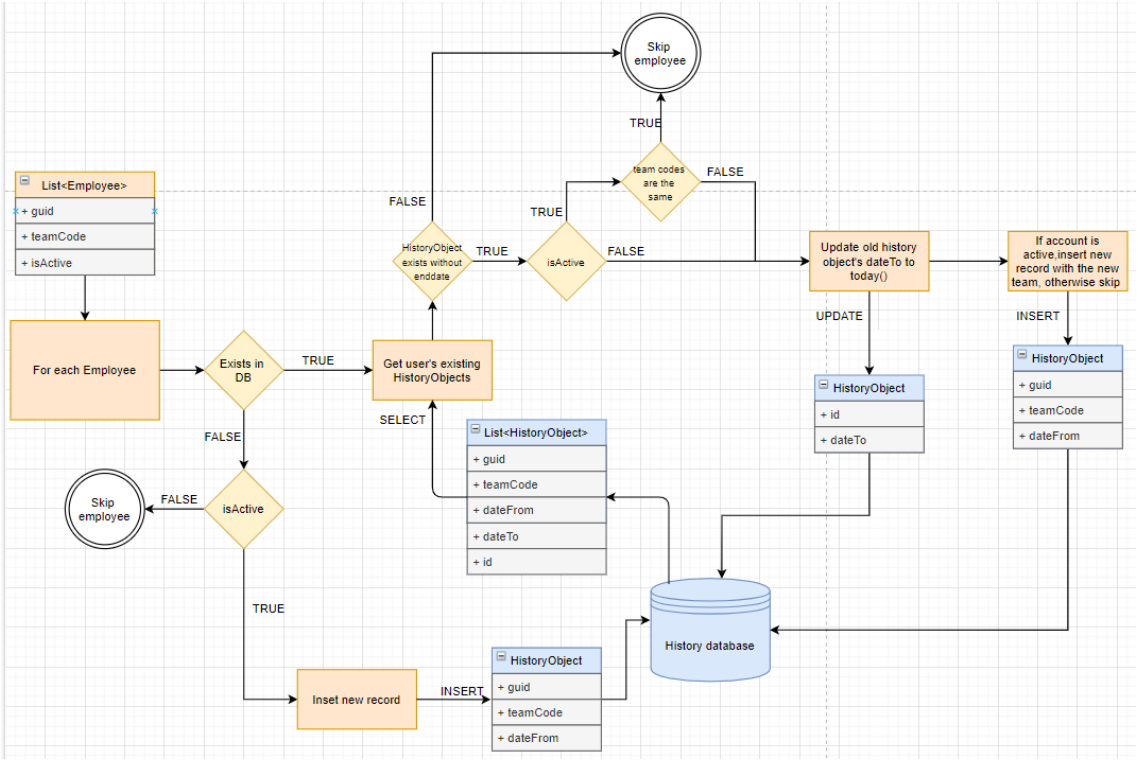
Saadud vastusest jagab teenus töötajad kaheks – töötajad, kes on juba andmebaasis olemas ning töötajad, keda veel ei eksisteeri. Iga töötaja kohta, keda andmebaasis ei ole ja kelle kasutaja on aktiivne, luuakse uus kirje ning määratakse kirjele:

1. töötaja guid;
2. meeskonna kood;
3. töötamise alguskuupäev – kuupäev, millal sünkroniseerimine toimus.

Tähendab, kui töötajat juba meeskondade ajaloo andmestikus ei olnud, liitus ta samal päeval, millal sünkroon toimus.

Kui töötaja juba eksisteeris andmebaasis, toimub kõigepealt kontroll, kas töötaja kohta on mõni kirje, millel puudub töötamise lõpetamise kuupäev. Kui selline kirje puudub, siis antud töötaja jäetakse vahele ning edasisi toiminguid ei toime – töötaja on nii-öelda ’lukus’ ehk *de facto* ettevõttest lahkunud, kuid tema kasutajat hoitakse veel alles. Kui aga leidub kirje, millel lõppkuupäev puudub, kontrollitakse kas töötaja praegune meeskond on sama mis antud kirjes, kui see nii ei ole, uuendatakse eelmine kirje määrates talle

töötamise lõpetamise kuupäev(eelmises meeskonnas) ning tehakse tabelisse uus kirje uue meeskonna kohta, alustuskuupäevaks päev, millal käesolev sünkroniseerimine aset leiab. Joonisel 14 on kuvatud protsessdiagramm antud protsessi kohta, mille sisendiks on loend kõikide töötajatega, mis Identity API teenusest saadud on ning protsess viiakse läbi iga töötaja kohta eraldi. Protsessdiagramm on joonistatud *flowchart* metoodikat kasutades [22].



Joonis 14. Protsessdiagramm ühe töötaja meeskonda kuuluvuse uuendamise protsessi kohta [22].

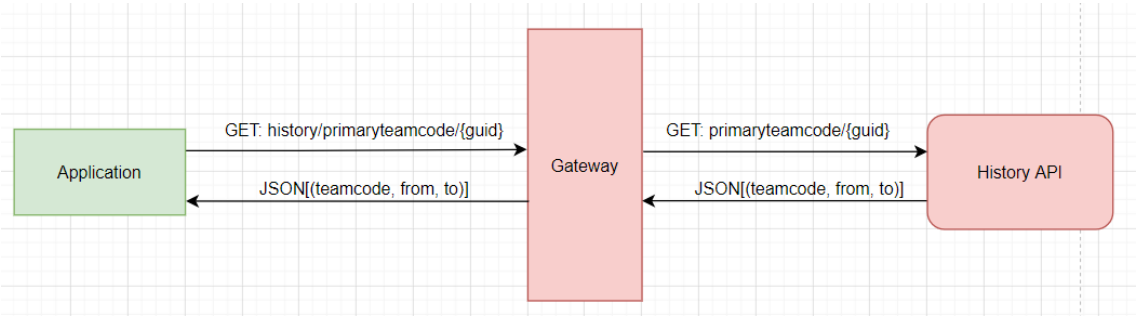
5.4 Töötajate ajaloo teenuse API kiht

Mikroteenuste arhitektuuri järgides jagunevad rakenduse eri osad kihtideks, jagunedes vastavalt funktsionaalsusele. Igal rakendusel, mis jagab andmeid, on vajalik andmevahetuskiht, mille kaudu saavad teised rakendused või kasutajad pärida antud rakenduses hoitavaid andmeid [13]. Andmevahetuskihi loomisel lähtuti ''*pragmatic REST*'' [20] lähenemisest, ehk API loodi võimalikult lihtsalt kasutatavaks arendajate poolt. Antud lähenemine on vajalik olukorras, kus paljud erinevad arendajad võivad läbi viia teenuse liidestamist erinevate süsteemide külge, mille puhul on vajalik, et API teenus oleks kiirelt omandatav ning selle kaudu saadavate andmete pärimine oleks võimalikult sujuv protsess. Antud teenuse jaoks sai valitud REST, eelistatud oma JSON

andmeformaadi ning kiire ja lihtsa implementeerimise poolest. Samuti on Helmese arendajate seas kõige rohkem just seda kompetentsi (võrreldes alternatiividega SOAP ja GraphQL).

