

“TTJA e-teenuste keskkonna TARVIK arendus”

Visioon

Riigihange nr 291973

Tellija:

**Keskkonna-
ministeeriumi
Infotehnoloogia-
keskus**

Dokumendi kuupäev:

28.04.2025

Dokumendi autorid:

**Katrin Vaisma,
Kertrud Järg, Rait
Raidma, Marten Tall**

Sisukord

Sisukord.....	2
1 Sissejuhatus	4
1.1 Mõisted ja lühendid.....	4
2 Executive Summary – Visioon TTJA TARVIK arenduseks	7
3 Visioon	8
3.1 Visiooni lähtepunktid.....	8
3.1.1 Analüüsi, disaini ja arenduse koostöö ja projektijuhtimise metoodika – <i>Dual Track Scrum</i>	9
3.2 Menetlustoote kasutajaliidese arendus.....	10
3.2.1 Ülevaade menetlusprotsessi hetkeolukorrast.....	10
3.2.2 Menetluse äriprotsessi nõuded ning UI/UX põhimõtete rakendamine	13
3.2.3 Võimalikud lahkevused ja ebakõlad võrreldes eelnevalt teostatud kasutajakogemuse analüüsiga	15
3.2.4 Menetleja kasutajaliidese arenduse tegevuskava - analüüs, disain ning arendus	16
3.3 Mikroteenuste arhitektuur	30
3.3.1 Mis on mikroteenusarhitektuur?.....	30
3.3.2 Peamised eelised	30
3.3.3 Peamised probleemid	32
3.3.4 Mikroteenusteks jaotamine	34
3.3.5 Mikroteenustel põhineva süsteemi arhitektuur	36
3.3.6 Menetlusprotsesside loomine	43
3.3.7 Üleminekuplaan.....	43
3.3.8 Riskid ja maandamismeetmed	45
3.4 Modelleerimismeetodi BPM tööprotsesside ja protsessimootori tehniline rakendamine	46
3.4.1 Protsessimootori valik	46
3.4.2 Protsessidiagrammid	47
3.4.3 Teenuste vaheline suhtlus	47
3.4.4 Jälgitavus	48
3.4.5 Arhitektuur	49
3.4.6 Kompenseerivad tegevused	51
3.4.7 Korduvad tegevused	51
3.4.8 Rakendamine	52
3.4.9 Üleminekuplaan.....	53
3.4.10 Riskid ja maandamismeetmed	53
3.5 Mikroesitluskihid	55
3.5.1 TARVIKu mikroesitluskihtide üldarhitektuur	55

3.5.2 Tehnoloogiad	57
3.5.3 Andmevahetus.....	58
3.5.4 Disain	59
3.5.5 Autentimine, autoriseerimine ja infoturve.....	60
3.5.6 Testimine	61
3.5.7 Kasutuselevõtt	61
3.5.8 Integratsioon väliste süsteemidega.....	66
3.5.9 Riskid ja maandamismeetmed	68
4 Tehisaru kasutamine.....	71
4.1 Automaatne menetlemine	71
4.2 Klienditugi	72

1 Sissejuhatus

Meil on hea meel esitada AS Helmes pakkumus „TTJA e-teenuste keskkonna TARVIK arendus“ hankes.

Tarbijakaitse ja Tehnilise Järelevalve Amet (TTJA) on viimastel aastatel suunanud olulist tähelepanu oma digivõimekuse tõstmisele ning e-teenuste ühtlustamisele. Erinevate süsteemide – sh Majandustegevuse Register (MTR), Tarbijakaitse ja Tehnilise Järelevalve Ameti Infosüsteem (JVIS) ja Numbri Broneerimise Andmekogu (NBA) – killustatus on loonud olukorra, kus teenuste kasutamine ei ole järjepidev ega alati kasutajasõbralik.

TTJA alustas 2023. aastal uue, mikroteenustel põhineva, e-teenuste platvormi arendustega (TARVIK), kuhu tuuakse aja jooksul üle kõik TTJA poolt pakutavad teenused erinevatest süsteemidest (MTR, JVIS, NBA). **TARVIKu eesmärk on pakkuda oma kasutajatele kaasaegset teenust, tagades sealjuures süsteemi töökindluse ning kasutusmugavuse.** Selle eesmärgi saavutamiseks on juba 2024. aastal teostatud TARVIKu kasutusmugavuse analüüs.

Hetkel on TARVIKu keskkonda olemasolevast MTRi süsteemist üle toodud 25 majandustegevusteadet koos osade tugifunktsionaalsustega. Iteratiivselt tuuakse funktsionaalsuseid teistest süsteemidest juurde. **Hetkel ei ole seoses menetlusprotsessidega (k.a loataotlused) TARVIKusse veel ühtegi funktsionaalsust loodud. Samas on loodud menetleja töölaud ning registripäeviku ja registritoimiku kuvad, mida võib liigitada menetlustoote alla.**

Käesoleva dokumendi eesmärk on kirjeldada pakkuja visiooni hanke raames loodavatest tehnilisest lahendusest ning kirjeldada eesmärkide saavutamiseks vajaminevad tegevused.

1.1 Mõisted ja lühendid

Mõiste	Selgitus
AMQP	<i>Advanced Message Queuing Protocol</i> . Avatud standardiga sõnumiprotokoll, mis võimaldab rakendustel usaldusväärselt ja järjekorras andmeid vahetada sõnumijärjekordade kaudu. See eraldab sõnumite saatjad ja vastuvõtjad, võimaldades süsteemidevahelist lahtist sidumist ja paremat skaleeritavust.
API	Liides, mille kaudu saab rakendusega suhelda.
Backend	Serveris jooksatav rakenduse poolne osa, mis käivitab äriloogikat.
BPMN	<i>Business Process Model and Notation</i> . Graafiline standard äriprotsesside kujutamiseks, mis võimaldab neid selgelt visualiseerida ja erinevate osapoolte vahel ühtemoodi mõista. See kasutab sümboleid nagu tegevused, otsused ja vood protsessi kirjeldamiseks.
Camunda	Protsessi- ja otsustusmootor, mis toetab BPMN- ja DMN-standardeid ning võimaldab äriprotsesside ja otsustusloogika automatiseerimist. Seda saab integreerida erinevatesse süsteemidesse ning kasutada mikroteenuste ja töövoogude haldamiseks.
CORS	<i>Cross-Origin Resource Sharing</i> . Klientrakenduse-välise domeenidega suhtlemise reeglid.
CSP	<i>Content Security Policy</i> . Failide jm ressursside laadimise reeglid.
DMN	<i>Decision Model and Notation</i> . Standard otsustusreeglite modelleerimiseks ja dokumenteerimiseks. See võimaldab ärikasutajatel reegleid visuaalselt kujutada ja IT-süsteemides automatiseerida.

Frontend	Kasutaja poolne vaade, mille kaudu saab kasutaja rakendusega suhelda.
Git	Tarkvara lähtekoodi versioonihalduse tööriist.
gRPC	Kaugkutsungite raamistik, mis võimaldab eri süsteemidel efektiivselt omavahel suhelda. See kasutab Protobuf-serialiseerimist ja töötab HTTP/2 protokollil.
Helm chart	Pakett, mis sisaldab kogu vajalikku konfiguratsiooni rakenduse paigaldamiseks Kuberneteses. See määratleb ressursid (nt <i>Deployment</i> , <i>Service</i>) ja võimaldab neid lihtsalt uuesti kasutada ning hallata.
HTTP	<i>HyperText Transfer Protocol</i> . Võrguprotokoll, mida kasutatakse andmete, eriti veebilehtede, edastamiseks kliendi (nt veebibrauseri) ja serveri vahel. See töötab päring-vastus põhimõttel, kus klient saadab serverile päringu ja server vastab soovitud andmetega.
Istio	Teenuste võrgustiku (service mesh) lahendus, mis haldab mikroteenuste vahelist liiklust, turvalisust, jälgitavust ja poliitikaid Kuberneteses ilma rakenduse koodi muutmata.
JVIS	Tarbijakaitse ja Tehnilise Järelevalve Ameti Infosüsteem.
KeMIT	Keskkonnaministeeriumi Infotehnoloogiakeskus.
Kubernetes	Platvorm konteinerite haldamiseks ja orkestreerimiseks, mis automatiseerib rakenduste paigaldust, skaleerimist ja haldust.
MFE	<i>Micro front-end</i> (Mikroesitluskiht). Esitluskihirakendus, mis on autonoomselt tarnitav ning käivitatav.
MFE host	MFE-sid orkestreeriv kasutajaliidese komponent.
MFE remote	Tarbitav/orkestreeritav MFE.
MFN	Mittefunktsionaalsed nõuded kirjeldavad süsteemi omadusi, mis ei ole seotud konkreetse funktsionaalsusega, vaid määravad ära kvaliteedi, töökindluse, jõudluse, turvalisuse, skaleeritavuse jms.
Module federation	Keskne esitluskihi moodulite (MFE-de) jagamise tehnoloogia.
Mono-repo	Mitut moodulit/rakendust koondav Git repositoorium.
Multi-repo	Iga mooduli/rakenduse jaoks eraldi Git repositooriumi kasutamine.
mTLS	<i>mutual TLS</i> . Turvaprotokoll, kus nii server kui ka klient autentivad üksteist, kasutades digitaalseid sertifikaate. See tagab kõrge turvalisuse, kuna mõlemad osapooled peavad enne ühenduse loomist enda identiteeti tõendama.
MTR	Majandustegevuse Register.
npm	<i>Node Package Manager</i> . Keskne esitluskihi komponentide haldussüsteem.
OpenTelemetry (OTel)	Raamistik, mis võimaldab rakendustest koguda jälgimis-, logi- ja meetrikaandmeid ühtses vormingus. See aitab paremini mõista süsteemide käitumist ja lihtsustab jälgitavuse tööriistade integreerimist.
OTLP	<i>OpenTelemetry Protocol</i> . Standardiseeritud protokoll, mida kasutatakse jälgitavusandmete (jäljed, meetrikad, logid) edastamiseks OpenTelemetry-põhistes süsteemides. See võimaldab kasutada ühtset andmeedastust jälgimisagendi ja analüüsisüsteemi vahel.
POC	<i>Proof of concept</i> . Lahenduse põhimõtete katsetamine. Selle raames testitakse ära süsteemi olulisemad kohad, kuid ei arendata valmis täielikku rakendust.
Pod	Väikseim juurutusüksus Kuberneteses, mis sisaldab ühte või mitut konteinerit.

Prometheus	Monitooringusüsteem ja andmebaas, mis kogub ja salvestab aja jooksul mõõdetud andmeid. Seda kasutatakse peamiselt süsteemide jõudluse jälgimiseks.
Sanity testing	Muudatuse/paranduse kontrollimine.
Sidecar	Muster, kus abiteenus käitatakse samas Podis koos põhiteenusega, et pakkuda lisafunktsionaalsust nagu logimine, monitooring või turvalisus. See võimaldab põhiteenust laiendada ilma selle koodi muutmata.
Smoke testing	Suitsutestimine. Põhifunktsionaalsuse toimimise kontroll pärast muudatusi.
SSO	<i>Single Sign-On</i> . Autentimise mehhanism, mis võimaldab kasutajal sisse logida ühte süsteemi ja pääseda seejärel automaatselt ligi kõigile teistele seotud rakendustele ilma uuesti sisselogimist nõudmata.
Tree shaking	Optimeerimise eesmärgil tarkvarakoodi eemaldamine sõltuvuste mahu vähendamiseks.
TTJA	Tarbijakaitse ja Tehnilise Järelevalve Amet.

2 Executive Summary – Visioon TTJA TARVIK arenduseks

Helmese visioon TTJA e-teenuste keskkonna TARVIK arenduseks tugineb meie pikaajalisele kogemusele riiklike infosüsteemide ja menetlusplatvormide ehitamisel, pakkudes tellijale skaleeritavat, paindlikku ja kasutajasõbralikku lahendust. Meie lähenemine arvestab KeMIT tehnoloogilise profiiliga, Eesti riigi IT-arhitektuuri põhimõtetega ning lõppkasutaja ootustega.

Visiooni keskmes on neli olulist alustala:

1. **Kasutajaliidese arendus** – lähtudes olemasolevatest disainijuhistest ja Trinidad Wisemani UI/UX-analüüsist, pakume välja rollipõhise töölaudade lahenduse, intelligentsete soovitude süsteemi ning sujuva ülemineku kasutajate harjumustest uue lahenduseeni. Eraldi tähelepanu on pööratud visuaalse hierarhia ja navigatsiooni selgusele.
2. **Mikroteenuste arhitektuur** – alustuseks seletame mis on mikroteenusarhitektuuri kasutamise eelised. Samas toome ka välja, milliseid probleeme see võib põhjustada ning kuidas neid lahendada. Selgitame, mida tuleks silmas pidada süsteemi jaotamisel mikroteenusteks ja kuidas seda edukalt teostada. Võttes arvesse teadmisi olemas olevast süsteemist ning süsteemile esitatud nõudeid, pakume välja arhitektuuri kogu süsteemi, jälgitavuse kui ka üksiku mikroteenuse vaatest, mis tagaks TARVIKu korrektse töö ja turvalisuse.
3. **Protsessimootori rakendamine** – kirjeldame visiooni, kuidas integreerida protsessimootor TARVIKu infosüsteemi. Selgitame koos põhjendustega milline hakkaks olema loodav arhitektuur koos põhjendustega. Samuti toome välja, kuidas toimuks suhtlus mikroteenuste ja protsessimootori vahel, kuidas kasutada protsessimootorit andmete terviklikuse tagamiseks, mis probleemid võivad esile kerkida ning kuidas neid lahendada.
4. **Mikroesitluskihtide arhitektuur** – kirjeldame visiooni TARVIKu mikroesitluskihtidele üleviimiseks, toetudes varasemale kogemusele eesti.ee portaalis. Võrdleme erinevaid tehnoloogilisi valikuid, pakume lahendusi levinud probleemidele ning meetmeid riskide maandamiseks. Selgitame eesti.ee portaaliga integreerimise võimalusi.

Visiooni loomise iga arhitektuurne ja tehniline otsus toetub **varasematele kogemustele ning praktilistele õppetundidele** sarnastes riiklikes süsteemides, kus skaleeritavus, protsesside jälgitavus ja kasutusmugavus on osutunud kriitiliseks. Tehnoloogiliste otsuste valikul on lähtutud ideest luua **paindlik ja jätkusuutlik platvorm**, mis on valmis nii tehnilisteks muudatusteks kui ka menetlusprotsesside sisuliseks arenguks.

Helmes on varasemalt TARVIK süsteemi arendustesse panustanud nii **arendus- kui ka analüüsitegevustes**, mis annab meile süvitsi mineva arusaamise olemasolevast lahendusest, selle tugevustest ja kitsaskohtadest. Tänu sellele kogemusele suudame pakkuda **asjakohast ja realistlikku visiooni**, mis ei alga nullist, vaid toetub teadlikult olemasolevatele komponentidele.

Meie visioon lähtub põhimõttest, et tehniline täiuslikkus peab käima käsikäes reaalse kasutajamugavuse, arendatavuse ja paindlikkusega.

3 Visioon

3.1 Visiooni lähtepunktid

Meie arusaam TTJA vajadusest tugineb nii hanke tehnilisele kirjeldusele, varasematele kasutajakogemuse uuringutele (sh 2024. a klienditoote UI/UX analüüs), toimunud infotunnile kui ka praktilistele kogemustele avaliku sektori digiteenuste disainimisel. Mõistame, et TTJA ei otsi pelgalt tehnilist teostajat, vaid partnerit, kes:

- suudab mõtestada keerukaid tööprotsesse, leida optimeerimiskohti ja tõlkida need toimivaks liideseks,
- omab kogemust suurte riiklike süsteemide disainimisel ja arendamisel,
- suudab kavandada lahendusi, mis arvestavad nii olemasolevate süsteemide pärandit kui ka tulevikku suunatud arhitektuurilisi põhimõtteid.

Disainides TARVIKu keskkonda tuleb silmas pidada, et selle kaugeleulatuvam visioon on olla ühtne keskkond, mis toob kokku nii lõppkasutaja poolt tehtavad toimingud, TTJA klienditoote kui ka menetlemise. Selleni liigutakse osade kaupa - täna on seal peamiselt majandustegevusteadete esitamise ja haldamise funktsionaalsus koos mõningate tugifunktsionaalsustega (näiteks autentimine ja autoriseerimine, kontaktandmete haldus, klassifikaatorite haldus, teavituste saatmine jne). Edasiarenduste puhul on soovitatav alustada enam kasutatavatest teenustest.

Visiooni elluviimisel ja **lahenduste jätkusuutlikkuse** tagamisel peame oluliseks silmas pidada ja võimalusel rakendada järgmist:

- **Täisdisainisüsteemi kasutuselevõtt TARVIKus.** Täisdisainisüsteemi (disain + *front-end* koodikomponentide kogum) ühtlustab ja parandab kasutajakogemust, hõlbustab vastavust ligipäätavuse nõuetele ning kiirendab edasistes arendustes kasutajaliidese töid vähendades seeläbi kulusid.
- **Erinevate teenuste tervikvaate (nt loataotlus ja selle menetlusprotsess) osapoolte rahulolu mõõtmine.** Menetlejate ja kasutajate tagasiside annab ülevaate nii taotluse kui menetluse kitsaskohtadest kui sisendi, et süsteem saaks olla pidevalt arenev ja jätkusuutlik.
- **Muudatuste juhtimise olulisus.** Süsteemi uuendus ja kokku toomine ei ole vaid tehnoloogiline muudatus, vaid nõuab ka ametnike kui teenusepakkujate kaasamist muudatustesse, muudatuste juhtimist organisatsioonis tervikuna, koolitamist ning tööd inimestega. Kui ollakse valmis ja võimelised muudatustest aru saama ja nendega kaasa minema, aitab see ühe komponendina tagada loodava süsteemi jätkusuutlikkust.

Oluline on läbi mõelda, kuidas seotud osapooltele kommunikeerida protsessis planeeritavaid muudatusi. Soovitame muudatuste läbimõtlemissse ja uute protsesside disaini kaasata neid, kelle tööd menetlusprotsess otseselt puudutab, et ka nemad saaks kaasa rääkida, kuidas oma tööd teistmoodi või targemalt teha (kuivõrd optimeeritud protsessid võivad tähendada kas teistsuguseid tööülesandeid või mõnel juhul ka rollide kaotamist).

Kaasates menetlejaid ja teisi rolle, kasutades nt disainimõtlemissse metoodikaid, et “kastist välja” lahendusi otsida minimeerime riski, et läheme uuendama olemuselt vanu protsesse ning lõikame läbi võimalused innovatsiooniks.

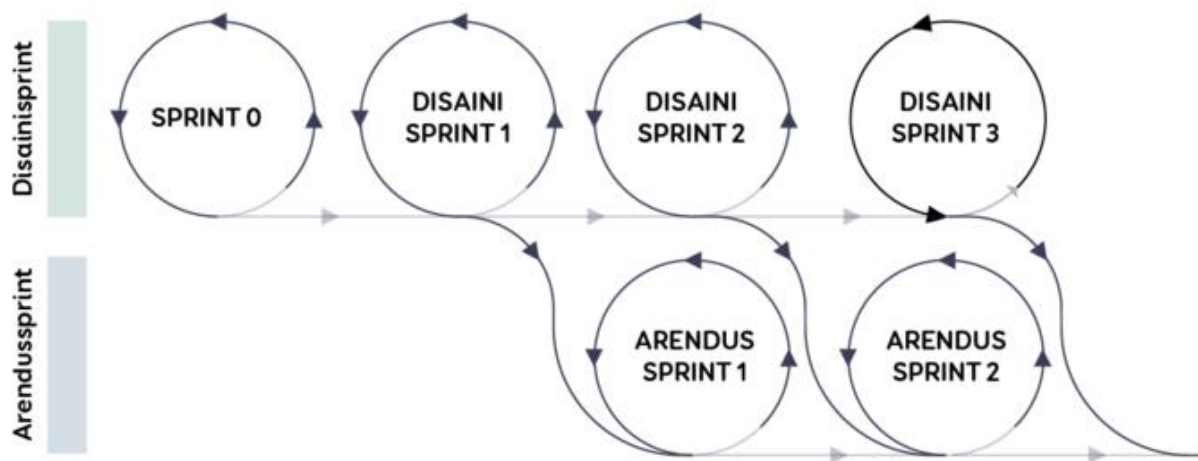
Visiooni elluviimise arendustsükli juhime *Dual Track Scrum* meetodil (vt järgmine ptk), et liikuda edasi agiilselt, kus disain ja analüüs annab arendusele ette teostatavaid loogilise tükke.

3.1.1 Analüüsi, disaini ja arenduse koostöö ja projektijuhtimise metoodika – *Dual Track Scrum*

Disaini, analüüsi ja arenduse sujuvaks koostööks ja projektijuhtimise raamistikuks soovitame kasutada **Dual Track Scrum meetodit**. Läbinuna ühe loataotluse tüübi menetluse (nt Merenduse load) eelanalüüsi ja disaini etapid 1-6, millest on täpsemalt kirjutatud lk 15 (ehk joonisel Sprint 0 faas), saab liikuda konkreetse loataotluse menetluse arendusfaasi. Seejuures paralleelselt jääb kestma ka disaini- ja analüüsitsükkel, mis erinevatele loogiliselt seostatud arendusetappidele detailsemaid vaateid ja arenduspileteid ette valmistab. Seejärel saab paralleelselt alustada omakorda uue loataotluse tüübi (nt Taksoload) menetlusprotsessi Sprint 0 faasiga.

Pakkuja kogemusel võimaldab selline lähenemine hoida analüüsi ja UX disaini ajakohasena ning hoiduda analüüsi „aegumisest“, mis võib juhtuda kui see valmib liialt palju enne vastava iteratsiooni arendustööde alustamist.

Dual Track Scrum



Joonis 1. Analüüsi, disaini ja arenduse koostöö ning projektijuhtimise metoodika - Dual Track Scrum.

Võrreldes teiste levinud metoodikatega nagu *Kanban*, tavaline *Scrum* või *Waterfall*, pakub just **Dual Track Scrum** olulisi eeliseid antud projekti kontekstis. *Kanban*, mis sobib hästi pideva voo ja töömahu ühtlustamiseks, ei võimalda piisavalt struktureeritud lähenemist paralleelselt toimuvate analüüsi- ja arendustegevuste koordineerimiseks. Tavaline *Scrum*, kuigi iteratiivne ja paindlik, ei paku eraldi "rada" disaini- ja uurimistegevusteks, mis on kriitilise tähtsusega kasutajakeskse arenduse puhul.

Waterfall meetod omakorda eeldaks kogu analüüsi lõpuleviimist enne arendustegevuste algust, mis tähendaks arendustiimide ebaühtlast ressursikasutust ning ei võimaldaks paindlikku reageerimist eelanalüüsi tulemitele. *Dual Track Scrumi* kasutamine võimaldab meil optimeerida nii arendusressursside kasutust kui ka tagada, et disaini- ja analüüsistegevused toetavad järjepidevalt arendustööd, säilitades samal ajal võimekuse reageerida muutuvatele vajadustele ja uutele sisenditele.

3.2 Menetlustoote kasutajaliidese arendus

3.2.1 Ülevaade menetlusprotsessi hetkeolukorrast

TTJA erinevad loateenused, sh nende menetlemine on jagunenud hetkel mitme süsteemi vahel – Majandustegevuse Register (MTR) ja TTJA Infosüsteem (JVIS). Lisaks, numbrilubasid hallatakse läbi Numbri Broneerimise Andmekogu (NBA), kuigi numbriloo taotluse esitamine ja selle menetlemine käib JVIS süsteemis.

Lisaks sellele, et menetlused käivad läbi mitme süsteemi, on ka erinevatel loateenustel erinevad menetlusprotsessid, mis lisab menetlejate igapäeva töösse veelgi keerukust.

Järgnevalt toome mõned näited olemasolevatest menetlusprotsessidest, mis ilmestab loataotluste menetlemise erinevusi.

Näide 1. MTR

Majandustegevuse Register (MTR) on Tarbijakaitse ja Tehnilise Järelevalve Ameti (TTJA) hallatav register, kus hallatakse majandustegevusteateid ning TTJA pädevusse kuuluvaid tegevuslubasid. Register võimaldab ettevõtjatel esitada vastavaid taotlusi ning ametiasutustel neid menetleda.

MTRis käib menetlus vastavate loataotluste kuvadel. Näitena toome Merenduse tegevusala loataotluse menetlusprotsessi:

Taotluse andmed

Number	20250411-782297
Tegevusala	Merendus
Menetluse tähtaeg	01.05.2025
Taotluse esitaja	Kristina Tija-Mtr (tel: 4545454, e-post: kristina@test.ee)
Taotluse sisestaja	Kristina Tija-Mtr
Sisestamise aeg	11.04.2025 10:38
Esitamise aeg	11.04.2025 10:39
Esitamisviis	E-post
Taotluse saabumise kuupäev	11.04.2025
Ametkondlikuks kasutamiseks	Ei
Riigilõiv	
Viitenumber	2900082993
Riigilõiv tasutud	
Staatuse	Taotluse läbivaatamine
Menetleja kommentaar	

Menetlejad

Vormi andmed

Taotluse versioonid

Dokumendid

Andmed kontrollitud

Haldusasutus	Ametikoht	Nimi	Isikukood	Telefon	E-post	
Tarbijakaitse ja Tehnilise Järelevalve Amet		Mary Änn O'Connež-Šuslik	60001019906			
Leiti 1 kirje						

Tagasi nimekirja

Lisa menetleja

Muuda

Pikenda menetluse tähtaega

Märgi taotlus läbivaadatuks

Märgi tagasi võetuks

Teavita ettevõtjat puudustest

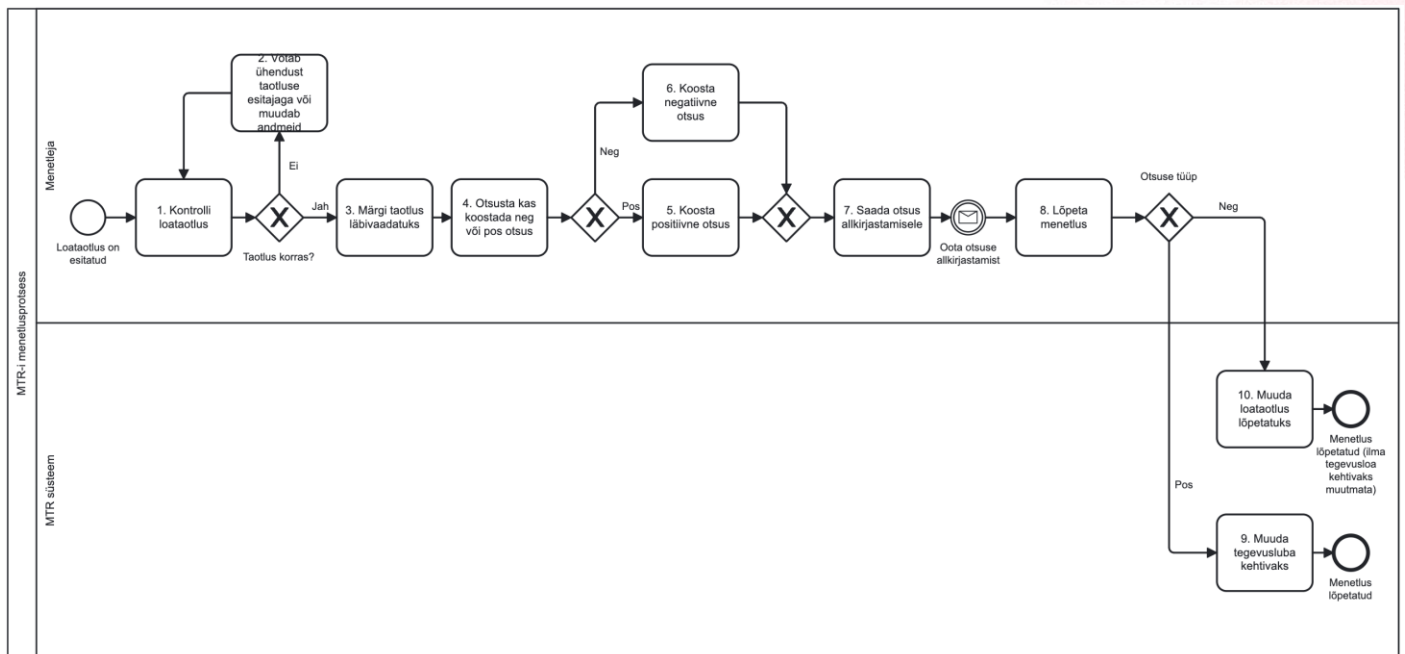
Lisa dokument

Joonis 2. Vana MTRi tegevusloa taotluse menetleja kuva

Vastavalt loataotluse staatusele kuvatakse nupud, millele vajutades käivitub vastav funktsionaalsus ning mööda protsessi edasi liikudes jõutakse lõpuks menetlusprotsessi lõppu (negatiivne või positiivne otsus). Osade loataotluste puhul on vajalik ka allkirjastamine (nt Merenduse loataotlus).

Kuna erinevaid loataotlusi on väga palju, siis on oluline märkida, et osade loataotluste menetlusprotsess võib sisaldada veel valdkonnaspetsiifilisi samme.

Järgnev joonis annab **lihtsustatud** ülevaate MTRi menetlusprotsessist Merenduse valdkonna näitel (näide algab sammust, kus taotlus on juba esitatud, sest menetlustoote kontekstis ei ole taotluse esitamise protsess oluline, kuna see on klienditoote osa):



Joonis 3. MTRi menetlusprotsess

Loataotluse esitamisel kontrollib menetleja loataotluse andmed üle (samm 1), mille järel ta kas märgib taotluse läbivaadatuks, juhul kui see oli korras (samm 3) või võtab ühendust taotluse esitajaga või muudab taotluse andmeid, kui see korras ei olnud (samm 2). Pärast taotluse läbivaatamist valib menetleja, kas koostada positiivne või negatiivne otsus, ning vormistab otsuse vastavalt tehtud valikule (sammud 5 või 6).

Juhul kui menetlusprotsess vajab ka allkirjastamist, saab menetleja otsuse allkirjastamisele (samm 7) ning peale allkirjastamist saab menetleja menetluse lõpetada (samm 8). Kui koostati negatiivne otsus, märgitakse loataotluse staatuseks, et menetlus on lõpetatud (samm 10). Juhul kui koostati positiivne otsus, muudab süsteem tegevusloa kehtivaks ning siis menetlus loetakse lõppenuks (samm 9).

Menetluse käigus on igal hetkel võimalik teha veel järgmiseid tegevusi:

- Lisada menetleja, kellel tekib õigus antud loataotlust menetleda;
- Võtta menetluse samm tagasi juhul kui varasemates sammudes ilmneb vigu;
- Pikendada menetluse tähtaega;
- Muuta loataotluse andmeid;
- Lisada dokumente;
- Märkida loataotlus tagasi võetuks, juhul kui taotluse esitaja selleks soovi avaldab;
- Kontrollida taotluse esitaja ning ettevõtte kohta nende võlgu või karistusi.

Kui otsus on juba koostatud, on võimalik MTRis lisada uus otsus ning otsuse dokumendi teksti muuta.

Tegevuslubade peal, mille menetlus on lõpetatud ning mis on kehtivad, saab menetleja teha järgmisi tegevusi:

- Muuta tegevusluba;
- Parandada registriandmeid;
- Esitada kahtlus registriandmete õigsuses;
- Esitada täielik või ajutine loobumisteade;
- Lisada ametkondliku kasutamise märget;
- Peatada või tunnistada kehtetuks;

- Kontrollida ettevõtte karistusi või võlgasid.

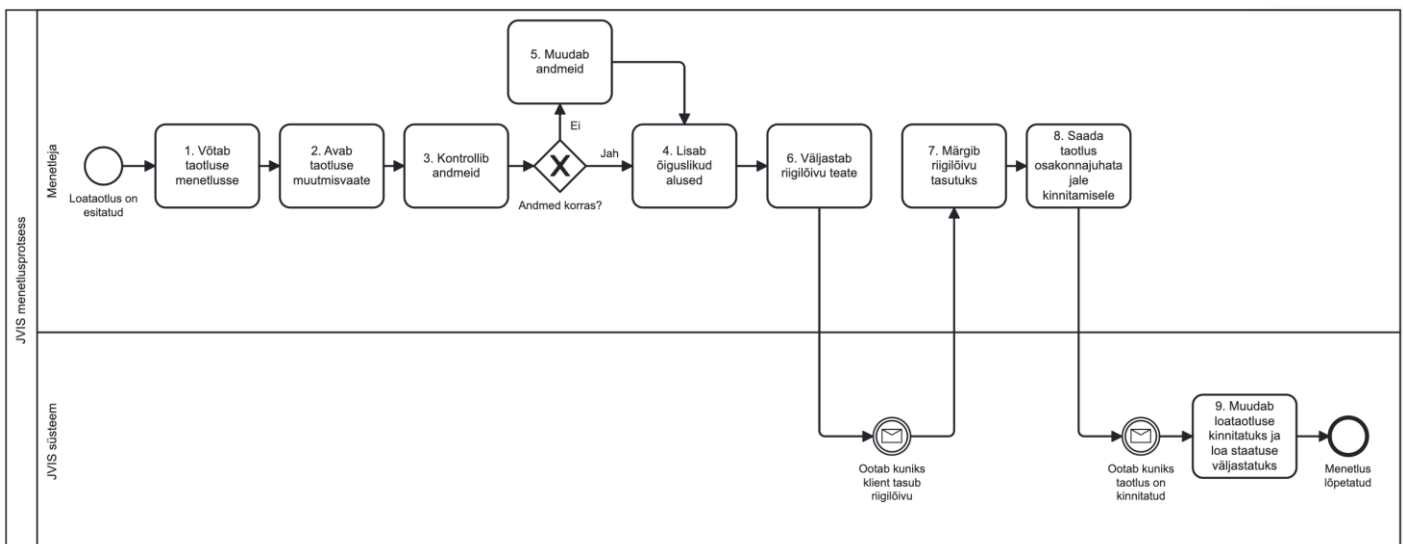
Näide 2. JVIS

Järelevalve Infosüsteem (JVIS) on samuti TTJA poolt hallatav infosüsteem, kus hallatakse osasid TTJA pädevusse kuuluvaid tegevuslubasid (mis ei ole MTRis) ning teostatakse järelevalvemenetlusi, näiteks sideload.

Järelevalvemenetluste puhul on võimalik teha järgmisi tegevusi:

- Sisestada protokoll;
- Sisestada ettekirjutusi;
- Teostada visiite;
- Sisestada akte;
- Sisestada kontroll-lehti;
- Teostada ekspertiise.

Loataotluste puhul on erinevatel taotlustel erinev protsess nende menetlemiseks. Ühe loataotluse menetlusprotsessi ilmutamiseks toome näite sagedusloa väljastamisest (vastavalt hankesisendis kirjeldatule):



Joonis 4. JVIS menetlusprotsess

Kui loataotlus on esitatud, võtab menetleja selle menetlusse (samm 1). Seejärel avab ta muutmiskaarte (samm 2) ning kontrollib andmeid (samm 3). Juhul kui selleks on vajadus, saab ta andmeid muuta (samm 5) ning kui kõik on korras, lisab ta ka õiguslikud alused (samm 4).

Seejärel vastavalt andmetele väljastab menetleja riigilõivu teate (samm 6) ja süsteem jääb ootama kuniks klient tasub riigilõivu. Kui raamatupidajatelt tuleb info, et riigilõiv on tasutud, märgib menetleja selle makstuks (samm 7). Siis menetleja saadab taotluse osakonnajuhatajale kinnitamisele (samm 8). Kui osakonnajuhataja kinnitab taotluse, muudab süsteem loataotluse kinnitatuks ning sideloa staatus muutub väljastatavaks (samm 9). Menetlusprotsess on sellega lõppenud.

Igal hetkel saab menetleja:

- Lõpetada menetluse nii, et sideluba ei väljastata;
- Lisada taotlusele faile;
- Saata taotlus teisele menetlejale menetlemiseks.

Kui sideluba on väljastatud, on võimalik teha sellega edasisi toiminguid, ehk:

- Tühistada;
- Esitada muutmise taotlust;
- Pikendada.

Näide 3. NBA

NBA rakendust kasutavad nii TTJA ametnikud kui ka numbriloo omanikud. Rakendus jaguneb omakorda kaheks mooduliks: Numbri Broneerimise Andmekogu (NBA) ja Numbri Broneerimise Andmekogu süsteemi Numbriliikuvuse moodul (NBA NL).

NBA-s on võimalik seadistada süsteemiparameetreid, teostada numbrite ning nende lubadega seotud päringuid ja neid hallata. Rakenduses NBA NL on võimalik esitada numbriliikuvuse taotlusi ning neid menetleda. Samuti on seal võimalik teostada numbriliikuvusega seotud päringuid, hallata sõnumite liikumist ning nende kanaleid.

Alates 2022. aastast kasutatakse numbrilubade menetlemiseks JVIS-i, kus on loodud eraldi moodul selle jaoks.

3.2.2 Menetluse äriprotsessi nõuded ning UI/UX põhimõtete rakendamine

TARVIK jaguneb funktsionaalselt kahte suuremasse kategooriasse: **klienditoode** mis teenindab loataotlejaid, ja **menetlustoode**, mis teenindab nii TTJA sisemisi kui ka teiste asutuste (Näiteks Muinsuskaitseamet, erinevad KOV-id, Transpordiamet jne) kasutajaid - menetlejaid. Käesolev visioon keskendub **menetlustoote kasutajaliidese arendusele**, mis peab toetama TTJA ning teiste asutuste ametnike igapäevatööd loataotluste, teavituste ja otsuste menetlemisel.

Eesmärgi saavutamiseks pöörame erilist tähelepanu muuhulgas ka sellele, et:

- pakkuda kasutajatele teenuste tarbimiseks **tänapäevaseid ja kasutajakeskseid lahendusi**;
- anda ülevaadet **menetlusprotsesside seisust ja pudelikaeltest**;
- leida kohti, kus **siseprotsesse optimeerida ja automatiseerida**.

Kasutajakesksete e-teenuste loomist alustati **loateenustest**, kus analüüsiti loateenuste klienditeekondi, tuvastati klientide peamised takistused ja raskuskohad ning pakuti välja lahendusi, mis muudavad teenuste kasutamise mugavamaks ja intuitiivsemaks. Lisaks visualiseeriti kasutajaliidest olulisemate põhiteekondade lõikes ning pandi kirja olulisemad disainiotsuste lähtekohad. Samuti on varasemate projektide raames alustatud juba mitmete tugiteenuste arendamisega, näiteks X-teede päringud (Ärregister, Rahvastikuregister), kontaktandmete haldus, teavituste saatmine, logide haldamine jpm.

Siit edasi on loogiline samm minna loateenuse TTJA poolse vaate ehk **menetlusprotsessi kasutajakogemuse ja kasutajaliidese arenduse** juurde, et loataotleja vaatest tekiks ühtne ja sujuv teenuse kogemise tervik. TTJA menetlustoote puhul on võtmeküsimuseks **kasutajaliidese funktsionaalne sobivus tööprotsessidega**, mis erinevad oluliselt kliendivaates kasutatavatest.

Nii kliendi- kui menetlustootes olulised kasutatavuse aspektid	Menetlustoote jaoks lisaks olulised kasutatavuse aspektid
--	---

Intuiitiivne, funktsionaalne ja ühildub olemasolevate infosüsteemidega.	Toetab tööprotsesside loogikat ning suurt andmemahutu
Vastab ligipääsetavuse nõuetele nii disainis kui abivahendite mõistes (nt ekraanilugerid, klaviatuurinavigatsioon).	Võimaldab kiiresti liikuda toimingute vahel (nt lubade kooskõlastamine, otsuste koostamine).
Annab piisava ülevaate menetluse staatusest.	Annab lisaks menetluse staatusele ülevaate ka töövoos protsessist.
	Võimaldab efektiivset teabehaldust ja otsuste tegemist.
	Toetab tõrgeteta koostööd teiste süsteemidega üleminekuperioodil ning võimaldab sujuvat üleminekut uuele lahendusele.

Hanke tehnilisele kirjeldusele ja visiooni koostamise juhendile tuginedes on alljärgnevalt toodud pakkuja põhjalikum nägemus menetlustoote kasutajaliidese loomise tegevuskavast. Töö aluseks on 2024. aastal Trinidad Wisemani poolt läbi viidud TTJA klienditoote UI/UX analüüs ja disainijuhised. **Menetlustoote kasutajaliidese disainimisel arvestame loodud disainijuhiste järjepidevusega, kuid kohandame need vastavalt menetlejate töövajadustele.**

Üleminekuperioodil on oluline, et menetlejal ei katkeks töövoog erinevate süsteemide vahel liikudes. Selle tagamiseks:

- Märgistame liideses tegevused, mis viivad teise süsteemi (nt JVIS, MTR, NBA vahel navigeerimine).
- Kuvame viite süsteemile, kust andmed pärinevad.
- Toetame tööprotsessi katkestuspunktide märkimist, et kasutaja saaks pooleli jäänud tegevust hiljem jätkata.
- Dokumenteerime kõik riskasutusest tulenevad piirangud ja pakume välja üleminekustenaariumid koos riskide maandamisega.
- Võimalusel integreerime vajalikud andmed TARVIKusse ilma kasutajat mujale suunamata.
- Refaktoreerime olemasolevad TARVIKu menetleja kavad (nt Registripäevik), et need moodustaks terviku ülejäänud TARVIKu kuvadega ja oleks kooskõlas ülejäänud menetlustootega.

Pakume menetlustoote kasutajaliidese arendamisel järgmisi **uuenduslikke ja praktiliselt** teostatavaid lahendusi:

- **Selge, väljaarendatud rollipõhine töövaade**, mis võimaldab menetlejal näha ainult neid toiminguid, mis on seotud tema vastutusalaga (nt Registripäevik või Järelevalvaja).
- **Personaalne töölaud**, kus menetlejale või teistele rollidele kuvatakse vastavalt personaalsetele eelistustele ja seadistustele eespool need toimingud või loatüübid, millega ta tihedamini kokku puutub. See võimaldab ka tiimide üleselt saada head ülevaadet tööde seisust ning annab võimaluse töid planeerida.
- **Lisaväärtusena UX-kirjutaja kaasamine**, et rakenduse abitekstid ja sisutekstid toetaks kasutajat tegevuses ega põhjustaks arusaamatusi.
- **Ligipääsetavuse standardile EN 301 549 vastavuse tagamine** läbi spetsialisti kaasamise. [EN 301 549](#) sisaldab endas nii rahvusvahelise standardi [WCAG 2.1 AA](#) taseme nõudeid kui ka täiendavaid lisanõudeid.