Uue rakenduse arendamisel Helmese sisemiste rakenduste võrgustikku, on esmane prioriteet paika panna loodava süsteemi õiguste struktuur. Antud juhul on tegemist mikroteenusega, millel puudub kasutajaliides, seega toimub kogu suhtlus teenuse ning väliste andmepärijate vahel läbi Helmes'e sisese *gateway*. See tähendab, et kõik päringud teenuse pihta käivad läbi vaherakenduse *gateway*, mis vahendab päringu edasi rakendusele endale. History API jaoks lisatakse uus *Active Directory* roll *Role – API GW – History*, jälgides varem kehtivat nime struktuuri sarnastele rollidele. Esialgu saab antud rolli endale külge vaid rakenduse CIR teenuskasutaja – tänu õiguste liikumisele läbi rolligrupi, on tulevikus lihtne ligipääsu teenusele jagada ka teistele rakendustele. Lisaks annab see ülevaate, kellel parasjagu on õigus rakendusele andmetele ligi pääseda.

Arendatavale teenusele luuakse üks päringu aadress, millel on üks kohustuslik sisendparameeter. Sisendparameetriks on otsitava kasutaja unikaalne guid kood ning vastusena tagastab päring kõik tabeli kirjed selle guid koodi kohta. Mitu kirjet tagastatakse nimistuna, nimistus võib olla 0...* kirjet, ehk mitte ühegi kirje puhul tagastatakse tühi nimistu. Joonisel 15 on kujutatud andmeliikumise protsess välise rakenduse ning History API vahel.



Joonis 15. Andme liikus loodava History API teenuse ning andmeid päriva rakenduse vahel.

Päringu sisendparameetrile ühtegi kontrolli ei tehta ning see on formaadis *string*, mis ei piira teksti sisu. See tähendab, et päring tagastab eduka vastuse ka juhul, kui sisendparameetrile ei vasta andmetabelis ühtegi kirjet – antud juhul tagastatakse tühi nimistu ning päringu staatuse kood on endiselt edukas.

6 Ajalogide vahendamise mikroteenuse analüüs

Antud peatükk kirjeldab uue loodava teenuse detailset analüüsi, mille eesmärk on vahendada erinevatelt ajalogimise rakendustelt logitud aega ning koondada need kindlale ühisele struktuurile, tagastades rakenduse lõpptarbijale ajalogimise rakenduse eripäradest sõltumatult ühise struktuuri ja vormiga andmed. Rakenduse nimetuseks saab *Timelog API* ning selle ainsaks äriliseks funktsiooniks jääb ajalogide vahendamine. Aplikatsiooni nimi kirjeldabki võimalikult lühikeselt selle ärilist funktsiooni ning mida see erinevatele kasutajatele pakub. Teenuse arendamisel peetakse esmajoones silmas, et see oleks võimeline skaleeruma olukordades, kus tuleb liidestada uusi ajalogimise rakendusi ning ei ole sõltuv mitte ühestki kindlast välisest rakendusest. Põhiline äriline kasu loodaval teenusel on pakkuda süsteemidele ligipääsu töötajate ajalogidele ilma, et päringut tegev süsteem peaks teadma, millise ajalogimise rakenduse poole eraldi peab pöörduma. Selliselt pole üksi ajalogidele ligipääsu nõudev süsteem otseselt sõltuv ühestki ajalogimise rakendusest – Helmeses on kasutuses vaid teenusena sisse ostetud ajalogimise rakendused, mille puhul on tähtis vältida liigset sõltumatust ühest kindlast rakendusest.

6.1 Ajalogide vahendamise teenuse komponendid

Teenus tegeleb andmete vahendamise ning konsolideerimisega, seega antud hetkel ei ole vajalik arendada eraldi andmebaasi. Põhiliseks komponendiks on API kiht, läbi mille tulev sisend alustab kõiki protsesse teenuses, eraldi perioodiliselt jooksvaid protsesse antud olukorras vaja ei ole. API kiht annab signaali vastavaks toiminguks edasi rakenduse *backend* poolele, mis pöördub vastavate ajalogimise rakenduste poole ning seejärel tagastab õiges formaadis vastuse API kihiline, mis õiges formaadis vastuse päringu sooritanud kasutajale tagastab.

6.2 Ajalogide vahendamise teenuse API kiht

Kõik päringud teenusele esitatakse läbi API kihi, mis saadab päringu edasi vastavasse *backend* funktsionaalsusesse, olenevalt päringust. API kihi arendusel peetakse kinni samadest põhitõdedest ja arusaamadest, mis on kirjeldatud peatükis 4.4 töötajate ajaloo

teenuse API kihi kohta. Sarnaselt eelnevalt kirjeldatud teenusele, sobib samadel põhjustel ka siin REST rakendamine.

Ligipääsu haldamiseks *Timelog API'*le, lisatakse *Active Directory* kataloogi uus rolligrupp nimega *Role – API GW – Timelog.* Ainukese rakendusena saab esialgu rolli külge CIR teenuskasutaja.

Planeerides võimalikke päringu variante API teenusele, lähtub auto varem mainitud ''*pragmatic REST''* lähenemisest ning analüüsib, millisel kujul ning milliseid andmeid rakenduse lõppkasutajad vajavad [20]. Selle järgi saab otsustada ka millisel kujul andmeid kogutakse ning mis on kohustuslikud väljad. Ajalogide vastu on suur potentsiaalne huvi mitmel erineval süsteemil, kuid esialgu lähtub autor konkreetsest crossbill funktsionaalsusest ning minimaalsetest andmetest, mis ühe crossbilli loomiseks teenusest vaja on:

1. töötaja guid, et tuvastada, kes on konsultant ning logis oma töötunnid;
2. projektikood, et tuvastada millisesse projekti tööd tehti ning millest on tuletatav, milline meeskond tehtud tööde kulud katab;
3. periood, et tuvastada millal antud töö on tehtud;
4. logitud aeg, et arvutada tehtud töö summa;
5. ajalogide allikas, et kindlaks teha, millisest ajalogimise rakendusest andmed pärit on.

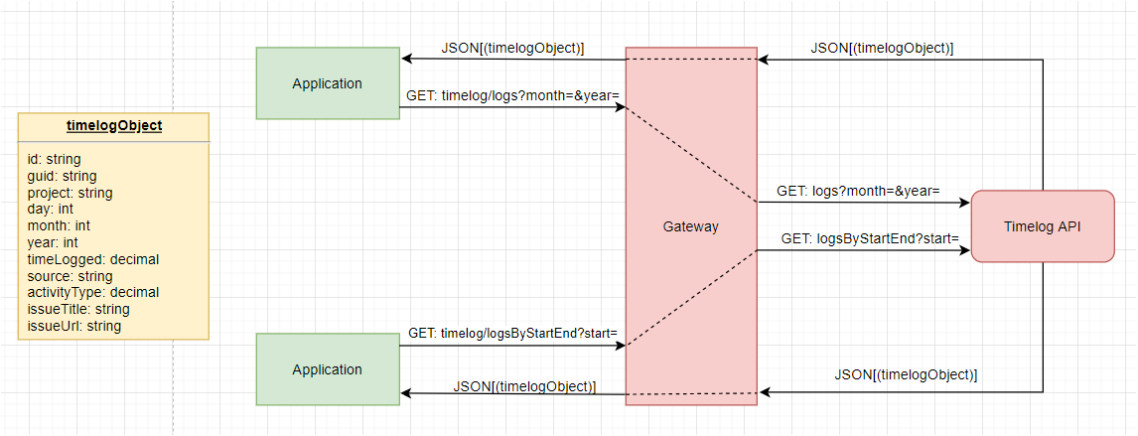
Lisaks minimaalsetele väljadele võeti arvesse ka väljasid, mis annavad crossbill funktsionaalsusele lisandväärtus:

1. töö tegemise tüüp – kas tegemist on tava tööajaga või ületundidega;
2. pileti nimetus, kuhu aega logiti – annab kasutajatele võimalusi nähe, kuhu töösse antud aeg kulus
3. pileti URL aadress – annab võimaluse tekitada kasutajatele hüpik aadress, kuhu vajutades on võimalik kiiresti navigeerida piletini, kuhu aega logiti.

Lähtudes asjaolust, et huvi ajalogide vastu on suure tõenäosusega tulevikus ka teistel süsteemidel, on vajalik, et kasutajatel oleks võimalik andmeid pärida erinvate filtrite ehk valikuliste sisendparameetritega. Esialgu luuakse teenusele kaks erinevat päringut:

1. ajalogide pärimine kindla kuu ning aasta kohta;
2. ajalogide pärimine määratud ajavahemikul.

Mõlema päringu vastus on samas formaadis, erinevus seisneb kohustuslikust sisendparameetrist. Antud olukorras on oht, et kahte teenust vaja ei lähe ning YAGNI [23] põhimõtte kohaselt tuleks esmalt arendada vaid päring, mida on teada et vaja läheb – kõik ülejäänud 'igaks juhuks' tehtud arendused on tavaliselt kas ebavajalikud või arendatakse need valedel alustel. Võttes arvesse eelpool mainitut, otsustas töö autor vastavalt arendusressursi jaotamise efektiivsusele töösse võtta siiski mõlemad päringud. Esimesel variandil on kaks kohustuslikku sisendparameetrit – kuu ning aasta. Ilma lisa parameetriteta tagastab päring kõik antud kuu ajalogid. Teise päringu kohustuslik sisendparameeter on perioodi algusaeg, perioodi lõppaeg on valikuline ehk antud päringuga on võimalik pärida andmeid pikema perioodi kohta kui üks kuu. Teine päring on kasutamiseks vaid erandlikel juhtudel ning vastavalt vajadusele – crossbill funktsionaalsus küsib ajalogisi kindla kuu kohta, sellisel viisil on ka pärimine arusaadavam ning lihtsam. Võimalusel on soovitatav esimese päringu kasutamine seetõttu, et pärija ei saaks tekitada liialt mahukat päringut – ajalogisi, mida päritakse on koguselt palju ning ilma perioodi määramata on võimalik, et päringu maht läheb liialt pikaks ning vastuse ootamine API teenuselt on liialt ajakulukas. Mõlemal päringu on lisaks kohustuslikele parameetritele ka kaks valikulist parameetrit – töötaja guid kood ning projektikood. Antud väärtused aitavad vastuseid filtreerida vastavalt kasutusvajadusele. Joonisel 16 on kujutatud kahe võimaliku päringu teostamine, päringute kohustuslikul väljad on joonisel ära märgitud. Samuti illustreerib joonis, et olenemata päringu aadressist, on vastuse formaat samasugune – nimistu ajalogidest.



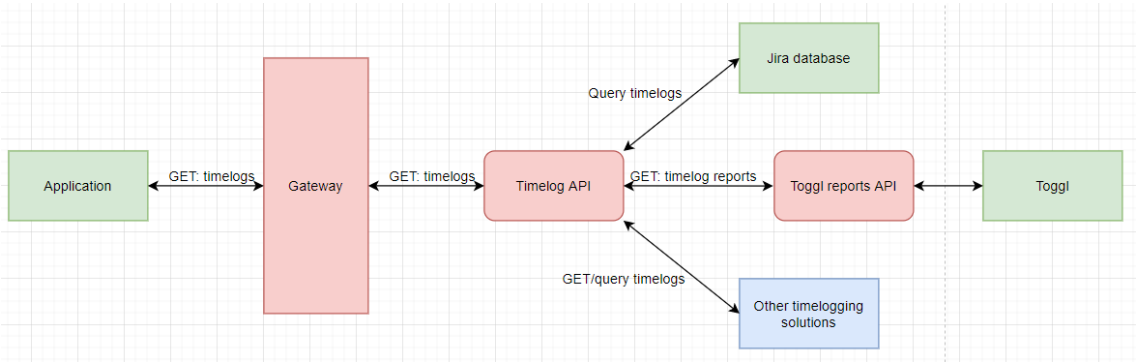
Joonis 16. Töötaja ajalogide pärimine kahel erineval viisil loodava Timelog API teenuse käest.

Iga ajalogi kujutab endast ühte korda, millal kasutaja ehk konsultant enda logitud aja registreeris. Üks tehtud töö võib sisaldada piiramatul arvul ajalogisi. Joonisel mainitud väärtused on alati igal ajalogil olemas olenemata rakendusest, kust ajalogid tulid. Rakenduste puhul, millel mõnda väärtust ei ole, tagastatakse antud väärtused tühjana.

Päringute vastuste formaadis kasutatakse võimalikult vähe piiravaid andmeformaate – ei ole teada, mis formaadis erinevad ajalogimise rakendused enda andmeid hoiavad, seega on ennetavalt välditud *int* formaadi kasutamist kus võimalik, ning selle asemel on kasutusel *string* formaat, mis on tunduvalt paindlikum ning ei piira sisendit.

6.3 Ajalogide vahendamise teenuse *backend*

Peale sisendi saamist API kihilt, pöördub *Timelog API* kõikide liidestatud ajalogimise teenuste poole ning pärib igalt süsteemilt eraldi ajalogid vastavalt API kihi kaudu saadud sisendparameetrile. Antud teenusega saab väliseid ajalogimise teenuseid liidestada kahel eri viisil – eelistatum on liidestada arendatav mikroteenus välise teenuse API kihiga, mille puhul suhtleksid omavahel kahe rakenduse andmevahetuskihid. Sellega oleks kinnitatud andmete vahetamine kindlaksmääratud ning kinnitatud struktuuril ning API kaudu uue ajalogimise lahenduse liidestamine oleks arenduse poolest vähe mahukas ülesanne. See on aga kasutatav ainult juhul kui välised rakendusel nimetatud API lahendus eksisteerib – vastasel juhul pöördutakse rakenduse andmebaasi poole ning päritakse andmeid sealt otse. Joonisel 17 on kujutatud kõrgel tasemel TO-BE protsess olukorras, kus rakendus pärib ajalogisi.



Joonis 17. Joonis süsteemide vahelise suhtluse illustreerimiseks käsitledes töötaja ajalogisi.

Esialgu liidestatakse *Timelog API* Jiraga ning luuakse sobiv keskkond, et liita teisi rakendusi. Järgmine liidetava rakendus peaks olema väline rakendus Toggl, millel on antud eesmärgile arendatud ka vastav API kiht, kust on lihtne ja turvaline andmeid pärida töötajate ajalogide kohta. Antud kihi nimetus on Reports API.