- **Täisdisainisüsteemi kasutuselevõtt**, sh front-end komponentide kogum, et tagada kasutajakogemuse järjepidevus ja skaleeritavus ning hilisem ressursisääst front-end töödes.
- **Menetlusprotsesside ühtlustamine**, et lihtsustada ja kiirendada menetlejate igapäevatööd.
- **Jätkusuutlik komponentide taaskasutus**, mis võimaldab hilisemates arendusfaasides uusi teenuseid lihtsasti sama kasutajaliidese loogika alusel liidestada. Samuti paneb komponentide taaskasutus aluse erinevate menetlusprotsesside ühtlustamisele.
- **Otsustusmootori kasutuselevõtt**, mis võimaldab vastavalt sisestatud reeglitele ja andmetele menetluse osas teha automaatseid otsuseid süsteemi poolt, mis lihtsustab menetlejate tööd.
- **Menetlusprotsessi sammude kuvamine**, et menetleja saaks aru kus ta asub protsessis ning kui suur osa menetlusest on veel läbimata.
- **Süsteemide vaheline integratsioon**, sh
 - TTJA vanade süsteemide ja TARVIKu vahel, et menetlejad ei peaks andmeid sisestama mitu korda, vähendades vea tekkimise ohtu
 - TARVIKu ja väliste süsteemide vahel, et optimeerida ja automatiseerida menetlusprotsessi. Näiteks võimaldaks see kontrollida riigilõivude laekumist automaatselt, et menetlusprotsessi järgmisesse sammu liikuda ilma menetleja poolt andmeid käsitsi sisestamata (lisaks veel teavituste automaatne saatmine riigilõivu laekumisel, NANDOga integratsioon jne)
- **AI integreerimine TARVIKusse**, mis võimaldaks AI poolt teostada automaatseid menetlustooteid ja aidata vastata kasutaja küsimustele. Täpsem AI kasutamise võimalikkus on kirjeldatud käesoleva dokumendi peatükis [„Tehisaru kasutamine“](#).

3.2.3 Võimalikud lahknevused ja ebakõlad võrreldes eelnevalt teostatud kasutajakogemuse analüüsiga

Trinidad Wiseman viis 2024. aastal läbi TTJA klienditoote kasutajakogemuse analüüsi, mille põhjal koostati ka disainijuhised. Menetlustoote arenduses kasutame seda analüüsi alusena, kuid arvestame menetluskeskkonna eriomadustega.

Kliendivaate puhul on rõhk teenuse leidmisel, juhendamisel ja veateadete mõistmisel. Menetleja töös on esikohal töökiirus, töövoos selgus ja süsteemne liidese struktuur. Menetlustoote liides peab toetama süstemaatilist otsustamist, võimaldama kiiret navigeerimist ja olema kooskõlas tööprotsessi etappidega. Võimalikud lahknevused ja erinevused, mis võivad ilmneda ja millega arvestame:

- **Menetlustusükli struktuurne erinevus:** kliendivaate ülesehitus on lineaarne, menetluses võib paralleelselt töös olla mitu protsessi, võidakse eelistada andmete kuvamist tabelitena, kaardivaadetena vms.
- **Kasutuslugude piiratus loataotluste UX analüüsis:** teostatud kasutajakogemuse analüüsis käsitleti vaid piiratud osa loataotluste tüüpidest ja seda eeskätt lõppkasutaja vaatest. Menetlusprotsessi analüüs võib juurde tuua vajaduse võtta kasutusele täiesti uusi disainikomponente.
- **Paindlikkuse vajadus erinevate interaktsioonide kasutamisel:** menetlejate vaadetes võib olla rohkem tingimuslikke välju ja erijuhtumeid, samuti võivad menetlejad eelistada vormiväljade kohandamisvõimalust spetsiifilisele menetluse liigile vastavaks.
- **Terminoloogia ja pisitekstide erinevus:** Kliendivaate abitekstid on rohkem selgitavad, menetlejatele kuvatakse aga sisekasutajatele suunatud täpsemat ja juriidilisemat sõnakasutust.

Vaatame kriitilise pilguga üle olemasolevad disainimustrid ning vajadusel täiendame või kohandame neid, et need sobituks menetlejate tööviisiga, arvestades ligipääsetavuse nõudeid ja olemasoleva disainisüsteemi disainikomponentide taaskasutust (Trinidad Wisemani loodud prototüübis kasutati Veera disainisüsteemi).

3.2.4 Menetleja kasutajaliidese arenduse tegevuskava - analüüs, disain ning arendus

Kasutajasõbralikuma menetlusprotsessi saavutamiseks saame lahendusi otsida erinevatest suundadest:

- lõppkasutajate loataotluse esitamise kitsaskohtadele lahenduste leidmine;
- menetlejate tööprotsesside ja kasutajaliidese kitsaskohtade parendused.

Loataotluse kasutajauuringu tulemused ja murekohtade prioriseerimine TTJA töögrupis tõid välja, et väga suur osa kasutajakogemuse puudustest tekib ebaselgest kliendiliidest, mis omakorda tekitab palju lisatööd menetlejale ja TTJA kasutajatoele.

Näiteks puudub kasutajatel selge arusaam menetlustähtaegadest ning nad ei saa piisavat tagasisidet taotluse menetlemise staatuse kohta. Seda annaks parandada näiteks selgemate menetlustähtaegade kommunikeerimisega taotluse esitamisel, automaatsete teavituste lisamisega taotluse staatuse muutumisel ja lihtsa staatuse jälgimise võimaluse loomisega kasutajaliideses. Seega omab juba **ainuüksi lõppkasutaja kogemuse parandamine ja ühtlustamine automaatselt suurt mõju ka menetleja töö efektiivsemaks muutmisele** - väheneks koormus kasutajate läbiaitamisel loataotluse protsessist ja tema infoväljas hoidmisel. Kui lõppkasutaja saab asjakohaseid teateid menetluse käigust ja taotlusvormi vigadest operatiivsetesse kanalitesse (e-mail, SMS), aitab see vähendada menetlejate ja klienditoe töökoormust info jagamisel.

Menetleja poolelt vaadatuna ilmneb nii MTR kui JVIS dokumentatsioonidest, et täna teeb menetleja väga palju korduvaid tegevusi käsitööna: nt sisestab riigilõivu laekumised (mille kohta on ka raamatupidaja eelnevalt teinud käsitööd, saataks teatud perioodi laekumised e-kirja teel menetlejale), teeb sissekande rahvusvahelisse keskkonda NANDO, et kooskõlastada teiste riikidega antud luba; teeb otsuse ühes süsteemis ja kopeerib selle käsitsi teise jne. Kõiki neid **täna käitisi tehtavaid tegevusi on võimalik automatiseerida läbi vajalike liidestuste**.

Taotluse ja menetluse ühte keskkonda toomine aitab kaasa tervikliku kasutajakogemuse loomisele nii lõppkliendi kui menetleja vaatest. Seda kõike arvesse võttes, **soovitame tegevuslubade üle toomisel TARVIKusse alustada enam esitatavate loataotlustega, koos selle menetlusprotsessiga üheaegselt**.

Menetlustoote kasutajaliidese arendus toimub etapiviisiliselt, lähtudes agiilsest arendusmudelist ja tihedast koostööst tellijaga. Üleminekuperiood eeldab, et uus menetlusliides töötab paralleelselt olemasolevate süsteemidega.

Menetlustoote kasutajaliidese arendus jaguneb järgnevateks etappideks. **Käesolevad tegevused on välja toodud ühe loeteluna, kuigi reaalsuses teostatakse neid tegevusi iteratiivselt, ehk Dual Track Scrumi mõistes on mitu paralleelselt toimuvat teekonda (Tracki), kus käiakse läbi sammud 1-8:**

1. **Olemasolevate disainijuhiste ja menetluskeskkonna analüüs.** Võrdleme klienditoote disainimustreid, MTR, JVIS kasutajaliidese funktsioone, et selgitada välja ülekantavad ja uued komponendid.

2. **Menetlejate kasutajakogemuse ja vajaduste kaardistamine.** Viime läbi intervjuud menetlejatega, mõistmaks, millised on tänase menetlusprotsessi kitsaskohad.
3. **AS-IS menetlusprotsessi kasutajateekonna kaardistamine.** Visualiseerime menetleja kui kasutaja tänased töövood ja teekonnad ning kokkupuutepunktid lõppkasutajaga.
4. **Nõuete ja skoobi täpsustamine.** Vastavalt AS-IS analüüsile ja eelmistes faasides tuvastatud valupunktidele kirjeldatakse koostöös Tellijaga ärinõuded ning selgitatakse välja skoopi kuuluvad ning sellest välja jääv funktsionaalsus. Lisaks tehakse antud sammus juriidiline analüüs, et veenduda, et loataotluste menetlused vastavad seadustele
5. **TO-BE kasutajateekonna ja navigatsiooniloogika visualiseerimine.** Visualiseerime, milliseid samme menetleja uues süsteemis ideaalis läbib, võttes arvesse ka võimalikke ühtlustamis- ja optimeerimiskohti.
6. **TO-BE lahenduse disain.** Looime menetleja vaadete prototüübid, testime neid kasutajatega, et viimased saaksid kindlust, et arendama asutakse just seda, mida oleme kokku leppinud, viime sisse muudatused, kinnitame ja disainime lõplikud vaated, kasutades täisdisainisüsteemi (nt TEDI ja TARVIK erikomponentide kombinatsiooni) ning järgides EL ligipäätavuse standardit EN 301 549. [EN 301 549](#) sisaldab endas nii rahvusvahelise standardi [WCAG 2.1 AA](#) taseme nõudeid kui ka täiendavaid lisanõudeid.
7. **Arendustööde teostamine.** Vastavalt teostatud analüüsile ja kokkulepitud skoobile, teostatakse arendustööd.
8. **Arendustööde testimine.** Valminud arendustööd testitakse testija või analüütiku poolt, et veenduda arenduse vastavust vastuvõtutingimustele.

Järgnevalt on kirjeldatud iga tegevus detailsemalt, selle eeldatav mõju ning tähelepanekud üleminekuperioodiks. Lisaks Dual Track Scrumi sammudele 1-8 toome olulise arendusprotsessi lõppsammuna välja ka arenduste tarnimise detailsema kirjelduse.

1. Olemasolevate disainijuhiste ja menetluskeskkonna analüüs

Tegevuse sisu	Koondame varasemad kasutajauuringu analüüsitulemused, UX disainiotsused ja teeme võrdluse JVIS kasutajaliidese ning uue disainisüsteemi komponentide vahel. Märgistame ära disainilised ja sisulised ebakõlad, mis võivad mõjutada töövoogu. Analüüs sisaldab ka menetleja töökoormuse ja tööprotsessi kaardistust (rutiinsed tegevused, pudelikaelad, korduvad tegevused, käsitsi töö jne).
Eeldatav mõju	Annab disainitöödele selgema fookuse, aitab vähendada hilisemates faasides UX/UI tööde mahtu ning ja loob järjepideva kasutuskogemuse.
Üleminekuperioodi tähelepanekud	Teadvustame, et osa menetlejaid võib paralleelselt kasutada nii uut kui vana süsteemi – disain peab toetama kahe süsteemi vahelist sarnast navigeerimisloogikat. Samas tagame uue, optimeeritud süsteemi arenduse kursil püsimist eelanalüüsi käigus loodava TO-BE lahenduse skoobis hoidmisega, et vana süsteemi loogika ei saaks takistuseks uue loomisel.
Ootused Tellijale	• Annab ligipääsu tööprotsesse puudutavale infole ning töökorraldusele • Annab tagasisidet ja vastab küsimustele

2. Menetlejate kasutajakogemuse ja vajaduste kaardistamine

Tegevuse sisu	Selle tegevuse eesmärk on mõista, milline struktuur ja liidese käitumine toetab menetlejaid kõige enam. Viime läbi intervjuud menetlejatega, mille peamine eesmärk on murekohtade ja eelistuste kaardistamine protsessi erinevates lõikudes, seejuures markeerides ka need kohad, kus menetlejatele tekib lisategevusi, kuna lõppkasutaja liides ei ole mugav. Lisaks palume menetlejalatel kommenteerida portaali kasutajakogemust erinevatest aspektidest lähtuvalt: funktsionaalsus, disain, tekstid. Eestis täna veel pigem uudse lähenemisena kaasame UX-kirjutaja, et testida sõnastust ning navigeerimise loogikat. Analüüsime kasutaja käitumist töövoo protsessi lõikes, et tuvastada tööprotsesside ebaefektiivsust ja optimeerimiskohti, mida annab lahendada erinevate tehniliste lahenduste abil: süsteemide vaheline integratsioon, automatiseerimine, parem teavituse- ja meeldetuletuse loogika, AI või protsessimootori kasutusele võtmine. Kaasame intervjuudesse võimalikult erineva tausta ja kogemusega töötajad, et luua lahendus, mis oleks intuiitiivne ka ilma pikki juhendeid lugemata. Intervjuud toimuvad individuaalselt, kestavad kuni 1 tund ning salvestatakse helis ja pildis, et analüüsida ka tehnilise lahenduse kasutatavust.
Eeldatav mõju	Menetlejate vaatenurk toob välja olulisi faktipõhiseid kitsaskohti, mis on sisendiks funktsionaalsusele ja kujundusele, võimaldades meil lahendada päris kasutajate päris probleeme. Menetlejate kaasamine muudatuste protsessi ja arendustesse lihtsustab hilisemat muudatuste elluviimist.
Üleminekuperioodi tähelepanekud	Arvestame, et vastavalt hanke alusdokumentidele migreeritakse menetlusprotsesse järk-järgult. Selgitame koostöös menetlejatega, millised on nende eelistused valikul ja loogika protsesside üleminekuperioodiks.
Ootused Tellijale	• Annab tagasisidet intervjuu kavale • Edastab vajalikud töötajate kontaktid

3. AS-IS menetlusprotsessi kasutajatekonna kaardistamine

Tegevuse sisu	Enne uue kasutajaliidese prototüüpimist kaardistame olemasolevad tööprotsessid, mida TTJA menetlejad igapäevaselt läbivad. Kasutame kogutud andmeid intervjuudest, aga ka varasematest kasutajauuringutest ja olemasolevaid protsessivooe, et visualiseerida kuidas menetleja liigub läbi erinevate tegevuste nagu taotluse vastuvõtmine, sisestamine, täiendamine, otsuse koostamine, edastamine jm. Loomes visuaalsed kasutajatekonnad (Tellijale sobivas keskkonnas), mille kaudu on näha tegevuste järjestus, tööriistad, infoväljad ja katkestuskohad. Lisaks toome teekonna pildile ka lõppkasutaja: millises punktis ta täna abi vajab, kus tahab olla või peaks olema infoväljas, millist täpset infot peaks loataotluse vigade kohta saama jms. Tegevuse viime läbi pakkuja sisemise töötoana ning teekondi valideerime ja kinnitame koos Tellijaga.
Eeldatav mõju	Annab tervikliku ja tõenduspõhise aluse uue liidese disainiks, tuues pildile mitmeid erinevaid tänaseid tööpraktikaid ning välistades oletuste põhjal otsustamise. Võimaldab paremini mõista, kuidas menetlejad eri

	osakondades süsteemi kasutavad ning millised osad vajavad standardiseerimist või täiustamist.
Üleminekuperioodi tähelepanekud	AS-IS kaardistus aitab selgelt eristada esialgu olemasolevatesse süsteemidesse jäävad ning uude menelustootesse viidavad tegevused. Märhistame need menetleja teekonnal eraldi, et aidata disainis ja kasutusjuhistes rõhutada süsteemide piire ja sujuvaid üleminekuid. Kaardistus saab olla ka koolitusmaterjali ja muudatuste kommunikatsiooni alus.
Ootused Tellijale	Annab tagasisidet AS-IS menetleja kasutajateekonnale

4. Nõuete ja skoobi täpsustamine

Tegevuse sisu	Kui AS-IS protsessid ja valukohad on kaardistatud, täpsustatakse koostöös Tellijaga (äri)nõuded TARVIKusse implementeeritavale menelustootele ning teostatakse süsteemi detailanalüüs. Nõuete täpsustamiseks viiakse läbi analüüsikoosolekuid kliendiga. Lisaks tehakse koostöös Tellijaga juriidiline analüüs iga loataotluse kohta veendumaks, et nende esitamine ning menelustprotsess vastavad Eesti Vabariigi seadustele. Antud sammus teostatakse ka süsteemianalüüs, mille käigus täpsustatakse vajatavad integratsioonid teiste süsteemidega, konfiguratsioonid, mikroteenused ning nende andmebaasid veendumaks, et menelusttoote nii äriine, kui ka tehiline disain moodustab ühtse terviku ning toimib vastavalt nõuetele. Kogudes vajalikku sisendit, dokumenteerib analüütik kogu info, sünteesib seda, teeb järeldusi ning kogutud infost lähtuvalt pakub välja lahendusi, mis kooskõlastatakse Tellijaga. Detailanalüüs käsitleb lisaks asjaolu, et paralleelselt peavad jääma töötama mitmed süsteemid ning ka vajadusel infot vahetama (nt JVIS-i ja TARVIKu vaheline integratsioon). Detailanalüüsi käigus selgitatakse välja ja dokumenteeritakse: • Nõuete täpsed sõnastused • Arenduse äriine vajadus • Sõnavara ja mõistete selgitused • Protsesside kaardistused • Süsteemi integratsioonide kirjeldused teiste (väliste) süsteemidega • Diagrammid, mis analüüsi kontekstis on olulised ja pakuvad väärtust (näiteks UML jadadiagrammid, BPMN protsessidiagrammid jne) • Vajalikud seadistused ja konfiguratsioonid • Süsteemi spetsifikatsioonid, sh ○ Andmestruktuur/andmebaasi disain ○ Mikroteenused Selleks et tagada arusaadav ja ühene arusaam projekti skoobist kõikidele osapooltele ning ennetada võimalikke arusaamatusi hilisemates etappides, dokumenteeritakse nii implementeeritavad nõuded kui ka need nõuded, mille rakendamist praeguses etapis ei kavandata, koos vastavate põhjendustega.
----------------------	--

	Pakkuja varasem kogemus näitab, et selline lähenemine aitab kõikidel osapooltel mõista oodatavat tulemust ning välistab olukorrad, kus nõudeid tõlgendatakse erinevalt, mis võiks põhjustada ebaselgust vastuvõtutestide käigus. Tulevastes iteratsioonides on võimalik skoobi ülevaatamine vastavalt vajadusele, mis toetab agiilset lähenemist.. Loodud dokumentatsioon kooskõlastatakse Tellijaga, mille järel on võimalik luua konkreetsed arenduspiletid Jira keskkonda.
Eeldatav mõju	Täpselt sõnastatud nõuded ja protsessid annavad mõlemale osapoolle kindluse, et loodava süsteemi nõuetest ja skoobist on üheselt aru saadud. Dokumentatsiooni kinnitustega annab Tellija nõusoleku antud arendustöödega alustamiseks, mis on ühtlasi arenduse eelduseks.
Üleminekuperioodi tähelepanekud	Nõute täpsustamisel ning detailanalüüsi käigus arvestatakse asjaoluga, et paralleelselt peavad jääma tööle mitmed süsteemid. Dokumenteeritakse nende süsteemide integratsioonid/üleminekud ning ka kitsaskohad, millega peab edaspidistes töövoogudes arvestama (kui neid on).
Ootused Tellijale	• Osaleb analüüsikoosolekutel • On kättesaadav küsimuste tekkimisel • Nimetab tooteomaniku/vastutaja (TTJA poolt), kes lahkarvamuste korral lõppotsuseid teeb • Vaatab üle ja annab tagasisidet jooksvalt detailanalüüsi tulemitele

5. TO-BE kasutajateekonna ja navigatsiooniloogika visualiseerimine

Tegevuse sisu	TO-BE kasutajateekonna ja navigeerimisloogika visualiseerimise eesmärk on enne detailsete vaadete prototüüpimist kujundada sihipärane ja kasutajakeskne struktuur menetlustoote uuele liidesele. Selles etapis visualiseerime, milline on menetlaja jaoks ideaalne kasutajateekond, st milliseid samme menetleja uues süsteemis läbib, võttes arvesse ka võimalikke optimeerimiskohti (nt infosüsteemide paremat omavahelist suhtlust, eeldefineeritud otsustuspuude või tehisarut kasutamist automaatsete otsuste tekitamiseks, osakondade vahelise tööjaotuse optimeerimist, vormiväljade maksimaalset standardiseerimist vms.). Visualiseerime need teekonnad (Tellijale sobivas keskkonnas) koos seotud vaadete tüüpidega (nt otsinguvaade, loetelu, vormivaade), määrates ära iga vaate rolli, sisendi ja järgmise loogilise sammu. Julgustades menetlejaid mõtlema oma tööprotsessidele kõrvalseisjana, loovalt ja “kastist välja” lähenemisega, viime läbi koosloome tööubasid, mille käigus kasutame disainmõtleamise meetodikaid lahenduste ideestamiseks, näiteks: • Hull 8, kus piiratud aja, 8 minuti jooksul, otsitakse lahendusi menetlusprotsessis ettetulevale konkreetsele probleemile. Töötoas osalejad kirjeldavad individuaalselt 8 võimalust, hiljem hääletatakse neist parim või kasutatakse mõju-teostatavuse maatriksit parimate lahenduse väljasõelumiseks • Jah, ja... Meetodika, kus töötoas osalejad peavad kolleegi lahendusideele leidma asjakohaseid täiendusi, avama võimalust veelgi enam (jah, hea mõte, ja lisaks veel...) vastukaaluks piiravale suhtumisele (“hea mõte, aga...”).
----------------------	--

	Visualiseerime nii kasutajateekondadena, kui ka BPMN protsessijoonistena, sest kasutajateekonnad ilmestavad inimese poolt ning BPMN protsessijoonised aitavad tuua tehnilisemat vaadet uutele teekondadele/protsessidele, mis hõlmavad ka suhtlust väliste süsteemidega. TO-BE navigatsiooniloogika hõlmab ka ülevaadet vasakmenüü struktuurist, töölaudadest, filtritest, <i>breadcrumb</i> loogikast jms. Kujundame selle loogikaga, mis toetab nii kogenud kasutajate kiirtegevusi kui juhatab vähemkogenud kasutajaid samm-sammult. Viime selle tegevuse läbi koosloome töötoana koos Tellijaga ning vormistame TO-BE kasutajateekonna ja navigatsiooniloogika lõplikule kujule.
Eeldatav mõju	TO-BE kaardistus loob tugeva aluse hilisemaks prototüüpimiseks – vaated lähtuvad protsessidest, mitte ainult tehnilisest ülesehitusest. Tagab, et liides on kasutaja jaoks loogiline ja suunav ning toetab töövoogi kiiret läbimist ilma ekslemiseta. Kogu töövoogi, sh sisemiste protsesside visualiseerimine aitab ühtlustada arusaama kogu meeskonnas (TTJA, disainerid, arendajad, menetlejad), milline on soovitud lõpptulemus.
Üleminekuperioodi tähelepanekud	TO-BE teekonnad kaardistatakse nii, et oleks võimalik paralleelne kasutus olemasoleva süsteemiga. Näiteks märgime need osad protsessist, mida uus süsteem veel ei kata ja mis viivad tagasi JVIS/MTR vaatesse. Toetame võimalust, et uude süsteemi liidetakse järk-järgult rohkem protsesse – navigeerimisstruktuur on selleks modulaarne ja skaleeritav.
Ootused Tellijale	- Osaleb projektimeeskonnaga aktiivselt koosloome töötoas, kaasates erinevat majasisest ekspertiisi ja kogemust. - Annab tagasisidet TO-BE menetleja kasutajateekonnale ja kasutajaliidese navigatsiooniloogikale.