6.3.1 Liidestamine Helmes Jira andmebaasiga

Arenduse planeerimise hetkel ei olnud Jira pool töötavat ning lihtsasti kasutatavat API lahendust töötajate ajalogidele ligi pääsemiseks, selle tõttu liidestatakse loodav teenuse otse Jira andmebaasiga. Jira andmebaas asub Helmese sisevõrgus ning seda majutab sisemise IT meeskond, seetõttu on otse andmebaasiga liidestus ka kõne all, sest andmevahetuse turvalisuse tagab ettevõtte sisevõrk. Pärides andmeid läbi SQL päringu välisest andmebaasist oleks tunduvalt rohkem turvariske ning sellisel juhul tuleks kaaluda teisi variante andmete kättesaamiseks.

Vastavalt sisse tulnud päringule, koostab mikroteenus SQL päringu, mille ta esitab Jira andmebaasis asuvale ajalogide tabelile. Kokku on 4 erinevat piirangut, mis olemasolu korral liidetakse üksteisele AND operatsiooniga. Ajalogi kuupäev on päringus alati olemas ning minimaalselt peab olema märgitud alates mis kuupäevast ajalogisi arvestatakse:

1. kohustuslik - perioodi algus kuupäev;
2. valikuline - perioodi lõppkuupäev;
3. valikuline - töötaja guid kood;
4. valikuline - projektikood.

Päringu vastusena tekib nimistu kõikidest ajalogidest mis antud kriteeriumite kombinatsioonile vastavad. Iga ajalogi kohta tuleb pikk kirje erinevate andmetega, *Timelog API* filtreerib vastusest välja väljad *id, guid, date, project, time_logged, activity_type, issue_id*. Allolevas tabelis ’tabel 2’ on kirjeldatud väljade nimetuste seose tagastatavate väljadega.

Tabel 2. Väljade kaardistamine Jira andmebaasi ning Timelog API päringu vastuse vahel.

Välja nimi Jira andmebaasis	Välja nimetus *Timelog API* päringu vastuses	Välja sisu
Id	Id	Ajalogi unikaalne nimetaja, tagastatakse samal kujul.
Guid	guid	Töötaja unikaalne nimetaja, tagastatakse samal kujul nagu see on Jira andmebaasis
Date	day, month, year	Jira andmebaasis piletil on kuupäev, millal ajalogi alustati. Kuupäevast võetakse välja kolm väärtust ning tagastatakse iga väärtus eraldi.
Project	projectCode	Projektikood, kuhu alla pilet kuulub ehk kuhu aega logiti
Time_logged	timeLogged	Jira andmebaasis millisekunditeks, *Timelog API* konverteerib minutiteks
Activity_type	activityType	Hoitakse Jira andmebaasis *string* formaadis ning on piiranguteta tekst. *Timelog API* konverteerib teksti vastavaks numbriliseks väärtuseks

Issue_id	issueTitle, issueUrl	Tegemist on pileti id'ga, kuhu aeg logiti. Antu id kaudu tehakse päring piletite tabelisse, kust on kätte saadav pileti nimetus.

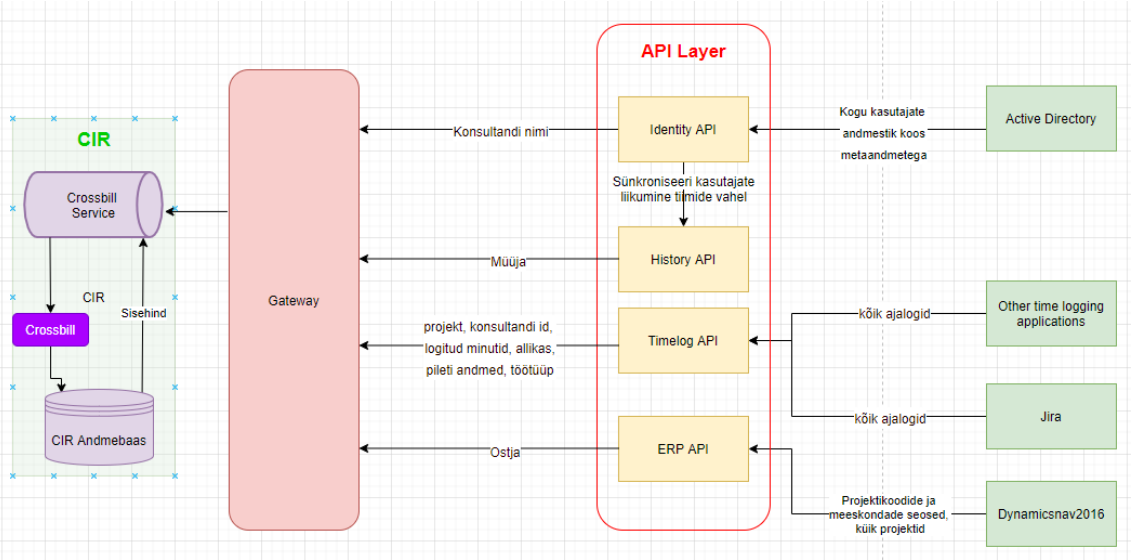
Lisaks ülal välja toodud väärtustele, tagastatakse veel lisaks ajalogimise rakenduse nimetus, antud juhul Jira. Peale Jira poolsete ajalogide leidmist tagastatakse need kõrgemale tasemele, kus need liidetakse teistest süsteemidest saadud ajalogidele.

7 Dünaamilise crossbilli tulevane loomise loogika

Peamised ärilised eesmärgid, mida tulevane protsess peab tagama, on loodavate dünaamiliste crossbillide usaldatavus ning korrektsed andmed, mis vastavad arveldamises nõutavatele standarditele. Samuti peab olema kasutajatele ehk tiimijuhtidele selge, kust ja kuidas on antud crossbill loodud. Muudatuse põhiliseks mõõdikuks on tagasi lükatud dünaamiliste crossbillide arv, mis peaks uute arendustega märgatavalt vähenema. Varasemalt lükati tagasi umbes 50% dünaamiliselt loodud crossbillidest, töö eesmärgiks on parandada seda suhtearvu 15%'ni. Samuti on mõõdikuks läbi Helmese HelpDeski loodud crossbill funktsionaalsuse vigasid teavitavate piletite arv, mis peaks vähenema 3 pileti pealt kuus 1 piletini 3 kuu jooksul peale uue loogika implementeerimist.

7.1 Andmeallikate muutused dünaamilise crossbilli loomisel

Andmeallikatena on tulevases mudelis juures kaks uut süsteemi – *Timelog API* ja *History API*. Eemaldatud on andmeallikana Jira andmebaas, ehk crossbilli luues ei pärita andmeid mitte üheltki rakenduselt otse vaid läbi mikroteenuste API kihtide. Crossbilli loov teenus seega ei pea teadma, millisest süsteemist ajalogid tulid ning ei arvesta selle süsteemi äriloogikaga. Selle asemel et jooksutada pikka SQL päringut Jira vastu, tehakse HTTP protokolliga API päring *Timelog API* pihta, et kõik vastava perioodi ajalogid kätte saada. Sellisel kujul kätte saadud ajalogisi on lihtne ka edasi töödelda. Teise uue teenusena kasutusel olev *History API* annab crossbilli loomisel vajaliku sisendi töötaja meeskonna kohta just antud perioodil ning selle tõttu kaob vajadus töötaja meeskonda Identity API'st küsida. Sellegipoolest jääb viimane liidestus alles, sest töötaja nime mitte kusagilt mujalt ei saa ning loodud crossbillil on see nimi vajalik, et tiimijuhil oleks võimalik tehtud töö isikustada. Joonisel 18 on uuendatud andmeallikate diagramm, mis kirjeldab andmete liikumist ühe crossbilli loomisel.



Joonis 18. Tulevane andmete liikumise protsess dünaamilise crossbilli loomisel.

7.2 Dünaamilise crossbilli loomise protsess TO-BE

Lähtudes uutest andmeallikatest, muutub ka crossbilli loomise loogika kardinaalselt. Saades uuelt mikroteenuselt kätte ajalogid eraldiseisvatena, lisatakse andmebaasi uus tabel just ajalogide hoidmiseks – seega tekib igale crossbillile alamtabel ajalogidega. Crossbill on ajalogidega seotud üks-mitmele loogikaga, igal crossbillil saab olla lõpmatul arvul ajalogisi, kuid iga ajalogi peab olema seotud ühe kindla crossbilliga. Seetõttu on mõistlik kaks tabelit omavahel siduda lisades crossbilli tabeli id väli ajalogide tabeli võõrvõtmeks. Lisatud ajalogide tabel on vajalik, et iga ajalogi kohta eraldi andmeid hoida, näiteks millisesse piletisse antud aeg logiti. Lisaks on see vajalik mitme ajalogimise rakenduse liidestamisel, kui kasutajatele on vaja ära näidata, mis ajalogid tulid konkreetselt millisest ajalogimise rakendusest ning iga rakenduse kohta on eraldi viide. Vastupidiselt eelnevale protsessile, kus crossbillid moodustati koheselt SQL päringu tulemusena Jira andmebaasist, tuleb uues protsessis esmalt leida igale ajalogile müüja meeskond. Selle jaoks pöördutakse *History API* poole ning leitakse ajalogis oleva konsultandi kohta nimistu meeskondadest. Kuna igal ajalogil on olemas töö tegemis kuupäev päevase täpsusega, saab ära jagada, millisesse meeskonda konsultant kuulus see hetk, kui ta aega logis. Peale müüja meeskonna määramist igale ajalogile, liidetakse sarnased ajalogid crossbillideks viie erineva atribuudi põhjal:

- konsultandi guid;

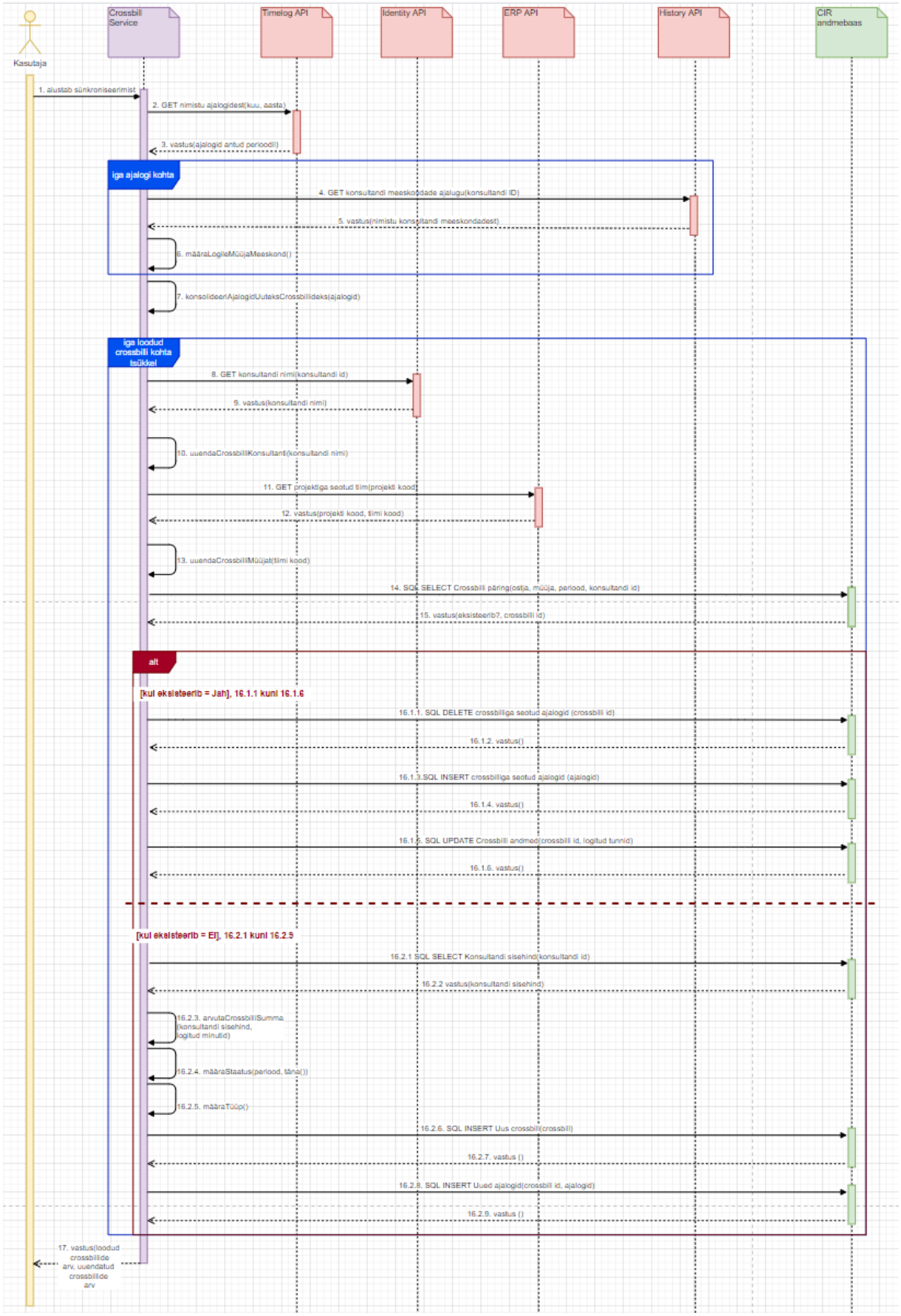
- projektikood;
- töötüüp;
- müüja;
- periood.

Töötüüp on ainuke uus väli nimetatud atribuutidest, mida crossbillidel varem ei eksisteerinud. Töö tüüp näitab, kas töö tehti tava ajast või on tegemist ületundidega. Kui tegemist on ületundidega, omab antud väli väärtust suuremat number ühest. Antud numbriga korrutatakse ka ajalogisse logitud tundide arv läbi, et saada õiglane hind. Kui töötaja on teinud tööd ületundidena, tehakse nende tundide kohta eraldi crossbill, et see erinevus kasutajatele paremini välja tuua. Tänu töötüübi lisamisele on crossbillide tabelile vaja lisada uus väärtus, mis kujutab endast töötüüpi. See on ka ainuke muudatus, mis on vaja crossbillide tabelis antud arenduse käigus läbi viia.