6. TO-BE lahenduse disain

Tegevuse sisu	Arendame välja klikitava prototüübi menetlustoote prioriteetsetest vaadetest, lähtudes TO-BE kasutajateekonnast ja navigatsiooniskeemist. Prototüüp sisaldab kõiki olulisi tööprotsessi toetavaid komponente ja elemente: näiteks taotluste loend, menetluse detailvaade, otsuse koostamise vorm ja töölaud. Samuti mõtleme läbi ja prototüübime lõppkasutajale olulise info kuvamise viisi menetlusprotsessi konkreetse etapi staatusest. Prototüüp luuakse kahes astmes: <i>Low-fidelity wireframe</i> – keskendub funktsionaalsele struktuurile, infopaigutusele ja navigatsiooniloogikale. Võttes kasutusele Veera disainisüsteemi kohe projekti alguses, saame juba ka selles faasis kuvada “nagu päris” vaateid. <i>High-fidelity</i> prototüüp – visuaalselt kujundatud versioon, kasutades täisdisainisüsteemi komponente (nt TEDI ja TARVIK erikomponentide kogum või kokkuleppel mõnda muud lahendust). Vajadusel loome uued taaskasutatavad komponendid.
----------------------	--

	Kaasame UX-kirjutaja, kes töötab paralleelselt disaineriga, et vormiväljade nimed, tegevusnupud ja süsteemsed sõnumid oleksid kohe prototüübis kasutatavuse seisukohalt sobivad ning igas vaates läbivalt ühtsed (nt ei kasutataks läbisegi termineid “Kinnita” või “Salvesta”, “Ootel” või “Avatud” vms). Prototüüpi testitakse menetlejatega individuaalsete kasutatavuse testide käigus, kasutades „valjusti mõtlemise“ meetodit. Testitavad täidavad neile tuttavaid tööülesandeid ning me jälgime nende arusaamist, eksimusi ja ootusi. Tagasiside põhjal täiustame prototüüpi.
Eeldatav mõju	Klikitav prototüüp võimaldab TTJA meeskonnal testida süsteemi loogikat ja disaini enne arendusressursside kasutamist. Parandused ja täpsustused on võimalik teha varases faasis, vältides hilisemaid kulukaid muudatusi. Kasutajatestide tulemusel saame faktipõhise kinnituse, et süsteem on arusaadav ja toetab tööprotsesse.
Üleminekuperioodi tähelepanekud	Prototüübi loome arvestusega, et osa menetlejate tööülesannetest jätkub vanas süsteemis. Seetõttu toome visuaalselt esile kohad, kus prototüüp suunab teise süsteemi või esitab seotud andmeid sellest. Kasutajale näidatakse selgelt, milline tegevus toimub millises süsteemis. Testimise ajal pöörame tähelepanu, kas kasutaja suudab aru saada, millal toimub tööprotsessi katkestus või süsteemidevaheline liikumine.
Ootused Tellijale	• Annab esmast tagasisidet prototüübile • Annab testijate kontaktid • Kinnitab menetleja kasutajaliidese põhivaadete disaini

Näited võimalikest TO-BE menetlejavaadete disainist

Kasutajakogemuse parendamise visiooni ilmentamiseks oleme kokku pannud võimaliku tulevikulahenduse prototüübi Merenduse loataotluse menetlusprotsessi näitel. Prototüübi ja selle läbimise juhiseiga saab tutvuda [sellelt Figma lingilt](#).

Alljärgnevalt on toodud põhilised menetlejavaate kasutajakogemuse parendamise mõtted, mida tahame esile tuua.

Personaalne töölaud

The screenshot shows the 'Töölaua' (Dashboard) interface in the TARVIK MTR system. The top navigation bar includes the system logo, user name 'Kristi Ploom (Menetleja)', and a 'Väljun' (Logout) button. The left sidebar contains a menu with options like 'Registripäevik', 'Registritoimik', 'Teavitused', 'Seaded', and 'Haldus'. The main content area is titled 'Töölaua' and features several sections:

- Tähelepanu vajavad asjad** (Items requiring attention): A summary of key metrics with filters for 'Tegevusala kaup', 'Valdkonna kaup', and 'Piirkonna kaup'. The metrics are: 3 taotlused on 1 päev menetlustähtajani, 52 menetlusel läheneb tähtaeg, 13 taotlust ootavad menetlejat, and 3 taotlust on pikemat aega olnud muutumatu staatusega.
- Minu pooleliolevad menetlused** (My ongoing cases): A table listing cases with columns for ID, description, date, status, and action. It includes a 'Vaata kõiki (13)' link.
- Teavitused (1 uus)** (Notifications): A table showing notifications with columns for date and description. It includes a 'Vaata kõiki (25)' link.
- Mulle allkirjastamiseks saadetud** (Cases sent for signature): A table listing cases with columns for ID, description, date, status, and action. It includes a 'Vaata kõiki (25)' link.

The footer contains contact information, a version number (1.0.19), and logos for the European Union and the Estonian Development Cooperation Fund.

Joonis 5. Personaalse töölaua vaade TARVIKus

Töölaua on **heaks alguspunktiks** menetleja enda tööde (ja tiimi tööde) progressi ja ülevaate saamiseks, enamkasutatavate päringute ning poolelioleva tegevuse juurde naasmiseks. Töölaua on menetleja enda poolt seadistatav. Näiteks saab espool vastavalt eelistustele ja seadistustele kuvada need toimingud või loatüübid, millega menetleja tihedamini kokku puutub, talle suunatud teateid, menetlusprotsessi tähtaegade lähenemist, taotlusi, millele ei ole veel menetlejat määratud jms. See võimaldab ka tiimide üleselt **saada head ülevaadet tööde seisust ning annab võimaluse töid planeerida**.

Kompaktsem menetluse vaade

The screenshot displays the TARVIK MTR web application interface. The main content area is titled 'Tegevusloa taotlus' and 'Tegevusala Merendus'. It contains a form for applying for a license, with fields for personal information, contact details, and application specifics. The right sidebar provides a summary of the process, including the status, deadline, and next steps. The bottom of the screen features a footer with contact information and logos.

Joonis 6. Tegevusloa infovaade TARVIKus

Kasutame disainis **paremriba (1)**, et esile tõsta menetluse protsessi puudutav oluline info: protsessi etapid, tähtaeg, menetleja kommentaarid, allkirjastaja ja otsus koos põhjendusega.

Toome **loogiliselt üksteisega seotud infoväljad ühte kohta (2)**: nt kõik menetlusega seotud kuupäevad ühes kohas, visuaalselt hästi eristatav tähtaeg ja võimalus seda muuta; menetlusprotsessi etapiline progressikuvaga koos samasse kohta toodud võimalusega võtta üks samm tagasi protsessi tagasi jms.

Kasutame akordioni komponenti, et **peita info, mis menetluse otsustusprotsessi seisukohast pole esmaoluline (3)**, et hoida fookust olulistel asjadel kompaktsena.

Visualiseerime menetluse protsessi sammudena (4) ja kuvame staatust. See võimaldab ka väiksema kogemusega menetlejal protsessist lihtsamini ülevaadet saada.

The screenshot displays the 'Tegevusloa taotlus' (Activity License Application) page in the TARVIK MTR system. The page is structured as follows:

- Üldandmed (General Information):** Includes fields for Number (MES000162), Vitenumber (2900082993), Riigilõiv (50€), Taotluse esitaja (Anna Kahvel), Telefon (+372 4545 454), E-post (anna.kahvel@kahvel.ee), Info registrist (Päring rahvastikuregistrist), Taotluse sisestaja (Endel Nuga), Eitamisiis (Ei), and Armetkondlikuks kasutamiseks (Ei).
- Ettevõtte andmed (Company Information):** Highlighted with a red box and a red circle containing the number 5. It includes a table for 'Ettevõtte lisainfo' (Company additional information) with fields like Registrikood, Telefon, Mobiiltelefon, E-post, Veebileht, Aadress, and Kommentaarid.
- Menetlejad (Applicants):** A section titled '1. Menetleja' (Applicant 1) showing details for 'Mihkel Mustikas', including Haldusasutus, Ametikoht, Nimi, Isikukood, Telefon, and E-post.
- Staatuse (Status):** Shows the application status as 'Altkirjastamisel' (Being processed) with a progress bar. It also includes a 'Menetluse tähtaeg' (Processing deadline) of 01.06.2025 and a 'Taotluse esitamise aeg' (Application submission time) of 11.04.2025 10:38.
- AI assistent (AI assistant):** Highlighted with a red box and a red circle containing the number 6. It provides a recommendation based on the application data, stating 'On olnud maksuvõlg. Siin on mitu tähelepanekut ja nüanssi, millele otsuse langetamisel toetuda.' (There has been an outstanding debt. There are several points and nuances to which the decision should be based on).

The bottom of the page features a footer with contact information and logos, including the European Union flag and the Estonian Government logo.

Joonis 7. Tegevusloa infovaade TARVIKus

Toome juurde **automaatpäringud seotud andmebaasidesse (5)**. Menetleja ei pea nupule vajutusega liikuma nt Maksu- ja Tolliameti infosüsteemi, vaid see päring tehakse tema eest automaatselt ning tulemus kuvatakse juba vastaval real, seejuures kriitiline ja positiivne info on visuaalselt eristatud värvidega).

Integreerime **AI-põhise sisselülitatava soovitusmootori menetluse otsuse kohta (6)**, mis kasutab mustreid varasematest sarnastele tingimustele vastavatest menetlustest. AI-põhine soovitusmootor ei ole kogu aeg aktiivne, et mitte koormata ressursse – seda saab vajadusel sisse ja välja lülitada.

Kompaktsem TARVIKu vasakmenüü

The screenshot shows the 'Registritoimik' (Registry) page in the TARVIK MTR system. The sidebar on the left contains navigation links. The main content area displays a table of registry entries. The table has columns: Jrk, Number, Ettevõtja nimi, Registrikood, Kehtivuse algus, Kehtivuse lõpp, Staatus, and Tegevusala. The first row is highlighted with a red box and labeled '9'. The second row is labeled '8'. The third row is labeled '7'. The table also has a 'Filtreeri' button. At the bottom of the table, there is a pagination bar showing '170 tulemust' and '1 2'.

Joonis 8. Registripäeviku ja registritoimiku võimalik koondvaade

Lihtsustamaks TARVIKu menüüd tervikuna, näeme võimalust liita tänased Registritoimiku ja Registripäeviku nimekirjad (7), kasutades tõhusat filtrisüsteemi koos otsingufunktsiooniga (8) ning UX-kirjutamise abi loogilisema menüüpunkti nimetuse leidmiseks, kuid selle otsuse tegemiseks on kahtlemata vaja põhjalikumalt analüüsi.

Ühendatud nimekirjas kuvame selgelt, millises süsteemis konkreetsed load või taotlused asuvad (9), kui neid ei ole üleminekuperioodil veel TARVIKusse toodud.

7. Arendustööde teostamine

Tegevuse sisu	Vastavalt UI disainis ja detailanalüüsis selgunud sisendile luuakse vastavad mikroteenused ja mikroesitluskihid. Mikroteenused liidestatakse protsessimootoriga võimaldamaks paremini juhtida menetlusprotsesside vooge. Vastavalt vajadusele luuakse liidestused ka väliste süsteemidega, näiteks JVIS, Äriregister, vana MTR jne. Arendustöid teostatakse taotluste ja menetluste liikide kaupa. Nii on menetlejal võimalik üleminekuperioodil töötada ühelaadsete loataotluste
----------------------	---

	ning menetlustega ühe süsteemi piires. Hoides vanade ja uute süsteemide vahelised seosed minimaalsed, on võimalik üleminekuperioodi arenduskui ka menetlusprotsesse oluliselt lihtsustada. Tagame süsteemide vahel sujuva navigeerimise rakenduse nime ja ikooni esile tõstmisega kasutajaliideses ning rakendame põhimõtet „autendi ühe korra“ kasutades Keycloak SSO lahendust. Arenduse käigus järgime Scrum põhimõtteid, millest on täpsemalt kirjutatud peatükis „3.1.1 Analüüsi, disaini ja arenduse koostöö ja projekti juhtimise metoodika – <i>Dual Track Scrum</i>“. Arendusi teostatakse pileтите põhjal. Loodud pileteid rafineeritakse ehk valmistatakse arendusteks ette regulaarselt. Lisaks funktsionaalsete nõuete täitmisele järgitakse ka Tellija mittefunktsionaalseid nõudeid ning üldlevinud häid tavasid, sh Clean Code printsiipe. Teostame koodiülevaatusi kogu koodile ning rakendame vähemalt 2 silmapaari põhimõtet. Lisaks inimkontrollile rakendame võimalikult palju automaatseid koodikontrolle, mis aitavad hoida koodis sarnast stiili ning ennetavalt leida üles võimalikud probleemid (nt Checkstyle). Kõik arhitektuurilised jm tehnilised lahendused dokumenteeritakse põhjalikult Tellija keskkonnas ning vajadusel kinnitatakse Tellijaga enne arenduste alustamist. Väliste liideste dokumenteerimiseks kasutame OpenAPI spetsifikatsioone.
Eeldatav mõju	Menetlustoode on piisavalt küps (vastuvõtu)testimiseks.
Üleminekuperioodi tähelepanekud	Ületoomine toimub taotluste ja menetlusliikide kaupa. Arendustööde teostamisel peetakse silmas, et tuues funktsionaalsust üle TARVIKusse, peavad paralleelselt toimima kõik süsteemid ning menetlusprotsessid peavad jääma terviklikuks, et menetlejate tööd mitte takistada. Tehniliste takistuste ilmnemisel kommunikeeritakse see Tellijale, et leida parim lahendus. Erinevate taotluste puhul võib kasutaja kasutada erinevaid süsteeme.
Ootused Tellijale	On kättesaadav küsimuste tekkimisel. Osaleb demodel ja retrodel, et aegsasti arendatud toote osas tagasisidet anda ning arendusprotsessi parendustes kaasa rääkida.

8. Arendustööde testimine

Tegevuse sisu	Valminud arendused läbivad testimist läbi mitme kihi, ehk peale arendustöö valmimist testib valminud tööd esialgu Pakkujapoolne meeskond ning kui arendus läbib testimise, suunatakse arendus Tellijapoolsesse testimisse. Kui arendus testimist ei läbi, teeb Pakkuja vastavad parandused ning see suunatakse uuesti testimisse. Pakkuja kogemusel käib valminud arenduste testimine kogu projekti vältel minimeerimaks riski, et projekti lõpus tuleb korrada teostada palju parandusi, mis seab ohtu projekti valmimise tähtaja. Seega testimist teostatakse pidevalt ja iteratiivselt koos arendustööde valmimisega. Valminud arenduste töökindluse ning funktsionaalsete nõuete täitmise tagamiseks luuakse arendustööde käigus ühiktestid vähemalt 80%
----------------------	---

	äriefunktsionaalsuse ulatuses. Samuti lisatakse automaattestid olulistele integratsioonidele ning tagatakse efektiivne automatsiseeritud regressioonitestimine. Lisaks kasutame staatilisi koodianalüsaatoreid ja sõltuvuste kontrollimise teeke, mis aitavad leida üles võimalikud turvaprobleemid (nt SonarQube, Spotbugs). Jõudluse tagamiseks teostatakse koormustestimine toodangule ligilähedases keskkonnas. Koormustestide jaoks kirjutatakse skriptid ja juhendid võimaldamaks koormusteste lihtsa vaevaga korduvalt jooksutada. Kuigi Pakkuja on läbinud vastavad turvalisuse koolitused ning järgib häid tavasid oma igapäevatoos, tuleks turvalisuses veendumiseks teostada ka turvatestimine kolmandate osapoolte poolt.
Eeldatav mõju	Käesolev tegevus võimaldab veenduda arendustööde korrektsuses, tarnimise valmiduses, aga lihtsustab ka tulevikus regressioonitestimist.
Üleminekuperioodi tähelepanekud	Testimisel peetakse silmas, et paralleelselt töötavad mitmed süsteemid ning testitakse kõik liidesed.
Ootused Tellijale	• Osaleb vastuvõtutestimisel • Annab testimise kohta aegsasti tagasisidet • On kättesaadav küsimuste tekkimisel • Korraldab kõikide uute komponentide turvatestimise

9. Arenduste tarnimine

Tegevuse sisu	Arenduse alguses pannakse koostöös Tellijaga paika uute loodavate teenuste tarneahelad, et arenduse käigus oleks võimalik Tellijale pidevalt uusi lahendusi üle anda. Iga sprindi lõpus tehakse Tellijale demo implementeeritud funktsionaalsustest ja tarnitakse kood koos testidega ning nimekiri funktsionaalsustest, mida antud tarne sisaldab. See võimaldab Tellijal varakult uut funktsionaalsust testida ning anda võimalikult varajases faasis tagasisidet tehtud töödele. Mida varem muudatusvajadused välja tulevad, seda lihtsam ja soodsam on neid juurde lisada. Uue taotluse ja menetlusprotsessi üle viimisega migreeritakse vajadusel vanast süsteemist andmed uude süsteemi. Tarnimise alguses lülitatakse vanas süsteemis olev funktsionaalsus välja, teostatakse andmemigratsioon, tarnitakse uus kood ning lülitatakse uues süsteemis funktsionaalsus sisse. Funktsionaalsuste sisse/välja lülitamiseks kasutame lipukest, mis võimaldab lihtsasti ja kiiresti funktsionaalsust vajadusel peita või avalikustada. Näiteks võimaldab see tarnida toodangusse taotluse koodi, mida pole veel mingil põhjusel soovi avalikustada. See lihtsustab koodihalduses harude majandamist.
Eeldatav mõju	Antud tegevuse järel on loodud arendused Tellijale tarnitud.
Üleminekuperioodi tähelepanekud	Peame oluliseks, et arenduste tarnimisega käib kaasas ka kasutajate koolitus ja tehniline tugi ning kommunikatsioon, mida oleme valmis koostöös Tellijaga planeerima ja korraldama.

Ootused Tellijale	• Parima tulemuse tagamiseks testib/valideerib operatiivselt tarnitavat lahendust agiilselt • Annab tagasisidet lahenduste osas (kas tarnitav lahendus on korrektne või mitte, muudatusvajadused, täiendused, edasiarendused jms) • Paigaldab jooksvalt tarnitava lahenduse (selle automatiseerimine, CI/CD)
--------------------------	--

3.3 Mikroteenuste arhitektuur

Arendajad on pidevalt otsinud uusi viise, kuidas infosüsteeme paremini ehitada. Üks populaarne lähenemine tänapäeval on viljeleda mikroteenusarhitektuuri. Küll aga ei tohiks mikroteenusarhitektuur olla omaette eesmärk, vaid vahend oma eesmärkide saavutamiseks. Antud peatükis räägime, mis on mikroteenusarhitektuur, käime läbi põhiprobleemid, mida mikroteenusarhitektuur lahendab, milliseid probleeme see võib tekitada, kuidas tekkinud probleeme omakorda adresseerida ning kuidas minna üle mikroteenusarhitektuurile.

3.3.1 Mis on mikroteenusarhitektuur?

Mikroteenusarhitektuur on arhitektuurstiil, kus süsteem koosneb mitmetest väikestest autonoomsetest teenustest, mis suhtlevad teineteisega üle kindlaks määratud API. Iga teenus on loodud lahendamaks mingit konkreetset probleemi ning seda saab iseseisvalt tarnida. Samuti on iga teenus oma programmeerimiskeele, andmesalvestus- ja muude tehnoloogiate valikul teistest teenustest sõltumatu. Selline lähenemine võimaldab iga teenuse puhul kasutada just selle probleemi lahendamiseks mõeldud parimaid tööriistu.