Lisaks töötüübile kontrollitakse crossbilli korduvust lisaks veel müüja meeskonnale, mida vanas loogikas ei tehtud. See tulenes seetõttu, et müüja meeskonnaks eeldati alati sama meeskonda, mis sünkroniseerimise hetkel konsultandi küljes oli.

Peale ajalogide crossbillideks jagamist, kontrollitakse iga crossbilli kohta, kas antud crossbill juba eksisteerib. Kontrollimiseks kasutatakse samasid atribuute, mis on üleval pool mainitud ajalogide crossbillideks koondamisel. Kui crossbill ei eksisteeri, lisatakse andmebaasi uus crossbill, ning lisaks lisatakse ajalogide tabelisse kõik selle crossbilliga seotud ajalogid. Erinevus eelneva süsteemiga on lisaks veel asjaolus, et dünaamiliselt loodud crossbilli tüübiks ei ole enam Jira vaid uus nimetus on ’Dynamic’. See tuleb asjaolust, et kõik dünaamilised crossbillid ei ole ainult Jiraga seotud. Olukorras, kus crossbill juba eksisteeris, uuendatakse olemasoleva crossbilli tundide arvu ning summat. Seejärel uuendatakse kõiki andmebaasis olevaid ajalogisi vastavalt vajadusele. Joonisel 19 on kujutatud TO-BE mudelit uuendatud crossbilli loomise protsessis järgnevusdiagrammina, järgides UML poolt seatud tavaside järgnevusdiagrammi

loomiseks [16].



Joonis 19. Järgnevusdiagramm dünaamilise crossbilli loomise protsessis [16] .

Lisaks mainitud muudatustele, lisatakse CIR süsteemile automaatne protsess, mis iga kuu viimase päeva õhtul alustab automaatselt crossbillide sünkroniseerimise protsessi, et vältida olukorda, kus osa andmeid on jäänud sünkroniseerimata kasutajate poolt.

7.3 Muudatused kasutajate vaates

Lisaks muudatustele crossbilli loomise protsessis, on uuendatud olukorras vajalik ka kasutaja vaatesse muudatusi tuua, et kuvada crossbilliga seotud uuenenud informatsiooni. Eesmärk on hoida muudatusi kasutajaliideses võimalikult väiksena, et kasutajad ei peaks juba tuttava süsteemiga uuesti harjuma, samuti on eesmärk teha kasutajaliides intuitiivseks, et iga uus kasutaja saaks kiirelt aru, kuidas enda soovitud toimingud täide viia . Iga crossbilli kohta peab olema võimalik näha, millised ajalogid selle taga on, ehk mis piletitesse logitud aja põhjal antud crossbill loodud on. Esimese variandi saab kasutaja vajutada crossbilli real olevale nupule, mis avab väikese hüpikakna. Selle sees on nimistu erinevatest piletite nimedest ning iga nimi on hüpiklink, mis tähendab et nimele vajutades avab CIR võimalusel antud pileti uuel brauseri vahelehel. Esialgne versioon põhineb Jira ajalogidel ning kasutaja viiakse Jira pileti vaatesse. Antud funktsionaalsus aitab kasutajata vahel selgitada, mis töödesse aega on kulutatud ning millega täpsemalt tegeleti.

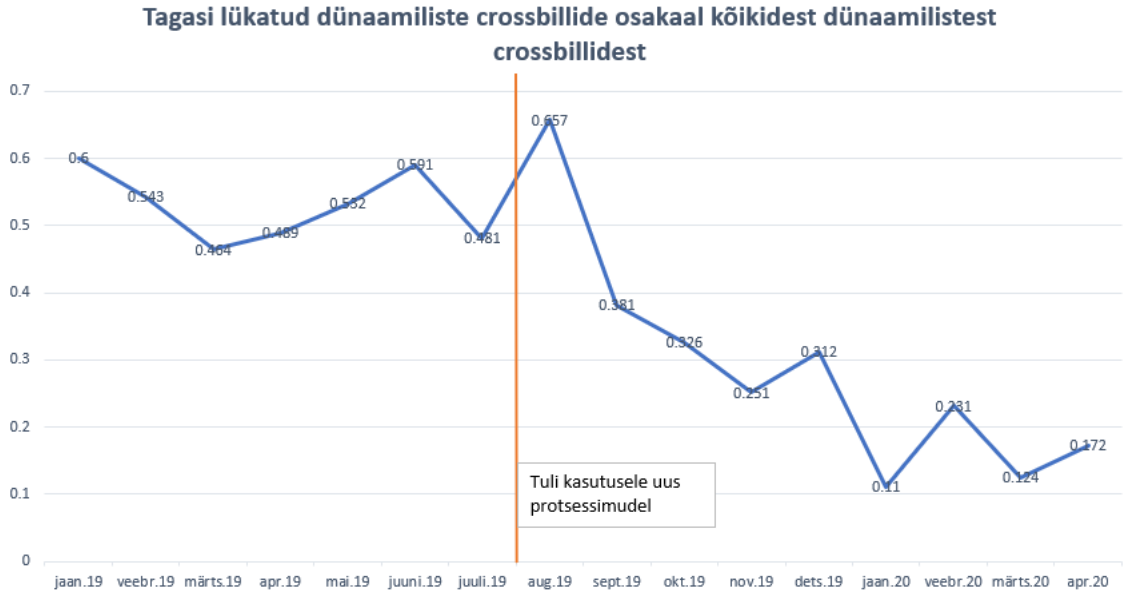
Järgmise muudatusena tuleb eristada ületundidena tehtud töö tavalisest tööst. Ületundidena tehtud töö põhjal tehakse kasutajale uus crossbill, seetõttu on tähtis, et oleks selge, mis põhjusel on näiliselt sama töö eest tekkinud kaks crossbilli juhul kui kasutaja ei märka tunnihinna erinevust. Lihtne lahendus on ületunnina registreeritud tunnis kuvada kasutajale punases ning rasvases tekstis, selliselt kuvatakse nii registreeritud tunnid, tunnihind kui ka rea kogusumma (Lisa 2 – Uus crossbillide UI demo). Lõpliku kasutajaliidese jaoks kaasas töö autor Helmese sisese disainimeeskonna, kes on spetsialiseerunud kasutajaliideste kujundamisele ning nende visandeid antud töös kujutatud ei ole.

Muudatusi kasutajaliideses tuleb tulevikus vastavalt liidestavate süsteemide eripärale, kuid tänu mikroteenuse kasutamisele on muudatused minimaalsed ning koosnevad suures osas nimetuste määramises ja väikestest äriloogika muudatustest.

8 Tulemuste analüüs

Töös kirjeldatud arendused teostati perioodil 2019 aasta suvi ning muudatused dünaamilise crossbilli loomises implementeeriti lõplikult 14.08.2019. Tagasiside arendustele on kogutud seisuga 14.04.2020 kui arendus oli kasutusel olnud täpselt kaheksa kuud. Tagasiside koosneb kahest osast – ette seatud mõõdikute hindamine ning kasutajatelt kogutud tagasiside.

Esimese mõõdikuna hindas autor tagasi lükatud dünaamiliste crossbillide arvu, mille hindamiseks teostati andmebaasi väljavõte CIR süsteemist ning hinnati andmeid viimase 16 kuu kohta, millest 9 jooksul oli kasutusel uuenenud protsessimudel. Augusti kuu crossbillid läksid samuti uue mudeli alla. Joonisel 20 on kujutatud joondiagramm, millel märgitud oranž vertikaalne joon märgib uute funktsionaalsuste sisse tulekut.

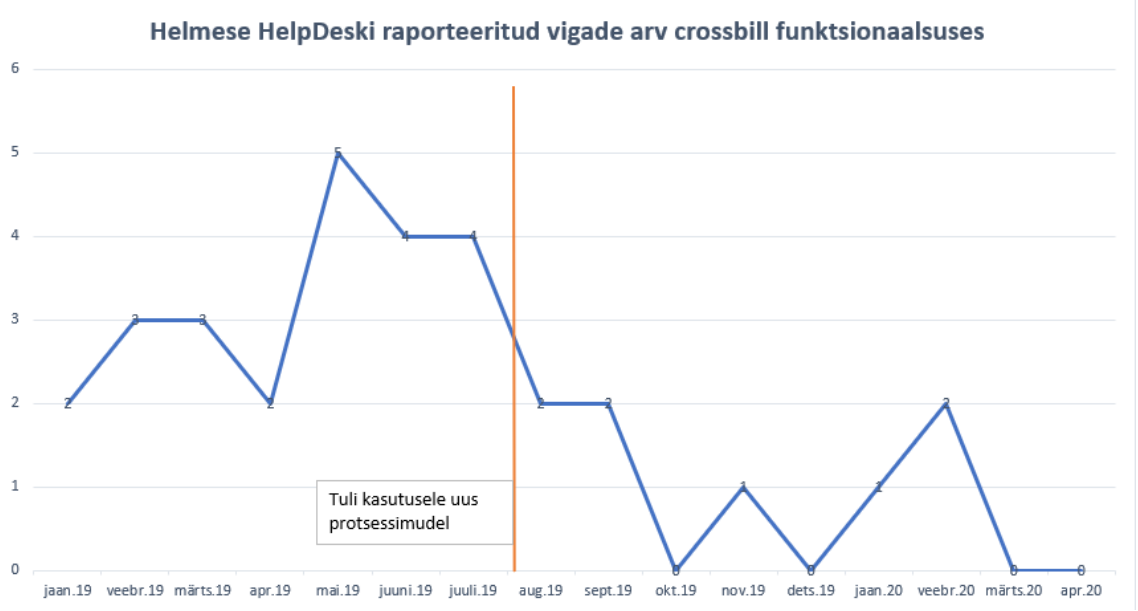


Joonis 20. Tagasi lükatud dünaamiliste crossbillide osakaal kõikidest dünaamilistest crossbillidest, eraldatud vertikaalse joonega protsessi uuendamise eelsed ning järgsed tulemused.

Jooniselt saab välja lugeda, et võrreldes varasemaga on tagasi lükatud crossbillide osakaal tunduvalt langenud, töö kirjutamise hetkel on see isegi alla sihitud eesmärgi, kuid iga kuine kõikumine on endiselt ligi 10%. Koheselt peale muudatust tagasi lükkamise osakaal tõusis hüppeliselt, mille põhjuseks on suure tõenäosusega uue funktsionaalsuse võõrandamine ning esialgne arusaamatus uue loogika taga. Septembri jooksul tehti kõigile Helmese tiimijuhtidele koolitus uue crossbilli funktsionaalsuse kasutamise osas, mis väljendub ka joonisel ligi 50%-lise langusena tagasi lükatud crossbillide osas. Kõiki

tagasilükkamisi ei saa samas lugeda vigadeks süsteemis, kohati kokkulepped muutuvad või lepitakse kokku mingi teine hind sellest, mis süsteemis on konsultandile määratud – sellisel juhul dünaamiline crossbill lükatakse tagasi ning luuaks käsitsi uus arve vastavalt kokkuleppele. Sellel põhjusel loeb töö autor tulemuse antud mõõdiku kohta edukaks.

Teine ette seatud mõõdik oli HelpDeski loodud vigasid kirjeldavate piletit arv. Vaatluse periood on sama mis eelmise mõõdiku korral. Joonisel 21 on välja toodud graafik, mis kujutab mitu vea piletit on ühes kuus tehtud viimase 16 kuu jooksul. Oranž joon märgib uue funktsionaalsuse kasutusel võttu.



Joonis 21. Helmese HelpDeski raporteeritud vigade arv crossbill funktsionaalsuses, eraldatud vertikaalse joonega protsessi uuendamise eelsed ning järgsed tulemused.

Graafikut analüüsides on selge, et vea pileteid on tehtud vähem võrreldes uuenduste eelse ajaga. Koheselt peale uuendust tehtud neli veateadet (2 augustis ja 2 septembris) olid seotud suures osas kasutajate poolse arusaamatusega süsteemis. Peale septembris aset leidnud koolitust standardiseerus veateavituste arv, kuid on hetkel endiselt veel kõrgemal pool mõõdikut. Seatud eesmärk oli 1 veateavitus 3 kuu jooksul, kuid perioodil 2019 september kuni 2020 aprill oli tulnud 6 teavitus 8 kuu jooksul, mis on ligi 2 korda kõrgem sihitud eesmärgist. Töö autor tõdeb, et ette seatud eesmärk antud mõõdikule ei olnud realistlik ega hästi läbi mõeldud. Ka hästi arendatud süsteemi korral tekib vigu ning raporteeritud vigade arv ei mõõda adekvaatselt toimunud arenduse ärilist mõju ettevõttele.

Viimase tagasiside vormina kogus autor tagasisidet tiimijuhtidelt intervjuu vormis. Intervjuu viidi läbi vabas vormis, kus tiimijuhid said jagada enda kogemusi seoses muudatustega ning millistes osades näeksid nad paranemisruumi. Intervjuu viidi läbi sama valimiga mis esimene intervjuu, välja arvatud üks meeskonnajuht, kes tagasiside hetkeks oli ettevõttes lahkunud. Intervjuus osalesid 4 tiimijuhti Minski ning 10 tiimijuhti Tallinna kontorist. Intervjuu käigus tuli välja, et 5 vastajat kasutavad *Timelog API* teenust väljaspool dünaamilise crossbilli protsessi ning 3 vastajat kasutavad *History API* teenust. Helmese siseselt on vastavalt vajadusele tiimijuhtidel võimalik ligipääsu taotleda teenustele, et andmestikku iseseisvalt kasutada. Mõlemad loodud teenused andsid vastajatele lisaväärtust ning vastajad, kes neid täna veel kasutanud ei olnud, olid huvitatud tulevikus selle kaalumisest. Kõik vastajad nägid paranemist dünaamilise crossbilli loomise loogikas ning lahendatud said kõik varem mainitud probleemid. Neljateistkümnest vastajast 9 tõdesid, et aeg mis nad kulutavad crossbillidega tegelemiseks on tunduvalt vähenenud ning ülejäänud 5 jaoks on kulutatud aeg sama. Kõigi vastajate jaoks tõusis crossbillide usaldatavus tunduvalt ning vaid üks vastaja oli viimase 8 kuu jooksul kohanud probleemi, kus crossbilli andmetel oli viga sees. Järgmiste vajalike arendustena näevad tiimijuhid võimalust luua perioodilisi crossbille, mis igakuiselt automaatselt tekivad ning võimalust liidestada Toggl *Timelog API* külge.