3.3.2 Peamised eelised

Ühtegi arhitektuurstiili ei tasu kasutada lihtsalt kasutamise pärast, vaid see peaks aitama lahendada mingeid probleeme. Antud peatükis arutame, mis probleeme mikroteenusarhitektuuri kasutamine lahendab ja kuidas seda rakendada kasutades parimaid praktikaid.

Autonoomsus

Korrektse lahenduse korral on iga mikroteenus autonoomne. See tähendab, et mikroteenus on piisavalt eraldiseisev, et seda saaks tarnida teistest teenustest sõltumatult. Andmete saatmine üle võrgu aitab jõustada teenuste vahelist eraldatust. Sellest üksi aga ei piisa. Oluline on ka see, mis andmeid teenus välja annab ja mida mitte. Kui teenust pakkuv mikroteenus edastab tarbivatele mikroteenustele liiga palju sisemist infot, siis teenuse tarbijad muutuvad pakkuva teenuse sisemisest ülesehitusest sõltuvaks, mis omakorda vähendab teenuste autonoomsust.

Agiilsus

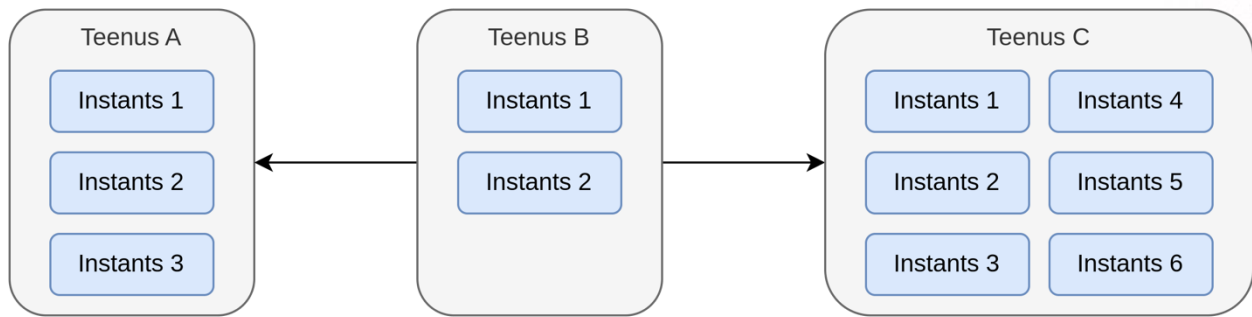
Sõltumata muudatuse suuruselt nõuab monoliitses rakenduses muudatuse tegemine kogu süsteemi ehitamist ja paigaldamist. See omab suurt mõju kogu süsteemile, mistõttu võtavad ettevetaatusabinõude kasutamised, näiteks kogu süsteemi testimine, märgatavalt aega. Selle aja jooksul võib aga valmida juba uusi funktsionaalsusi, mis samuti vajavad testimist. See aga tähendab, et üks hetk, kui ollakse valmis uut versiooni tarnima, sisaldab kood juba suurel hulgal muudatusi. Mida rohkem aga on muudatusi seda suurem on tõenäosus, et midagi võib katki minna ning seda raskem on tuvastada probleemi põhjust.

Mikroteenuste puhul on nende autonoomsuse tõttu võimalik teha muudatus üksnes ühes teenuses ning tarnida see teistest sõltumatult. See võimaldab uue funktsionaalsusega kiiremini toodangusse minna. Vea ilmnemise korral on seda aga lihtsam tuvastada ning konkreetse teenuse varasem versioon taastada.

Skaleeritavus

Jõudluse suurendamiseks võib rakendusest käivitada mitu instantsi. Monoliitse rakenduse puhul tähendab see kogu rakenduse skaleerimist, mis ei ole ressursikasutuse vaates mõistlik. Mikroteenuste puhul on võimalik luua lisainstantse üksnes sellest teenusest, mille poolt pakutavat funktsionaalsust on vaja üles skaleerida. Sarnaselt saab vähendada instantsi nendest teenustest, mis nii palju ressursse

ei vaja. Samuti pole monoliitse rakenduse korral võimalik kergesti tuvastada alammoodulite ressursivajadusi.



Joonis 9. Teenustest kindla arvu instantside loomine vastavalt vajadusele

Vastupidavus

Üks võimalus muuta süsteeme tõrkekindlamaks, on tekitada selle komponentide vahele kindlad piirid, mis takistavad probleemide levikut. Kui monoliitses rakenduses lakkab teenus töötamast, siis lakkab töötamast ka kogu süsteem. Kui aga üks mikroteenus lakkab töötamast, siis korrektse ülesehituse puhul esineb probleem vaid teatud funktsionaalsuses, kuid ülejäänud süsteem töötab edasi.

Siiski, tuleb aru saada, et süsteemi jagamine väikesteks teenusteks üksi süsteemi vastupidavust ei tõsta. Teenused peavad olema teineteisest ka piisavalt sõltumatud suutmaks jätkata tööd, kui mõni teenus enam ei vasta. Selleks peavad teenused omama suurt sisemist ühtekuuluvust kuid lõtvu seoseid teiste teenustega. Sellest räägitakse lähemalt peatükis [Mikroteenusteks jaotamine](#). Lisaks teistele teenustele võib lakata töötamast ka võrk üle mille teenused omavahel suhtlevad. Antud teemat on täpsemalt lahatud peatükis [Võrguprobleemid](#).

Tehnoloogiline heterogeensus

Iga mikroteenus on mõeldud lahendama mingit konkreetset probleemi. Erinevate probleemide lahendamiseks võib olla aga vaja kasutada erinevaid tehnilisi lahendusi. Monoliitse rakenduse korral oleme me kammitsetud nende tehnoloogiliste valikutega, milles rakendus on üles ehitatud. Samuti on uute tehnoloogiate kasutuselevõtu mõju terviksüsteemile suurem.

Mikroteenusarhitektuuri puhul võib aga iga teenus kasutada erinevat tehnoloogilist pinu. Näiteks on kõik TARVIKu mikroteenused loodud kasutades Java programmeerimiskeelt. Kui aga soovitakse kasutusele võtta mõni masinõppelahendus, mille jaoks hetkel parimad teegid on loodud näiteks Pythonis, siis saab konkreetselt selle teenuse jaoks kasutusele võtta täiesti teise programmeerimiskeele.

Taaskasutatavus

Monoliitsete rakenduste puhul implementeerib iga rakendus enda jaoks vajalikku funktsionaalsust uuesti. See tähendab, et sama koodi kirjutatakse üha uuesti ning kui selle loogikat on vaja muuta, siis tuleb seda teha mitmes rakenduses. Näiteks on X-tee liides arendatud valmis nii MTR kui ka JVIS rakendustes. Kui mõlemas rakenduses oleks vaja hakata üle X-tee lisaandmeid küsima, siis tuleb vastav arendus teha mõlemas rakenduses eraldi.

Mikroteenuste puhul kapseldatakse loogika eraldi teenusesse, mida teised teenused tarbida saavad. Võtame näiteks X-tee mikroteenuse. Iga rakendus, mis peab X-tee poole pöörduma, ei pea sisaldama keerukat SOAP-päringute seadistamise loogikat. Selle asemel võivad nad pöörduda X-tee

mikroteenuse poole, kus antud probleem on ühe korra lahendatud. Samuti saavad tarbivad teenused ligipääsu ka teistele X-tee päringutele, mida X-tee mikroteenusesse juurde arendatakse.

3.3.3 Peamised probleemid

Samal ajal kui mikroteenusarhitektuur lahendab osad probleemid, võivad tekkida uued, mida näiteks monoliitsete rakenduste puhul ei esinenud. Antud peatükis käime läbi tekkivad probleemid ning pakume välja võimalikud lahendused.

Hajutatud transaktsioonid

Transaktsioone kasutatakse andmete tervikluse tagamiseks - kõik tehtud muudatused kas talletatakse või lükatakse tagasi. See aitab vältida vigase andmekoosseisu teket, kus osad muudatused on salvestatud ja osad mitte. Selle saavutamiseks rakendavad andmebaasimootorid rida omadusi ([ACID](#)), mis antud võimekust tagavad.

Monoliitses süsteemis on transaktsiooni haldamine üldjuhul lihtne, kuna transaktsioon toimub ühe konkreetse rakenduse protsessi raames. Mikroteenusarhitektuuri puhul, kus igal mikroteenusel võib olla oma andmebaas, läbivad päringud mitmeid rakendusi, mis omakorda võivad teostada andmemuudatusi mitmes andmebaasis. Selleks, et vältida vigase andmekoosluse teket, kus ühes mikroteenuses andmed salvestati, aga teises mikroteenuses mitte, tuleb läbi mõelda kuidas süsteem erandjuhtude korral tervikuna käitub.

Kõige lihtsam variant on see, kui muudatuste talletamine ei ole hajutatud mitme mikroteenuse peale laiali. Seda aitab saavutada mikroteenuste piiride korrektne paikapanek. Täpsemalt räägitakse antud teemast peatükis [Mikroteenusteks jaotamine](#). Hajutatud andmete talletamine on teatud juhtudel aga vältimatu. Sellisel juhul tuleb kaaluda alternatiivseid variante.

Üheks hajutatud transaktsioonide haldamise võimaluseks on kasutada kahe-faasilist kinnitamist (*two-phase commit*, 2PC). 2PC puhul algatab koordinaator (transaktsioonihaldur) uue transaktsiooni ja küsib igalt andmebaasilt, kas see saab antud andmed talletada. Selleks veendumiseks tuleb andmebaasidel lukustada vajalikud tabelid, read või väljad. Kui muudatuste talletamine on võimalik, siis teavitatakse sellest koordinaatorit. Kui kõik andmebaasid on andnud positiivse vastuse, teavitab koordinaator kõiki andmebaase, et muudatused võib lõplikult talletada (*commit*). Kui mõni andmebaas annab negatiivse vastuse, siis koordinaator annab andmebaasidele tagasivõtmise käsu (*rollback*).

2PC negatiivse küljena võib aga välja tuua selle, et kui mõni andmebaas peaks lakkama töötamast või mõnel muul põhjusel mitte vastama, siis jääb koordinaator ootama ning oma toiminguid ei saa lõpetada ka teised antud transaktsiooniga seotud andmebaasid. See omakorda aga tähendab, et andmebaasilukud jäävad peale, mis hakkab takistama juba järgmiste päringute tööd, mis järjekorras ootavad. Samuti saab koordinaatorist keskne teenus, mille tööst sõltuvad kõik teised teenused. See vähendab märgatavalt mikroteenuste autonoomsust ja viib need teineteisega suurde sõltuvusse.

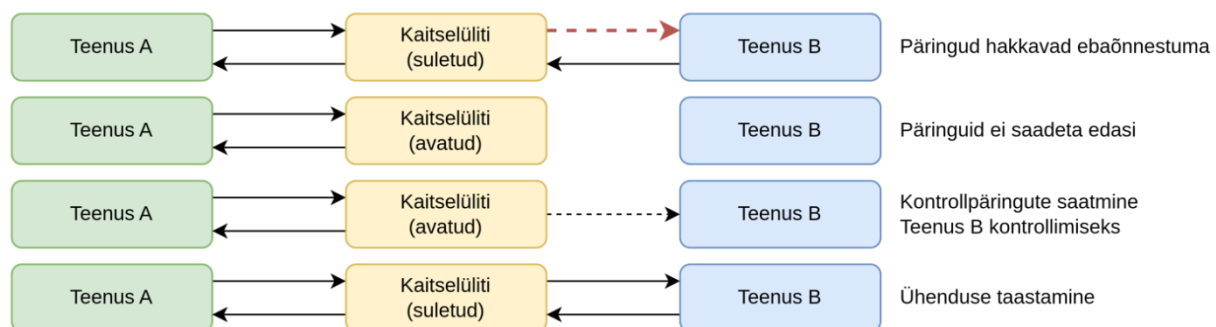
Teine võimalus on võtta kasutusele SAGA-muster. SAGA on jada mikroteenuste lokaalsetest transaktsioonidest. Iga lokaalse transaktsiooni raames uuendatakse ühe mikroteenuse andmebaasi. Kui muudatused õnnestuvad, siis käivitatakse järgmise mikroteenuse poolt tehtavad andmebaasimuudatused. Kui lokaalne transaktsioon aga ebaõnnestub, siis käivitatakse kompenseerivad tegevused, mis juba eelnevalt lõpetatud lokaalsete transaktsioonide muudatused tagasi võtavad. SAGA mustrit on võimalik implementeerida kasutades protsessimootorit, mis tagab hea ülevaate põhivoo ja kompenseerivatest tegevustest. SAGA mustri kasutamist protsessimootoris on käsitletud peatükis [Kompenseerivad tegevused](#).

Võrguprobleemid

Monoliitsetes süsteemides käitatakse suur osa äriloogikast ühe rakenduse piires. Mikroteenusarhitektuuri puhul toimub aga suur osa andmevahetusest üle võrgu. Seetõttu tekib vajadus pöörata suuremat tähelepanu võrguliiklusest tingitud probleemidele. Teenused, mille pihta päringuid tehakse, ei pruugi olla kättesaadavad või vastavad tavapärasest oluliselt aeglasemalt. Esimesel juhul on veaolukorda lihtsam tuvastada ning sellele ka reageerida. Aeglaselt vastavate teenuste puhul kõik justkui töötaks, aga reaalsuses pole süsteem kasutatav – selle tuvastamine on palju keerulisem.

Esimese, kõige lihtsama abimeetodina tuleks määrata päringutele mõistlikud ajapiirangud (*timeout*), mis määravad ära, kui kaua päring võib üldse kesta. See aitab vabastada ressursse, lõpetades hangunud päringud. Samuti aitab see vältida süsteemi tõrgete kaskaadistumist, kus süsteem justkui töötab, aga kõik teenused ootavad tegelikkuses vastuseid, mida nad kunagi ei saa. See võib omakorda põhjustada ootamatu teenuste töötamise lõppemise tulenevalt vabade ressursside puudumisest.

Kui alamteenus enam ei vasta, siis pole mõtet sinna rohkem päringuid teatud aja jooksul saata, vaid tagastada kliendile kohe viga ning lasta alamteenusel taastuda. Selleks kasutatakse kaitselüliteid (*circuit breaker*). Kui järjestikuste ebaõnnestunud päringute arv ületab ette antud piirmäära, siis päringuid enam edasi ei saadeta ning mõne aja jooksul ebaõnnestuvad kõik alamteenusele saadetavad päringud. Kui oodatav aeg saab läbi, siis hakkab kaitselüliti mõningaid päringuid läbi laskma kontrollimaks, kas ühendus on taastunud. Kui kontrollpäringud õnnestuvad, siis jätkab kaitselüliti normaalset tööd.



Joonis 10. Kaitselüliti tööpõhimõte

Idempotentsus

Kuna mikroteenused suhtlevad üle võrgu, siis võib juhtuda, et päring jõudis alamteenusesse kohale, aga vastus päringu algatajale mitte. Sellisel juhul võidakse sama päring uuesti teostada. Olenevalt päringu tüübist võib see tekitada probleeme. Näiteks kui päringu tulemusena saadetakse välja uus email, võib juhtuda, et sama emaili saadetakse mitu korda. Sellise probleemi vältimiseks on mõned võimalused nii saatja kui vastuvõtja poole jaoks.

Kui toiming on asünkroonne, siis võib kasutada järjekordi (*queue*) nagu nt RabbitMQ, koos transaktsiooniga – andmebaasimuudatus ja sõnumi järjekorda panemine tehakse ühe atomaarse toiminguna kasutades 2PC-d. See garanteerib, et sõnum pannakse järjekorda üksnes siis, kui muud sammud samuti õnnestuvad. See aitab vältida sama sõnumi mitmekordset väljasaatmist.

[Hajutatud transaktsioonide](#) peatükis sai aga välja toodud 2PC negatiivsed küljed. Antud juhul saab selle vältimiseks kasutada [Transactional outbox](#) mustrit, kus ühe transaktsiooni koosseisus salvestab teenus esmalt andmebaasimuudatused koos välja saadetava sõnumiga andmebaasi. Seejärel saadab teine eraldiseisev protsess andmebaasi salvestatud sõnumi järjekorda.

Järjekorras andmete lugemine peab sellisel juhul toimuma samuti transaktsiooni koosseisus. Tarbiv teenus loeb järjekorras sõnumi ja teostab vajalikud toimingud. Kui antud toimingud õnnestuvad, siis sõnum kustutatakse. Ebaõnnestumise korral võtab teenus tehtud muudatused tagasi ning sõnum pannakse järjekorda tagasi.

Kui järjekorda pole aga võimalik või mõistlik kasutada, siis tuleks tagada päringute idempotentsus. See tähendab, et sama päringu mitmekordsel saamisel süsteemi olek ei muutu. Selleks antakse kaasa tunnus(ed), mille abil saab alamteenus veenduda, kas antud tunnusega seotud toimingud juba teostati või mitte. Näiteks jätab teenus meelde, et teatud tüüpi email on antud aadressile juba saadetud ning uuesti seda pole vaja enam saata.

Jälgitavus

Võttes kasutusele mikroteenusarhitektuuri, on vaja ühel hetkel jälgida kümnete mikroteenuste tööd ja nende vahelist suhtlust. Selleks, et saada hea ülevaade süsteemis toimuvast on vaja koondada laiali olev telemeetria info ühte tsentraalsesse kohta, mis võimaldab andmeid analüüsida ja informatsiooni otsida. Kui kliendi poolt algatatud päring läbib mitmeid teenuseid, millest võib joosta mitmeid instantse, siis on vaja telemeetria andmetele lisada juurde unikaalne kirje, mille abil saaks erinevates rakendustes tekkivad kirjed kokku agregeerida. See võimaldab lihtsasti jälitada päringut läbi mitmete teenuste, lihtsustab logide uurimist ning võimaldab kiiresti probleeme tuvastada.

Turvalisus

Tänapäeval on levinud praktika, et kogu suhtlus kliendi ja serveri vahel toimub üle turvatud võrgu kasutades TLS-i. Mikroteenusarhitektuuri puhul tekib juurde aga palju uusi rakendusi, mis samuti suhtlevad teineteisega üle võrgu. Tavalise HTTPS ühenduse puhul veendub klientrakendus, et serverrakendus on see, mis ta väidab enda olevat, aga mitte vastupidi. Enamasti toimub see suhtlus küll sisevõrgus, kuid [Zero Trust arhitektuuri](#) põhimõtetest lähtuvalt ei tohi usaldada ühtegi kasutajat, seadet ega süsteemi isegi siis, kui nad asuvad kohalikus võrgus. Siinkohal tuleb appi mTLS, mis tagab, et mõlemad osapooled on need, kes väidavad end olevat.

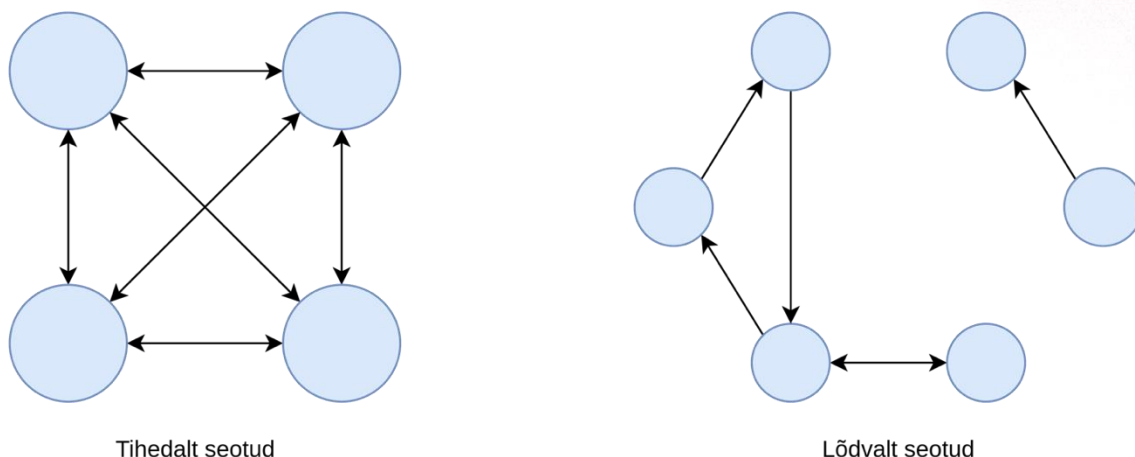
Mikroteenusarhitektuuri puhul on seega vaja turvata kümnete teenuste vahelist suhtlust. Selle probleemi lahendamiseks on võimalik kasutada näiteks [Istiot](#), tarkvarakihti, mis aitab turvata mikroteenustel põhinevaid rakendusi ilma, et oleks vaja muuta rakenduste lähtekoodi. Selleks lisab Istio Kubernetese Podidele juurde *sidecar-proxy*, mis krüpteerib väljuvad päringud ja dekrüpteerib sissetulevad päringud, nii et päringu sisu pole välistele osapooltele arusaadav täites MFN-is välja toodud TLS valmiduse nõude.

3.3.4 Mikroteenusteks jaotamine

Eelnevat kirjeldasime mikroteenusarhitektuuri kasulikke omadusi, aga ka sellega kaasnevat lisakeerukust ning probleeme. Järgmistes peatükkides anname ülevaate headest praktikatest süsteemi mikroteenusteks jaotamisel. Heal mikroteenusel on mitmeid omadusi, millest vast kõige olulisemad on lõdvad seosed väliste teenustega ning suur sisemine ühtekuuluvus.

Lõdvad seosed

Lõdvad seosed (*loose coupling*) tähendavad, et teenuste vahel on vähe sõltuvusi, kui üldse. Sellisel juhul ei nõua ühe teenuse muutmine teiste teenuste muutmist, mis on mikroteenuste üks peamisi eesmärke. Lõdvalt seotud teenused teavad teistest teenustest nii vähe kui võimalik. Samuti suhtlevad sellised teenused teineteisega minimaalselt, sest lisaks jõudlusprobleemide tekkele võivad palju suhtlevad teenused olla teineteisega tihedalt seotud, mis tähendab, et me ei pruugi saada neid teenuseid eraldi tarnida.



Joonis 11. Tihedalt ja lõdvalt seotud teenused

Suur ühtekuuluvus

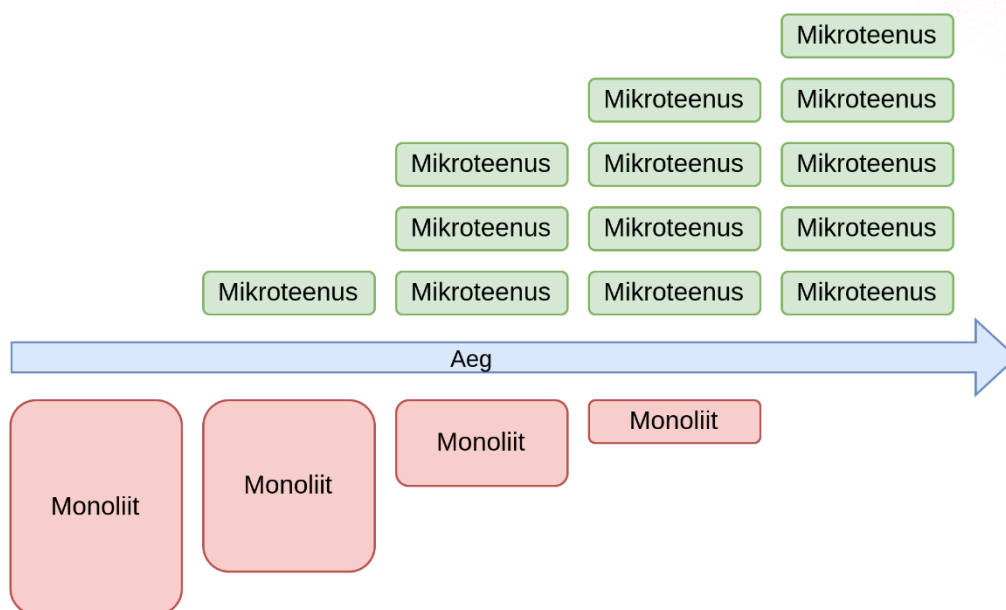
Suur ühtekuuluvus (*high cohesion*) tähendab, et igal teenusel on konkreetne kitsas ja selge vastutusalala ning kõik teenuse funktsioonid ja andmed kuuluvad loogiliselt kokku. See võimaldab muuta loogikat vajadusel ühes kohas ning tarnida muudatused võimalikult kiiresti. Kui sama loogika on jaotatud mitmesse teenusesse laiali, siis tuleb muudatusi teha mitmes teenuses ja võib-olla need ka koos tarnida. Sellisel juhul pole tegu mitte mikroteenustega vaid hajutatud monoliidiga.

Domeenipõhine disain

Mikroteenuste piiride määratlemisel on domeenipõhine disain ([DDD](#)) tõhus lähenemine, kuna see keskendub äri loogikale ja domeeni mõistmisele. DDD aitab vältida teenuste lõhkumist tehniliste või juhuslike piiride järgi, keskendudes selle asemel äri väärtusele. Esimene samm on äri valdkonna kaardistamine ning domeeni ekspertidega koostöös ühise sõnavara loomine. Domeeni võib omakorda jagada väiksemateks spetsiifilisteks aladomeenideks. Seejärel jaotatakse domeen selgelt piiritletud loogilisteks üksusteks ([Bounded Context](#)), mille sees mõisted ja reeglid on üheselt mõistetavad. Iga loogiline üksus on hea kandidaat iseseisvaks mikroteenuseks. Tähtis on leida loomulikud teenuste piirid, kus muutused ühes teenuses ei mõjutaks teisi. Mikroteenused peavad omama selget vastutust ning mitte jagama andmebaase teiste teenustega. Kommunikatsioon erinevate mikroteenuste vahel peab toimuma kindlalt piiritletud API kaudu. Selline lähenemine võimaldab hõlpsasti lisada uusi komponente või vajadusel olemas olevaid välja vahetada.

Monoliidi lammutamine

Rakenduste üleviimine uutele tehnoloogiatele on aeganõudev protsess. Vana süsteemi ümber kirjutamine võib võtta aastaid. Selle asemel, et oodata, kuni uus süsteem täielikult valmis saab, võib sellega hoopis järkjärguliselt toodangusse minna. [Strangler Fig](#) muster võimaldab sujuvat ja kontrollitud moderniseerimist, säilitades süsteemi töökindluse kogu protsessi vältel. Esialgu identifitseeritakse selged ärifunktsioonid või domeenid, mida saab eraldada ilma kogu süsteemi korraga ümber kirjutamata (vt peatükki [Domeenipõhine disain](#)) ning luuakse nende jaoks iseseisvad mikroteenused, millel on oma andmehoidla ja API. Pääsulüüsi (API Gateway) abil suunatakse liiklus kas vana või uue teenuse poole sõltuvalt funktsioonist. Järk-järgult kantakse ka andmed üle uutele teenustele ning vanas süsteemis vastavad komponendid deaktiveeritakse. Andmete migreerimiseks saab kasutada nt SQL-is kirjutatud migratsiooniskripte koos [Liquibase](#)-ga, mis võimaldab migratsioone versioneerida. Aja jooksul väheneb monoliidi roll, kuni see muutub täielikult asendatuks.



Joonis 12. Strangler Fig muster

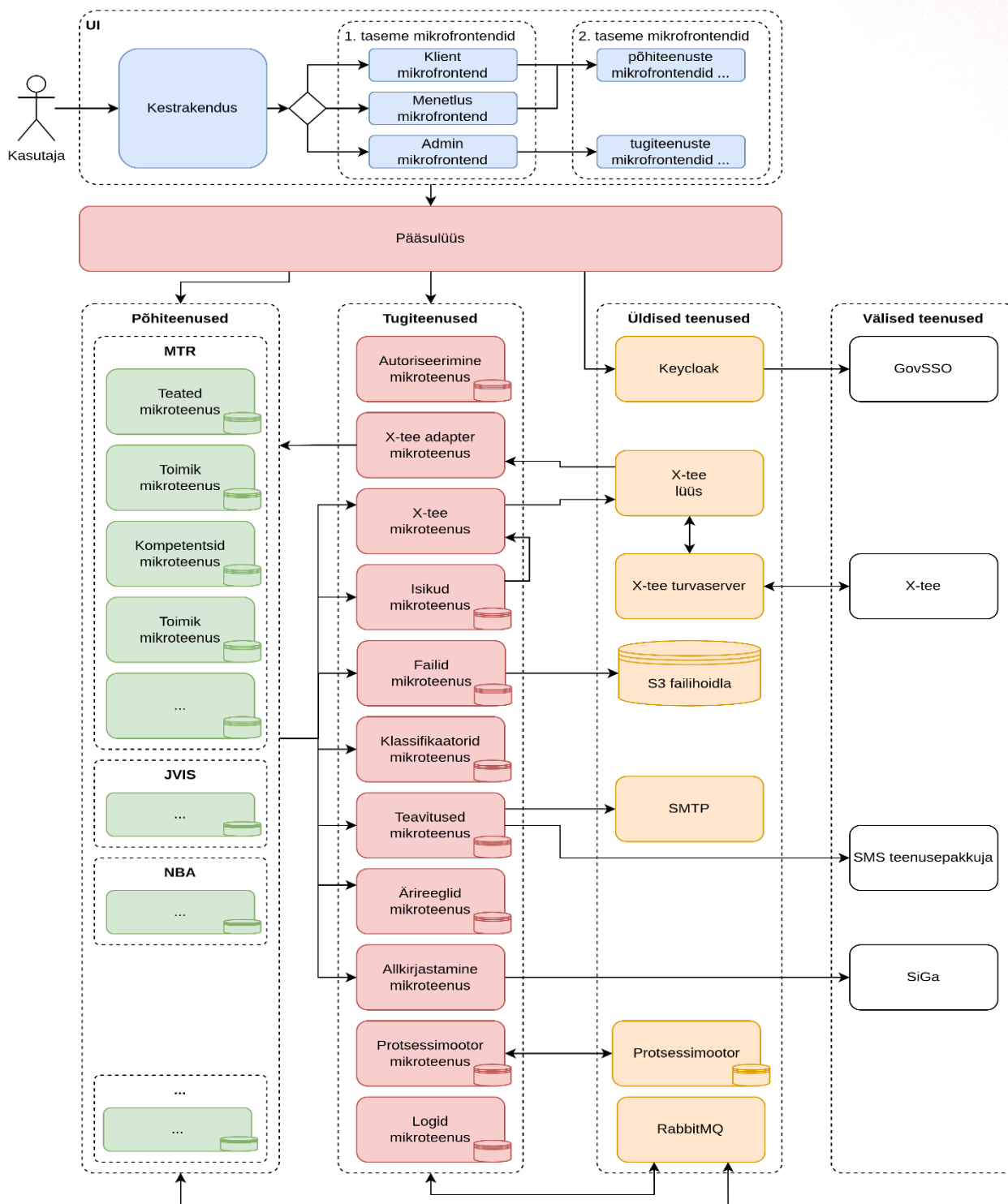
3.3.5 Mikroteenustel põhineva süsteemi arhitektuur

Võttes aluseks hankes alusdokumentides oleva arhitektuuri visiooni dokumendi, olemasolevat süsteemi kirjeldava arhitektuuri joonise, KeMITi IT-profiili ning mittefunktsionaalsed nõuded, pakume antud peatükis välja Pakkuja poolse arhitektuuri visiooni. Samuti võtame arvesse infotunnil välja toodud asjaolu, et praeguseks implementeeritud mikroteenused suhtlevad teineteisega kasutades HTTP-d, mitte AMQP protokollit ning seda ei soovita muuta. Küll aga jääb AMQP kasutusse tegevuslogide saatmiseks praegusel kujul. Samuti kasutame AMQP protokollit uute asünkroonsete tegevuste puhul nagu näiteks protsessimootoriga suhtlemine.

Diagrammil on välja toodud need teenused, mis hetkel teadaolevalt eksisteerivad. Loodavate mikroteenuste koosseis selgub alles [analüüsi faasis](#).

Süsteemi arhitektuur

Antud peatükis kirjeldatakse süsteemi arhitektuuri. Mikroesitluskihtide ja Protsessimootori arhitektuur on kirjeldatud allpool olevates peatükkides. Kolme punktiga (“...”) on näidatud, et lisaks teada olevatele teenustele kuuluvad siia ka analüüsi käigus selguvad ning teistesse domeenidesse kuuluvad teenused.



Joonis 13. Mikroteenusarhitektuuril põhineva TARVIKu arhitektuurijoonis

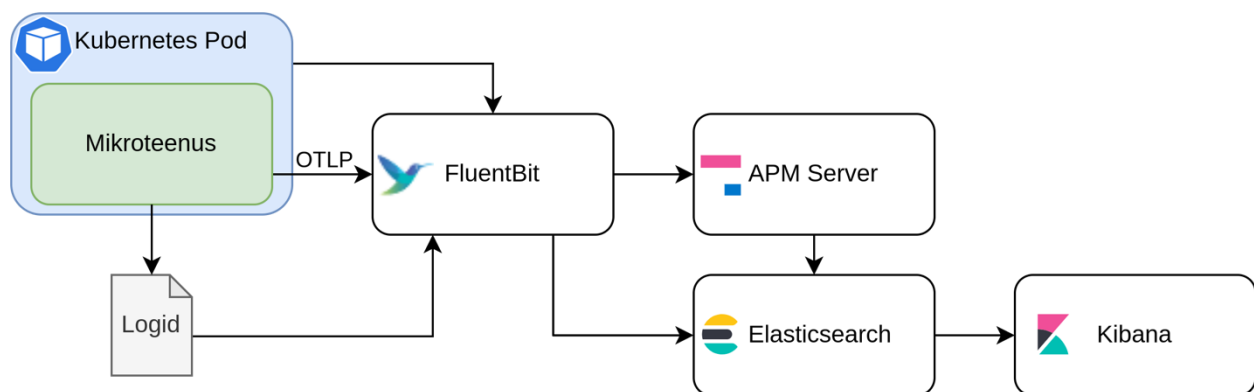
- **Pääsulüüs** - Kogu suhtlus välismaailma ja mikroteenuste vahel toimub läbi pääsulüüsi. Pääsulüüs kontrollib, kas kasutaja poolt päringuga kaasa antud *token* on korrektne ning vastavalt sellele suunab päringu edasi õigele mikroteenusele või tagastab vea. Kuna tegu on keskse komponendiga, siis tuleb antud teenus hoida võimalikult lihtne, vältides äriloogika lisamist antud kihti. Samuti peab sellest kindlasti käivitama mitu instantsi, kuna vastasel juhul on antud teenuse maha kukkumise korral terve süsteem kättesaamatu.
- **Põhiteenused** - Teenused, mis sisaldavad ärispetsiifilist funktsionaalsust.

- Tugiteenused – Teenused, mis pakuvad tuge põhiteenustele pakkudes vajalikke teenuseid ja liidestusi
 - Autoriseerimine mikroteenus – Tegeleb rollide ja privileegide haldamisega. Siin pannakse paika, mis õigused mingi rolli omamisega saadakse.
 - X-tee adapter mikroteenus – Pakub liidest välistele süsteemidele info pärimiseks TARVIKust üle X-tee. Selle teenuse kaudu ei küsi TARVIK andmeid üle X-tee.
 - X-tee mikroteenus – Pakub liidest TARVIKu sisemistele mikroteenustele andmete pärimiseks välistest süsteemidest üle X-tee (nt rahvastikuregistrist ja äriregistrist). Võimaldab mikroteenustel teha X-tee päringuid ilma, et peaksid implementeerima X-tee spetsiifilist loogikat.
 - Isikud mikroteenus – Haldab süsteemis (nt majandustegevusteadetel) olevaid isikuid ja nende kontaktandmeid (aadress ja telefon). Vajadusel küsib ja uuendab andmeid rahvastikuregistrist üle X-tee.
 - Failid mikroteenus – Pakub failide üleslaadimise ja allalaadimise teenust.
 - Klassifikaatorid – Teenus haldab klassifikaatoreid, mida kasutatakse üle süsteemi. Tagab, et kõikidel teenustel on kasutada täpselt samad klassifikaatorid.
 - Teavitused mikroteenus – Pakub sisemistele teenustele funktsionaalsust, mille abil saab välja saata erinevat tüüpi teavitusi (nt email, SMS).
 - Ärireeglid mikroteenus - Võimaldab hallata ärireegleid kasutades DMN-i. Ärireeglite abil saab mõjutada süsteemi või protsessimootori käitumist.
 - Allkirjastamine mikroteenus – Vastutab mikroteenusetele ette antud dokumentide allkirjastamise eest kasutades riiklikku SiGa teenust.
 - Protsessimootor mikroteenus – Teenus, mis käitub vahetult protsessimootori ja teiste mikroteenuste vahel. Vaata täpsemalt [protsessimootori arhitektuuri](#) peatükist.
 - Logid mikroteenus – Kogub kokku teenuste poolt RabbitMQ-sse saadetud toimingute logid (auditlogid) ning talletab need baasi. Pakub liidest logide vaatamiseks. Võimaldab administraatoritel saada ülevaate kasutajate tegevustest.
- Üldised teenused – Teenused, mis ei oma spetsiifilisi teadmisi TARVIKu kui süsteemi kohta.
 - Keycloak – Identiteedi ja ligipääsuhaldamise süsteem, mis tegeleb kasutajate autentimisega ja ligipääsuõiguste kontrollimisega. Suunab kasutaja edasiseks autentimiseks riigi poolt pakutavasse GovSSO-sse.
 - X-tee lüüs - KeMITis kasutatav rakendus, läbi mille suhtlevad sisemised rakendused X-tee turvaserveriga.
 - X-tee turvaserver – Rakendus, mille kaudu suheldakse üle X-tee teiste teenusepakkujatega (nt rahvastikuregister ja äriregister). Turvaserveri peamine ülesanne on päringute vahendamine viisil, mis tagaks nende hilisema tõendusväärtuse.
 - S3 failihoidla – MinIO rakendus, mis implementeerib S3 liidest, mille kaudu saab lisada ja küsida faile.
 - SMTP – Emaili server.
 - Protsessimootor – Teenuste komplekt, mis on mõeldud BPMN-is kirjeldatud protsesside jooksutamiseks. Vaata täpsemalt [protsessimootori](#) peatükis.
 - RabbitMQ - Sõnumivahetusteenus, mida kasutatakse süsteemi erinevate komponentide vaheliseks asünkroonseks suhtluseks. TARVIKu puhul saadetakse selle kaudu näiteks toimingute logisid. Samuti käiks üle selle suhtlus ka protsessimootoriga.
- Välised teenused

- GovSSO – Riigi poolt pakutav SSO teenus, mille abil asutus saab oma e-teenuses ühekordse sisselogimisega autentida ID-kaardi, Mobiil-ID, Smart-ID ja Euroopa Liidu eID kasutaja.
- X-tee - Riiklik andmevahetusplatvorm, mis võimaldab turvalist ja tõestusväärtust tagavat internetipõhist andmevahetust liitunud asutuste vahel.
- SMS teenusepakkuja - Väline teenusepakkuja, kes pakub API-t, üle mille on võimalik saata kasutajatele SMS-e (nt Telia, Twilio, vms).
- SiGa – Riiklik allkirjastamise teenus, mis võimaldab luua allkirjastatud ümbrikke. Toimingud tehakse kasutaja saadetud dokumendiräside põhjal.

Jälgitavuse arhitektuur

Antud peatükis on ära kirjeldatud, kuidas tagatakse ülevaade süsteemi tööst.



Joonis 14. Jälgitavuse arhitektuur

Telemeetria andmete kogumiseks on välja töötatud ja laialdaselt kasutusel gRPC-l põhinev [OTLP](#) protokoll, mis on disainitud kogumaks ja edastamaks telemeetria andmeid nagu logid (*logs*), meetrikad (*metrics*) ja jäljed (*traces*). Logide puhul on tegemist tekstiliste andmetega, mida rakendused maha kirjutavad. Meetrikad on ajas mõõdetavad andmepunktid rakenduse oleku, jõudluse vms hindamiseks. Jäljed kirjeldavad konkreetse päringu voogu läbi hajutatud süsteemi aidates mõista, päringute teekonda ja kulunud aega. TARVIKus loodud ja loodavad mikroteenused kasutavad IT-profiilis välja toodud Spring Boot raamistikku, mis toetab telemeetria andmete saatmist OTLP protokoll põhjal. Arhitektuuri visiooni dokumendis on ka mainitud FluentBiti ning IT-profiilis Elasticsearchi. Lähtudes nendest teadmistest pakume välja järgmine koosseisu jälgitavuse tagamiseks. Võimalikult palju on üritatud ära kasutada olemas olevaid lahendusi ning Elastic-u tooteperekonda kuuluvaid.

- Mikroteenus - Iga mikroteenus saadab välja OTLP gRPC standardile vastavaid päringuid meetrikate ja jälgede kohta. Logide puhul kirjutatakse logid standardväljundisse, kust FluentBit neid lugeda saab. Logide formaat järgib [ECS spetsifikatsiooni](#) (iga logikirje on üherealine JSON objekt, isegi veateadete korral) lihtsustamaks nende töötlemist Elasticsearchi poolt. Samuti sisaldab iga logikirje *trace_id* ja *span_id* väärtusi, võimaldades agregeerida kokku logikirjed, mis on seotud mitut mikroteenust läbinud päringuga.
- FluentBit – Alates versioonist 2.0 toetab FluentBit OTLP protokoll telemeetriaandmete saatmiseks. Pakutavas visioonis kogub FluentBit rakenduse logiandmeid konteineri logifailist ning saadaks edasi Elasticsearchi. Samuti võtab see vastu meetrika ja jälgede andmed ning edastab Elastic APM Serverile. Me ei kasutaks OTLP protokoll logide kogumiseks, kuna võrreldes failist lugemisega oleks see ressursinõudlikum ja ühenduse puudumise korral

võivad osad logid kaduma minna. Meetrikate ja jälgede puhul see aga probleeme ei põhjustaks.

- APM Server – Olemasolevat tehnilist pinu tuleks täiendada Elastic-u poolt pakutava [APM Server](#)-iga, mis võimaldab vastu võtta OTLP protokolliga saadetud andmeid ning talletada need sobival kujul Elasticsearchi.
- Elasticsearch - [Elasticsearch](#) on andmebaas ja otsingumootor, mis võimaldab efektiivselt otsida ja analüüsida suurtes kogustes andmeid.
- Kibana – Kibana abil on võimalik otsida rakenduste logisid, jälgida päringute vooge läbi hajutatud süsteemi ning visualiseerida meetrikaid. Lisaks on võimalik kirjeldada ära reeglid, millele meetrikad peavad vastama ning mittevastavuse korral saab välja saata [teavitusi](#). See võimaldab kiiresti reageerida võimalike süsteemi vigade korral.