Kogu tagasiside kokku võttes järeldab töö autor, et tehtud projekt oli edukas esinenud probleemide lahendamiseks ning lõi lisaks lisaväärtust uute teenuste näol, mida saab kasutada teistes süsteemides. Töö kirjutamise ajaks kasutatakse *History API* teenust kolmes Helmese sisemises rakenduses ning ajaloolisi andmeid hoiustatakse mitmete erinevate objektide kohta. *Timelog API* arendus algab 2020 aasta suvel, mille käigus liidetakse Toggl Helmese sisesüsteemide võrguga, kasutades selle Reports API teenust. Tänu tehtud arendustele, on see hõlpsalt *Timelog API* külge liidestatav ning andmeid tarbivad süsteemid muudatusi ei vaja.

9 Kokkuvõte

Antud töö eesmärgiks oli analüüsida probleeme tänases automaatses meeskondade vahelises arvelduse protsessis ettevõtte Helmes AS siseselt ning pakkuda välja optimaalsed lahendused protsessi puuduste parandamiseks. Protsess töötab Helmese sisemiselt arendatud rakenduse CIR osana.

Töös on analüüsitud puudusi tänases protsessis, põhinedes kasutajate tagasisidele ning aja jooksul kogutud veateadetele, mis antud protsessi kohta raporteeritud on. Lisaks lähtuti analüüsis veel ettevõtte sisestest tarkvaraarenduse arhitektuuritavadest, mis peaks olema rakendatud igas Helmese sisemises süsteemis. Lisaks on tehtud detailne analüüs tänasele dünaamilise crossbilli loomise protsessile, et paremini tuvastada nõrkasid kohti uue meeskondade vahelise arve loomises. Põhinedes kasutajate tagasisidele ning ettevõttes rakendatavale mikroteenuste arhitektuurile, jõudis töö autor järelduseni, et protsessi parandamiseks on vaja luua kaks uut mikroteenust – üks kogub ja tagastab töötajate ajaloolisi andmeid ning teine vahendab töötajate registreeritud ajalogisi ajalogimise rakenduste ja andmete lõpptarbija vahel. Mõlema uue teenuse jaoks viis töö autor läbi vajaliku analüüsi koos nõuetega teenustele, et alustada kahe uue rakenduse arendust. Lõpliku osana analüüsiti antud töös uute teenuste liidestamist dünaamilise crossbilli loomise protsessi ning kujundati kõnealune protsess ümber vastavalt uuenenud ärilistele nõuetele.

Töö tulemusena valmis vajalik analüüs, et alustada arendust dünaamilise crossbilli loomise protsessi uuendamiseks. Tehtud analüüsi põhjal arendati valmis kaks uut teenust ning täiustati olemasolevat arvete loomise protsessi. Tehtud arenduste tulemusena lahendati kõik kasutajate nimetatud probleemid ja loodi täielikult automatiseeritud ja efektiivselt töötav dünaamiliste crossbillide loomise protsess. Lisaks loodi uute teenuste näol lisaväärtust ka teiste Helmese siseste rakenduste jaoks, mis antud andmestikele kasutust leiavad. Samuti paranes üleüldine sisesüsteemide arhitektuur Helmeses, saades lahti olukorrast, kus kaks andmeid tarbivat rakendust suhtlevad omavahel ilma vaheteenuseta.

Töö järgselt arendatakse parandatud protsessi edasi ning lisatakse dünaamilise crossbilli loomisesse ka teisi ajalogimise rakendusi peale Jira. Tänu mikroteenuste arhitektuuri implementeerimisele ajalogide liikumisel on lisanduvad arendused vähe mahukad.

Kasutatud kirjandus

[1] e-teatmik, [WWW], http://vallaste.ee/, (01.05.2020)

[2] Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures, 2000 [WWW], https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm, (04.05.2019)

[3] Introducing JSON, [WWW], https://www.json.org/json-en.html (10.05.2020)

[4] The Business Analyst's Survival Guide: Managing Interpersonal Dynamics and Leveraging Knowledge of Repeat Behavior Patterns. / ed. LeAnn Simonson. CreateSpace (2012)

[5] Introduction to JSON Web Tokens, [WWW], https://jwt.io/introduction/, (05.05.2019)

[6] What is Docker?, [WWW], https://opensource.com/resources/what-docker (10.05.2020)

[7] A Universally Unique Identifier, [WWW], https://www.ietf.org/rfc/rfc4122.txt (01.05.2020)

[8] What is an API Gateway?, [WWW], https://nordicapis.com/what-is-an-api-gateway/, (04.03.2020)

[9] Dave Nicolette. Utilization thinking vs. throughput thinking. [WWW], https://davenicolette.wordpress.com/2012/02/04/utilization-thinking-vs-throughput-thinking/ (15.05.2020)

[10] Dave Nicolette. Pros and Cons of dedicated teams, [WWW], https://davenicolette.wordpress.com/2012/07/28/pros-and-cons-of-dedicated-teams/ (15.05.2020)

[11] Ihor Feoktistov. The Ultimate Guide to Dedicated Software Development Teams, [WWW], https://relevant.software/blog/the-ultimate-guide-to-dedicated-software-development-teams/ (15.05.2020)

[12] "Crossbill process pre-analysis", ettevõttesisene dokumentatsioon

[13] Building Microservices: Designing Fine-Grained Systems. / ed. Sam Newman. O'Reilly (2015)

[14] Kamil Figura. Why you should choose the microservices architecture?, [WWW], https://pretius.com/why-you-should-choose-the-microservices-architecture/ (15.05.2020)

[15] Hospital Management: UML Class Diagram Example, [WWW], https://www.uml-diagrams.org/examples/hospital-domain-diagram.html (16.05.2020)

[16] UML Sequence Diagrams, [WWW], https://www.uml-diagrams.org/sequence-diagrams.html (19.05.2019)

[17] Mastering the Requirements Process: Getting Requirements Right. 3rd edition / ed. Robertson, S., Robertson J. Pearson Education, Inc. [Online] (01.06.2019)

[18] "Helmes sisesüsteemide arhitektuuri tavad", ettevõttesisene dokumentatsioon

[19] Zuul, [WWW], https://github.com/Netflix/zuul/wiki, (02.02.2020)

[20] Web API Design: Crafting interfaces that Developers Love. / ed. Brian Mulloy. Apigee (2002). [Online] (09.05.2019)

[21] Privacy by Design, [WWW], https://gdpr-info.eu/issues/privacy-by-design/, (04.06.2019)

[22] Flowchart Tutorial, [WWW], https://www.visual-paradigm.com/tutorials/flowchart-tutorial/ (18.05.2019)

[23] You Arent Gonna Need It, [WWW], http://wiki.c2.com/?YouArentGonnaNeedIt (05.06.2019)

Lisa 1 – Töötaja meeskondade ajaloo demo

	id [PK] bigint	töötaja_guid character varying	meeskond character varying	alates date	kuni date
1	1	abcdefg1997	OLDTEAM	2019-04-05	2019-10-10
2	2	abcdefg1997	NEWTEAM	2019-10-10	[null]

Lisa 2 – Uus crossbillide UI demo