Latency distribution



Trace sample < 1 of 500 >

Investigate

View full trace

2 days ago 13 ms (100% of trace)

GET http://10.11.249.199:3000/api/stats

200 OK

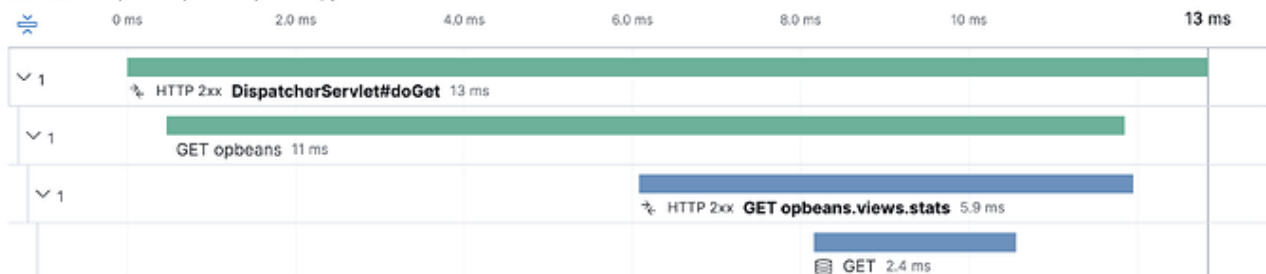
Python aiohttp (3.3.2)

Timeline

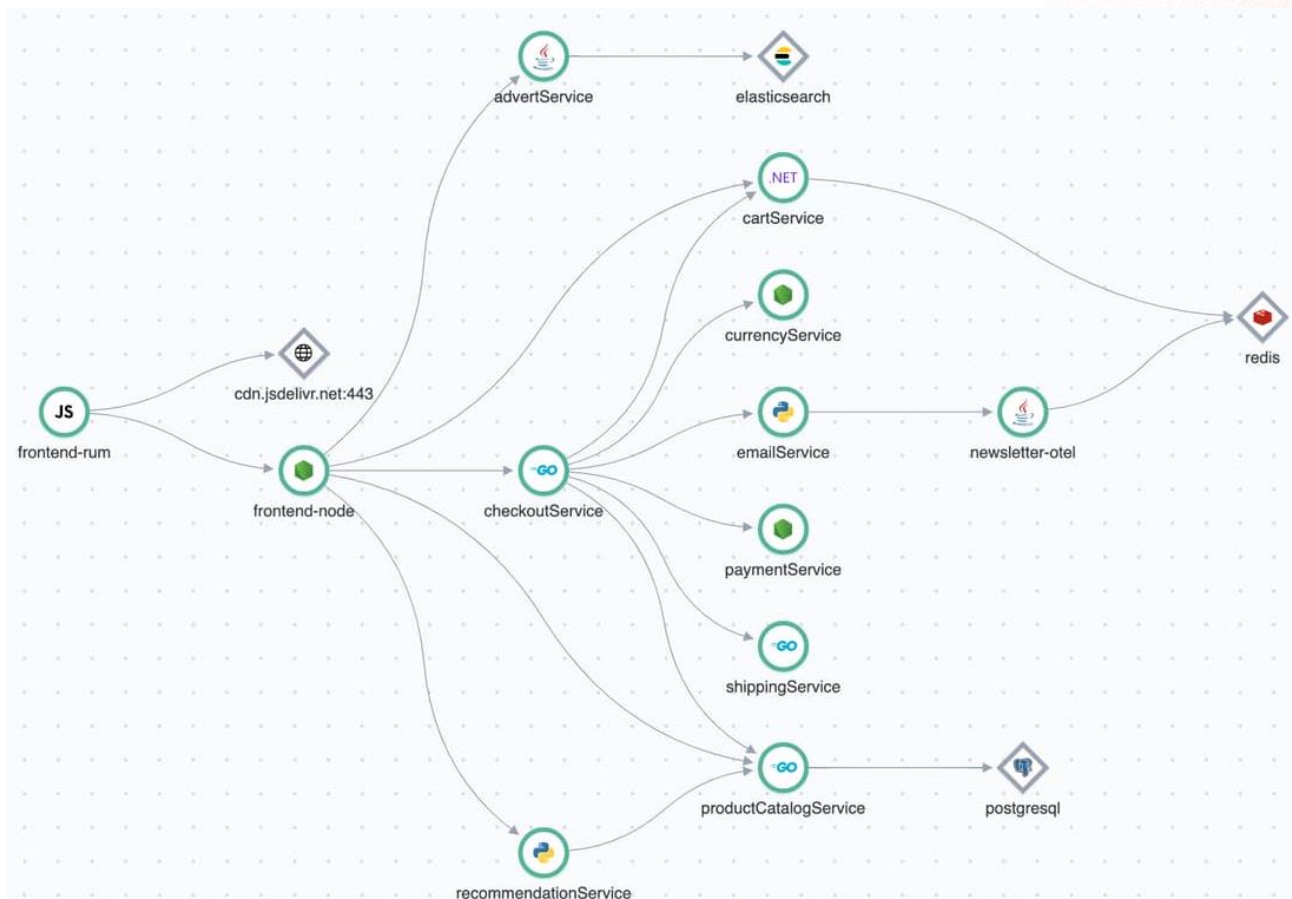
Metadata

Logs

Services ● opbeans-java ● opbeans-python



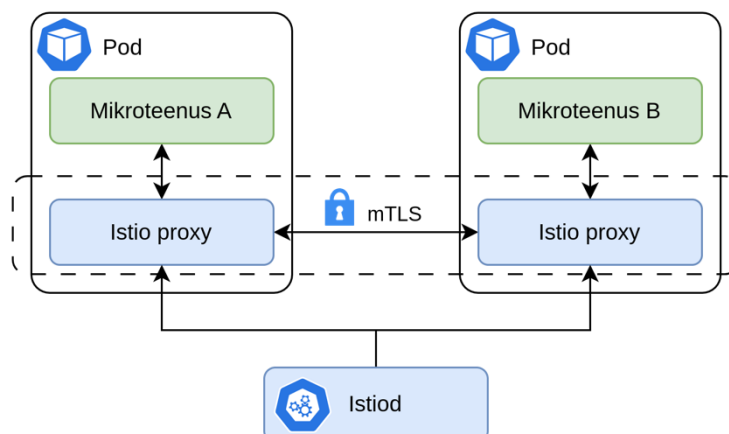
Joonis 15. Jälgitavuse kasutajaliidese näidis Kibanas



Joonis 16. Kibana pooli pakutavad vaated jälgede visualiseerimiseks

Mikroteenuste arhitektuur

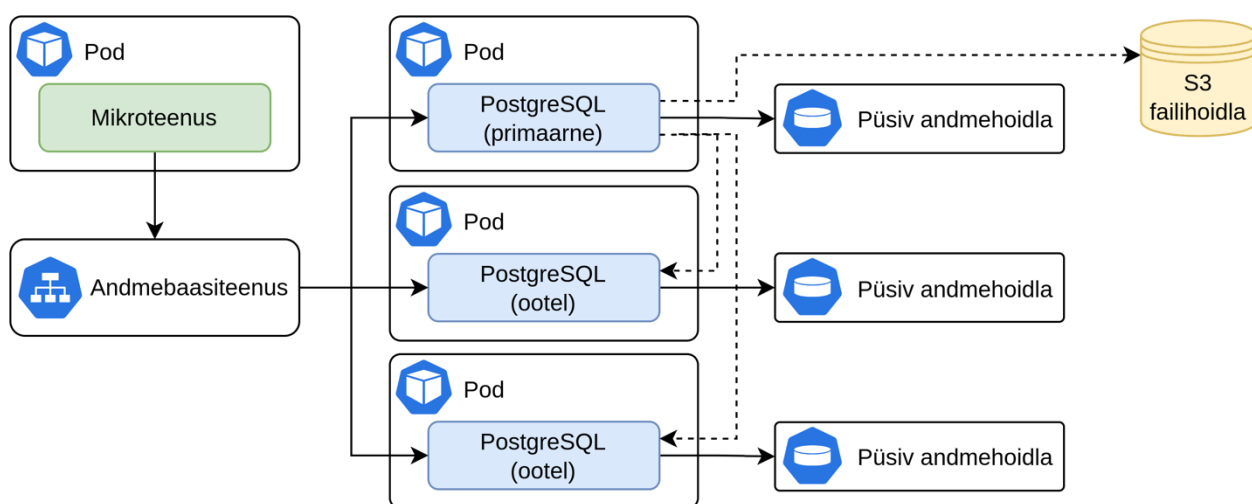
Loodavad mikroteenuseid kasutatakse Kubernetesi klastris, mis asub sisevõrgus. Lähtuvalt [Zero Trust arhitektuuri](#) põhimõtetest ei tohiks aga usaldada ka teenuseid, mis asuvad sisevõrgus. Veendumaks, et nii päringu algataja kui ka vastuvõtja on need, kes nad väidavad end olevat saab kasutada mTLS-i (*mutual TLS*). mTLS on meetod vastastikuseks autentimiseks, kus mõlemad osapooled esitavad TLS sertifikaadi enda õigsuse tõestamiseks ning kogu ülejäänud andmevahetus toimub kasutades juba krüpteeritud sõnumeid. Selleks, et antud loogikat ei peaks igas mikroteenuses eraldi implementeerima on võimalik kasutusele võtta nt [Istio](#), mis selle probleemi Kubernetesi tasemel ära lahendab. Lisaks sõnumite turvalisele andmevahetusele pakub Istio ka kaitselülite funktsionaalsust (loe lähemalt peatükist [Võrguprobleemid](#)). Alljärgnevalt on välja toodud, kuidas Istio ja mikroteenused Kubernetesi klastrisse sobituvad.



Iga mikroteenus jookseb oma konteineris, mis omakorda asub Podis. Istio lisab igale Podidele juurde Istio *proxy* teenuse, mis hakkab vahendama päringuid teenuste vahel. Vajalike võtmete ja seadistuste haldamise ning jagamisega tegeleb Kubernetesesse lisatav laiendus Istiod. Selline lahendus võimaldab lisada mTLS-i toe ilma, et mikroteenused sellest ise teadlikud peaksid olema. Istiot kasutavad Kubernetese klastris mitmed riigiasutused nagu nt Riigi Infosüsteemide Amet ning Regionaal- ja Põllumajandusministeerium.

Pea iga mikroteenusega on seotud oma andmebaas. See omakorda tähendab, et mikroteenuse pidevalt tööl olemiseks peab ka andmebaas kogu aeg kättesaadav olema. Rakenduste puhul jooksutatakse neid tihti peale mitme öla peal võimaldamaks suunata päringuid töötava instantsi pihta. Andmebaaside puhul on lahendus veidi keerulisem, kuna nendega on seotud andmed, mida peab mitme instantsi vahel sünkroonis hoidma. Samuti peab andmeid iga teatud aja tagant varundama võimaldamaks nende taastamist kõikide andmebaaside hävinemise korral.

Arhitektuuri visiooni dokumendis on soovitatud majutada andmebaasid eraldiseisvates virtuaalmasinates. Kubernetese alguaegadel oli see levinud praktika. Tänapäevaks on aga välja töötatud mitmeid lahendusi haldamiseks andmebaase Kubernetese klastris võimaldades hallata kõiki süsteemi tööks vajalikke komponente ühes kohas. Siinkohal tuleb appi [CloudNativePG](#), mis on mõeldud PostgreSQL andmebaasisüsteemi kõrgkäideldavana jooksutamiseks Kubernetese klastris. CloudNativePG lihtsaks ja kiireks kasutuselevõtuks on loodud vastav [Helm chart](#).



Joonis 18. CloudNativePG tööpõhimõtte Kubernetes platvormil

Kõrgkäideldavuse tagamiseks loob CloudNativePG Operaator mitu andmebaasi instantsi, millest üks on primaarne ja teised ootel olekus. Kui primaarse andmebaasiga peaks midagi juhtuma ning see pole enam kättesaadav, siis muudetakse üks ootel olev andmebaas primaarseks ja mikroteenus saab ilma tõrgeteta oma tööd jätkata.

Andmete lisamise ja muutmise päringud suunatakse primaarse baasi vastu, mis omakorda replikeerib andmemuudatused edasi ootel olevatele andmebaasidele tagades, et kõikides andmebaasides on sama seis. Koormuse jaotamiseks suunatakse lugemisoperatsioonid ootel olevate andmebaaside vastu. Andmete varundamiseks saadab CloudNativePG WAL (*Write Ahead Log*) logid iga teatud aja tagant S3 andmehoidlasse. Talletatud logide pealt on võimalik taastada andmebaasi varasem seis kindlal ajahetkel.

3.3.6 Menetlusprotsesside loomine

Analüüsi käigus tuvastatakse taotluste ja menetlusprotsesside tüüpilised sammud ja funktsionaalsused ning selle põhjal luuakse taaskasutatavad plokid (komponendid või mikroteenused), näiteks oleku muutmine, digiallkirjastamine, tagasiside andmine, failide laadimine jms. Need plokid opereerivad taotluse tüübist sõltumatult. Menetlusprotsesside koostamisel kasutatakse protsessimootorit, mis täidab orkestraatori rolli ning kutsub välja vajalikke mikroteenuseid. Tänu sellele saab olemasolevaid protsesse lihtsalt muuta või uusi protsesse kiiresti juurde luua, kombineerides juba olemasolevaid plokkide erineval viisil. Selline plokipõhine lähenemine toetab komponentide ja mikroteenuste sõltumatut asendamist ning täiendamist ilma kogu süsteemi ümber ehitamata. Uue menetlusprotsessi lisamisel tuleb defineerida protsessiskeem, mis kombineerib olemasolevad ja vajadusel ka uued teenused ühtseks tervikuks.

3.3.7 Üleminekuplaan

Suurte süsteemide üleviimine mikroteenusarhitektuurile on pikk ja keeruline protsess. Esmalt on vaja kindlasti veenduda, kas juba loodud lahendused vastavad reaalselt arhitektuuridiagrammidele. Samuti tuleks enne suuremate arendustööde algust koostöös KEMITi arhitektiga üle valideerida, kas olemasolev lahendus on jätkusuutlik või vajab tulevaste arendustöid silmas pidades mõningast eelkorramist.

Enne uute mikroteenuste loomist on vaja aru saada domeenist mida hakatakse üle viima. Selleks analüüsitakse koostöös kliendiga äriprotsesse ning kaardistatakse need. See aitab aru saada ärivaldkondadest ning panna paika mikroteenused ning nende vahelised liidesed (loe lähemalt peatükist [Domeenipõhine disain](#)). Samuti kaardistatakse ära, millised andmed on vaja vanast süsteemist uude üle kanda.

Iga loodava mikroteenuse jaoks luuakse koodihoidlasse oma projekt. Samuti seadistatakse GitLabis rakenduse ehitamise ja tarnimise protsessid. Automatiseeritus aitab hallata suurt hulka teenuseid ning vältida manuaalsetest tegevustest põhjustatud inimlikke vigu.

Funktsionaalsust hakatakse liigutama olemasolevatest monoliitsetest süsteemidest uutesse mikroteenustesse kasutades Strangler Fig mustrit (loe lähemalt peatükist [Monoliidi lammutamine](#)). Samuti luuakse vajalikud andmemigratsiooni skriptid olemas olevate andmete liigutamiseks vanast süsteemist uude. Loodav kood kaetakse automaatsetest aitamaks veenduda koodi korrektsuses ning vältimaks koodi katkiminekut hilisemate koodimuudatuste korral.

TTJA e-teenuste protsessimootori detailanalüüsi käigus tuli välja, et TARVIKu koodi kvaliteet ei vasta headele tavadele ega mikroteenusarhitektuuri põhimõtetele sisaldades liiga pikki meetodeid, koodikordusi, puudulikku testikativust ning liiga suure vastutusala mikroteenuseid. Neid probleeme lahendatakse vajaduspõhiselt, kui konkreetse koodiosaga kokku puututakse. Näiteks lisatakse testid TARVIKu olemasolevatele komponentidele, kus testide katvus pole piisav, enne olemasoleva koodi refaktoreerimist. Samuti tõstetakse mikroteenusesse mittekuuluv loogika välja eraldi teenusesse kasutades eelpool mainitud Strangler Fig mustrit.

Paralleelselt sisuliste teenuste analüüsimise ja arendamisega saab hakata tegelema infrastruktuuri täiendamise ja jälgitavuse täiustamise töödega. Jälgitavuse parandamiseks tuleb lisada mikroteenustele vastav [OpenTelemetry teek](#), lisada APM Server ning seadistada mikroteenused, FluentBit, APM Server ja Kibana telemeetria-andmete saatmiseks ja kuvamiseks. Sellest sõltumatult saab tegeleda CloudNativePG seadistamise ning vajadusel olemas olevates andmebaasides olevate andmete migreerimisega uude andmebaasiklastrisse. Täpsem kirjeldus võimalikest andmebaaside migreerimise võimalustest on kirjeldatud peatükis [Andmebaaside migreerimine](#). Viimase sammuna

infrastruktuuri muudatustes lisatakse Kubernetese klatri koosseisu Istio ning aktiveeritakse automaatne Istio *proxy* lisamine uutesse loodavatesse Podidesse (eeldab uute Podide loomist).

Pakkuja kogemus näitab, et korduvate tegevuste automatiseerimine vähendab töövoos tekkivaid vigu märgatavalt. Seetõttu loome ja kasutame komponentide paigalduseks ja teenuste tarnimiseks koodipõhiseid konfiguratsioone nagu nt Helm *chart*, *kubeconfig*, *.gitlab-ci.yml*, *ansible*, *terraform* vms. See võimaldab samu protsesse korduvalt arenduskeskkonnas järgi proovida vältimaks vigade tekkimist toodangu keskkonnas.

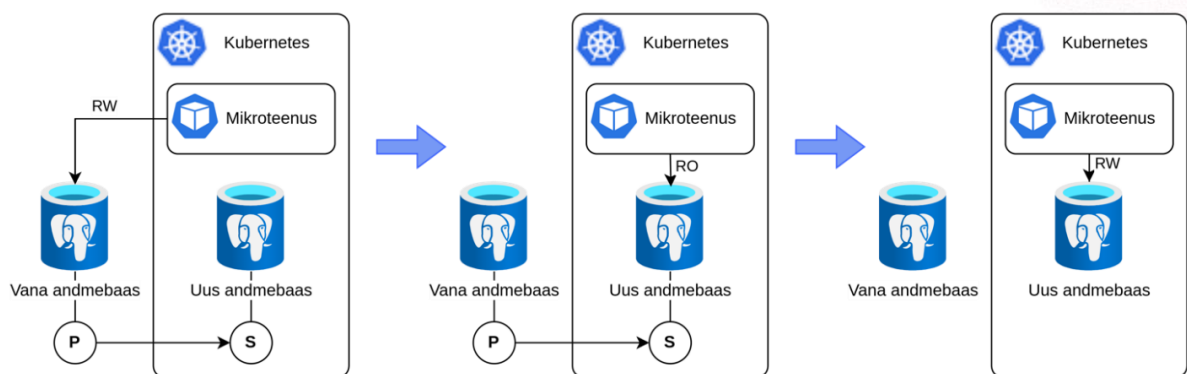
Andmebaaside migreerimine CloudNativePG klastrisse

Sõltuvalt sellest, kas andmebaaside migreerimise hetkeks on TARVIK juba toodangus või mitte ning kui on, siis kui pikalt võib süsteem kasutajatele kättesaamatu olla, tuleb valida andmebaaside migreerimiseks vastavad vahendid. Kui TARVIK ei ole antud hetkeks veel toodangus või kui toodangu keskkond võib olla pool tundi kättesaamatu, saab andmete migreerimiseks CloudNativePG klastrisse kasutada lihtsamat lähenemist tehes ühes andmebaasis varukoopia *pg_dump* abiga ning laadida andmed teise andmebaasi *pg_restore* abiga.

Kui andmebaasi migreerimine peab käima aga kasutajate jaoks võimalikult märkamatu, siis tuleb kasutada loogilist replikeerimist. Loogiline replikeerimine on meetod andmeobjektide ja nende muudatuste liigutamiseks. Loogiline replikeerimine kasutab *publish-subscribe* mudelit, kus üks või mitu tellijat (*subscriber*) tõmbavad andmeid ühelt või mitmelt avaldajalt (*publisher*). Tabeli loogiline replikeerimine algab hetketõmmise tegemisega ning selle tellijale saatmisega ning seejärel jooksvalt ülejäänud muudatuste, mis avaldaja poolel andmekoosseisus pärast hetketõmmise tegemist toimunud on, saatmisega.

Loogilise replikeerimise eeldused ja tegevused:

- Kuberneteses jooksevad migreeritud rakendused, mis on suunatud vana andmebaasi vastu
- Uus andmebaasiklaster töötab ja pääseb ligi vanale andmebaasile.
- Uues andmebaasis on rakenduse jaoks loodud 2 kasutajat. Esimene lugemise ja kirjutamise õigusega (RW) ja teine ainult lugemise õigusega (RO).
- Vanas andmebaasis luuakse PUBLICATION, uues andmebaasis luuakse SUBSCRIPTION vanas andmebaasis loodud PUBLICATION-i vastu
- Veendutakse, et andmete replikeerimine töötab. Replikeerimise ajal ei tohi teised rakendused uude andmebaasi midagi kirjutada, et ei rikutaks andmete terviklikkust.
- Muuta rakenduste andmebaasiühenuse parameetreid nii, et päringud lähevad uue andmebaasi vastu, kasutades üksnes lugemisõigusega kasutajat (RO).
 - Praegusest hetkest alates ei saa toodangu andmebaasi midagi kirjutada.
- Oodata, et pärast replikeerimise algust tehtud muudatused jõuaks samuti uude andmebaasi.
- Muuta rakenduste andmebaasikasutaja lugemis- ja kirjutamisõigusega kasutaja vastu (RW).
 - Nüüd saab toodangu keskkonnas uude andmebaasi kirjutada.
- Uuest ja vanast andmebaasist tuleb eemaldada PUBLICATION ja SUBSCRIPTION.



Joonis 19. PostgreSQL andmebaasi loogiline replikeerimine

Antud lahenduse puhul toimib lugemine kogu aeg. Kirjutamine on aga lühikest aega võimatu. Üksnes lugemisõigusega kasutajat kasutatakse selleks, et rakendus oleks pidevalt kättesaadav aga andmebaasi ei oleks võimalik andmete terviklikkuse tagamiseks võimalik kirjutada vältimaks olukorra, kus uude baasi lisatakse kirje, kuid vanast baasist pole viimased kirjeid veel üle tulnud.

Pärast antud tegevuste lõpetamist kasutavad süsteemid uut andmebaasiklastrit ning vana andmebaasi võib eemaldada.

3.3.8 Riskid ja maandamismeetmed

Alljärgnevalt on välja toodud mikroteenusarhitektuurile üleminekuga seotud riskid ja maandamismeetmed.

Risk	Tõenäosus	Mõju	Maandamismeetmed
Süsteemi keerukuse kasv	Kõrge	Kõrge	Võimalikult palju tegevusi automatiseeritakse vähendamaks manuaalsetest tegevustest tekkida võivaid vigu. Süsteemi kirjaldamiseks luuakse ka põhjalik dokumentatsioon.
Võrguprobleemid (latentsus, teenustevahelised tõrked)	Keskmine	Kõrge	Päringute ajapiirangute kehtestamine, kaitselülitite rakendamine, põhjalik monitooring vigade kiireks ja täpseks tuvastamiseks.
Andmete korrektsus, mis on hajutatud üle mitme teenuse	Keskmine	Kõrge	SAGA-mustri kasutamine hajusate transaktsioonide haldamiseks. Korrektne teenuste vaheliste piiride leidmine vältimaks seotud andmete hajutamist üle mitme teenuse.
Keeruline testida rakendust üle mitmete teenuste	Keskmine	Keskmine	Mikroteenuste vahelise suhtluse testimine kasutades lepingute põhilist testimist, kus teenused lepivad kokku, milliseid andmeid vahetatakse.
Tarnimise keerukuse kasv	Keskmine	Keskmine	Paigaldusprotseduuri automatiseerimine.
Keeruline leida vigu	Kõrge	Keskmine	Kasutusele võetakse vastavad logimise ja jälgitavuse tööriistad, mis agregeerivad andmed tsentraalselt kokku pakkudes kiire ja lihtsa ülevaate süsteemis toimuvast.

3.4 Modelleerimismeetodi BPM tööprotsesside ja protsessimootori tehniline rakendamine

Levinud lähenemine on kirjeldada ärianalüüsi käigus äriprotsesside diagrammid, mis on üheks sisendiks arendajatele. Sellisel lähenemisel on aga mitmeid puudusi. Näiteks ei saa veenduda, et arendatav protsess vastaks diagrammil esitatule. Samuti võib protsess aja jooksul muutuda, kuid muudatusi ei viida sisse diagrammile. See omakorda võib tekitada arenduse ja äripoolel erineva arusaama antud protsessist.

Siinkohal tuleb appi standardiseeritud *Business Process Model and Notation* (BPMN), mis võimaldab äriprotsessi vastavalt kindlatele reeglitele kirja panna ning hiljem seda sama protsessi rakenduse poolel käitada. Sellisel viisil tagatakse, et kirja pandud protsess on kõikidele arusaadav ning täpselt see, mida rakendus oma töö ajal kasutab.

3.4.1 Protsessimootori valik

BPMN-is kirjeldatud protsesside jooksutamiseks on olemas palju erinevaid protsessimootoreid, mille vahel valida. 2024. aastal teostas Helmes TTJA-le analüüsi, mille käigus võrreldi erinevaid protsessimootoreid ja nende võimekusi TTJA poolt esitatud nõuete vastu. Nende seast tõusis enim esile Camunda, mis kattis suurima osa nõutud vajadustest. Samuti tasub mainida, et Camunda on ennast tõestanud ka teistes riigi infosüsteemides, nt TEHIKu poolt arendatavas [Tervise Infosüsteemis](#).

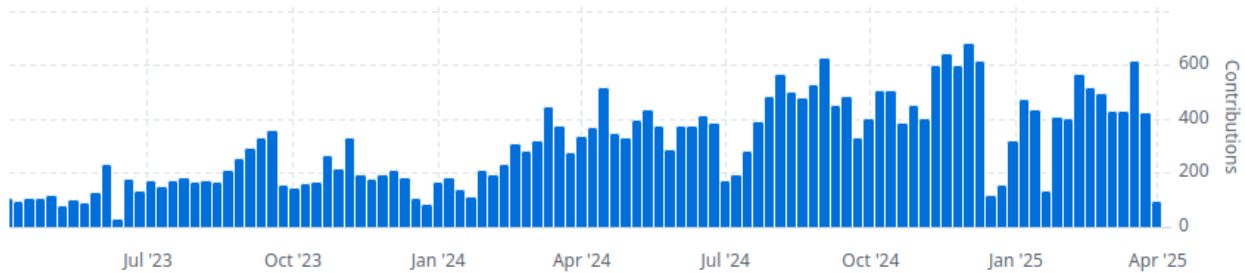
Teine oluline komponent protsessimootori valikul on aga hind. Kui Camunda 7 oli tasuta kasutatav ning sisaldas endas nii protsessimootorit kui ka haldusliidest, siis Camunda 8 puhul on protsessimootor (Zeebe), haldusliides (Operate) ja diagrammide loomise vahend (Web Modeler) lahku löödud. Zeebe on endiselt tasuta kasutatav, kuid teiste komponentide litsentsi eest tuleb maksta. Haldusvahendi alternatiivina on vabalt kätte saadav [Zeebe Simple Monitor](#), mida kommuuni poolt pidevalt arendatakse ja täiendatakse ning protsessidiagramme saab luua ka Desktop Modeleris, mis on samuti tasuta kättesaadav. Lisaks aegub Camunda 7 tugi 2027. aasta aprillis. Tasuliste komponentide kasutamisel läheks ainuüksi litsentsikuludeks 141 800 € aastas. Riigi üks eesmärkidest on aga leida [kokkuhoiukohti](#). Selle raha saaks suunata süsteemide arendusse ja hooldusesse, et pakkuda veelgi enam lisandväärtust. Vaatamata 2024. aasta analüüsis toodud soovitusel kasutada tasulist lahendust arvame eelpool välja toodud põhjenduste tõttu, et mõistlikum on kasutada tasuta pakutavat lahendust. Seetõttu keskendume edaspidises visioonikirjelduses tasuta komponentide kasutamisele.

Kolmas oluline komponent protsessimootori valikul on selle hallatavuse lihtsus ja seda nii paigalduse kui ka monitooringu poole pealt. Camundal on välja töötatud [Helm chart](#), mille abil saab kogu vajaliku tarkvara lihtsasti Kubernetese klastrisse paigaldada, sobitundes KeMITi IT-profiiliga. Helm chart sisaldab kõiki Camunda poolt pakutavaid komponente, kuid *.enabled*-võtmega saab määrata paigaldamiseks vaid tasuta komponente. Samuti pakub see sisse ehitatud lahendusi [jõudluse jälgimiseks](#). Vaikimisi suudab Zeebe anda välja meetrika infot Prometheusi formaadis, kui ka saata andmeid standardiseeritud OTLP protokolliga, mida suudavad tarbida mitmed monitooringuandmete kogumise ja visualiseerimise tööriistad nagu nt FluentBit, OTel Collector, Elastic APM Server, Zipkin, Jaeger jpm.

Viimane, kuid mitte väheoluline kriteerium on projekti aktiivsus. Camunda [koodihoidla](#) statistikat vaadates (vt Joonis 20) on näha, et seal toimub aktiivne arendus. Samuti on neil väga põhjalik [dokumentatsioon](#), mis võimaldab arendajatel lihtsasti oma küsimustele vastused saada.

Commits over time

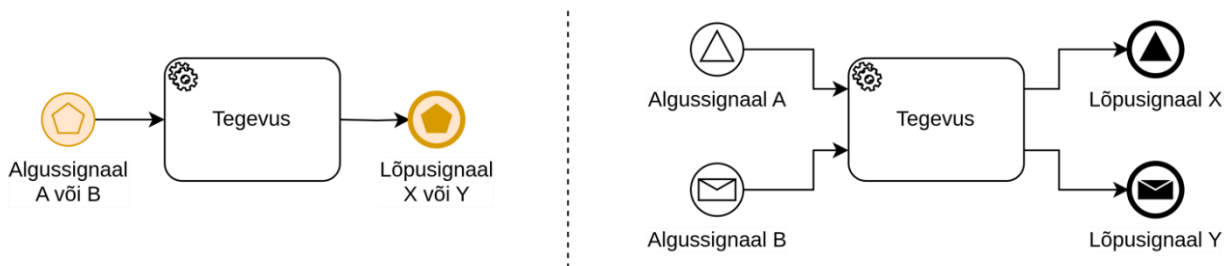
Weekly from Apr 8, 2023 to Apr 6, 2025



Joonis 20. Camunda arendusaktiivsuse graafik

3.4.2 Protsessidiagrammid

Camunda kasutab protsessidiagrammide loomiseks BPMN 2.0 standardile vastavat graafilist esitusviisi, mis on mõeldud äriprotsesside kirjeldamiseks. Toetatud pole küll kõik elemendid, kuid [valdav osa](#). Neid elemente, mida pole toetatud, on võimalik modelleerida teisiti. Nt puudub tugi “multiple”-elemendi jaoks, mille eesmärk on koondada mitmeid erinevaid sündmusi ühe elemendi alla. See teeb mudeli mõnevõrra väiksemaks, aga peidab ära olulist informatsiooni, mis võib kahe silma vahele jääda. Alljärgnevalt on esitatud kaks mudelit, mis täidavad sama eesmärki, aga esimese puhul on kasutatud “multiple”-elementi, kuid teise mudeli puhul on selgelt välja toodud, millised tegevused protsessi algatavad ja lõpetavad. Seega nt “multiple”-elemendi puudumine ei tekita probleeme diagrammide loomisel.



Joonis 21. “Multiple”-elemendi kasutamine ja selle alternatiiv

Lisaks protsessidiagrammidele toetab Camunda ka [DMN-i](#), mis on mõeldud otsustusmodelite kirjeldamiseks. See võimaldab protsessimootoril vastavalt sisendile võtta automaatselt vastu otsus, milliste tegevustega jätkata.

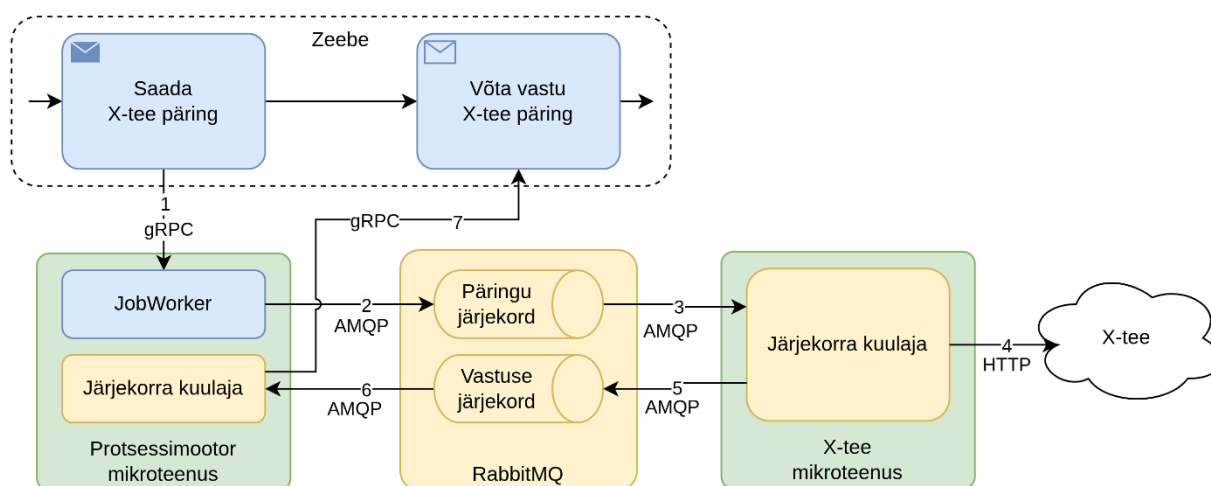
3.4.3 Teenuste vaheline suhtlus

Selleks, et hoida protsessimootor ülejäänud TARVIKu süsteemidest eraldiseisvana, käib kogu suhtlus TARVIKu mikroteenuste ja protsessimootori vahel läbi “Protsessimootori mikroteenus”. Näiteks ei pöördu ükski loataotlusega seotud mikroteenus otse protsessimootori (Zeebe) poole vaid vajaduse korral pöördub see “Protsessimootori mikroteenus” poole, mis omakorda saadab päringu edasi protsessimootorile. Sarnaselt toimub suhtlus ka teistpidi.

Sarnast lähenemist on Pakkaja edukalt kasutanud ka teistes projektides. See aitab vältida protsessimootori-spetsiifilise info lekkimist teistesse teenustesse ning võimaldab vajadusel protsessimootor välja vahetada ilma, et teisi mikroteenuseid muutma peaks. Võimalus oleks lisaks kasutusele võtta ka [Zeebe Connectors](#), et protsessimootor saaks teha otse päringuid välistesse

süsteemidesse, kuid see suurendaks protsessimootori poolset keerukust, suurendaks otsest sõltuvust teistest süsteemidest ning raskendaks automaattestimist. Seetõttu me seda antud lahenduses ei kasuta.

TARVIKu mikroteenused suhtlevad “Protsessimootori mikroteenusega”, kasutades AMQP protokoll. See aitab suurendada süsteemi töökindlust, kuna protsessi loomise hetkel ei pea “Protsessimootori mikroteenus” püsti olema. “Protsessimootori mikroteenus” omakorda suhtleb Zeebe Gateway’ga kasutades gRPC-d. Kuna protsessid oma olemuselt on asünkroonsed, siis juhul, kui protsessimootoril on vaja suhelda teiste teenustega, käib ka see läbi “Protsessimootori mikroteenuse”, kasutades AMQP protokoll. Näiteks kui protsessi käigus on vaja teha X-tee päring, siis pöördub Zeebe protsessimootor “Protsessimootori mikroteenuse” vastava [Job Workeri](#) poole, mis saadab sõnumi läbi RabbitMQ “X-tee mikroteenuse” pihta. Sarnast lähenemist on soovitatud ka Camunda enda dokumentatsioonis [heade tavade](#) all.



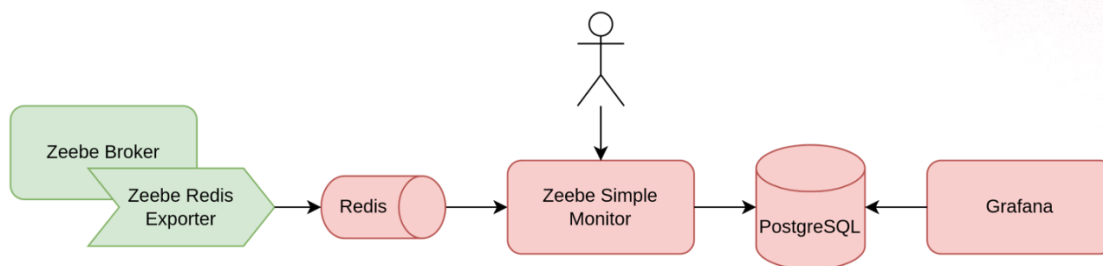
Joonis 22. Protsessi käigus andmete küsimine üle X-tee

3.4.4 Jälgitavus

Jälgitavuse puhul räägime kahest eri tüüpi jälgitavusest. Ühel juhul soovime jälgida, mis seisus protsess parasjagu on. Teisel juhul soovime aga jälgida, kuidas protsessimootor töötab.

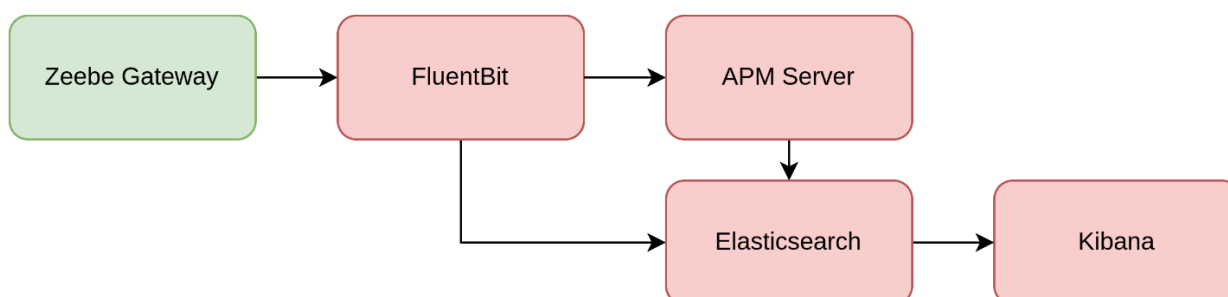
Protsessi jälgimiseks on võimalik kasutada kas Camunda poolt pakutavat tasuta haldusliidest Operate või kommuuni poolt arendatavat teenust Zeebe Simple Monitor. Kuna Operate kasutuskulud on üpris suured, siis visioonis keskendume Zeebe Simple Monitorile. Selleks, et saada infot protsessis toimuvate tegevuste kohta, saadab [Zeebe Exporter](#) info välisesse teenusesse, mis talletab info hilisemaks kuvamiseks. Näitaks saab antud juhul ära kasutada [Zeebe Redis Exporterit](#), mis saadab andmed [Redis Streami](#). Zeebe Simple Monitor omakorda loeb sealt toimunud sündmused sisse ning talletab need relatsioonilisse andmebaasi, nt PostgreSQL-i. Lisaks pakub Zeebe Simple Monitor kasutajaliidest protsesside visuaalseks kuvamiseks ning haldamiseks. Kui mingit infot ei saa Zeebe Simple Monitorist kätte, siis on võimalik lisada juurde Grafana ja teostada spetsiaalsed päringud huvitava info kättesaamiseks.

Nii Redis kui ka PostgreSQL on TARVIKus juba kasutusele võetud komponendid, seega saab olemasolevat tehnilist pinu suures osas ära kasutada.



Joonis 23. Protsessimootoris jooksvate protsesside monitoorimiseks vajalik arhitektuur

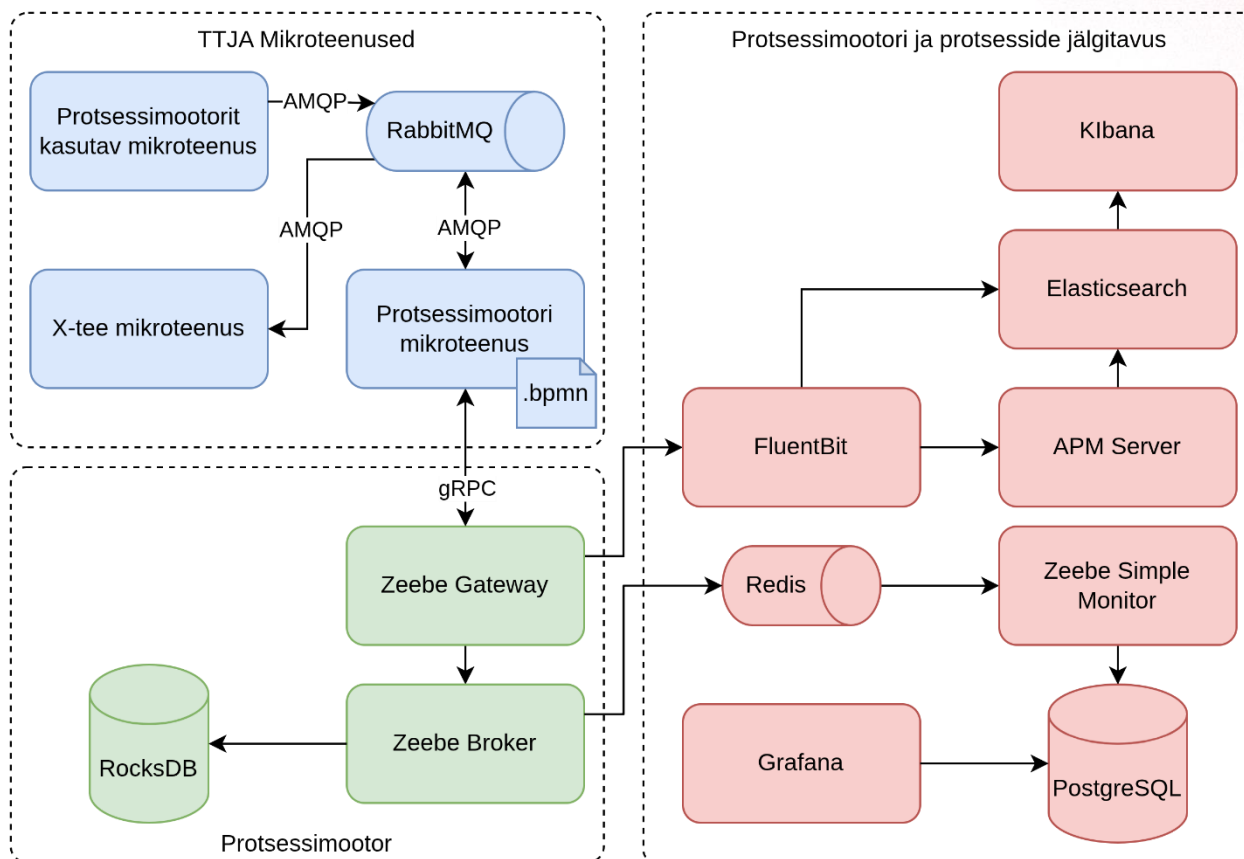
Teine asi, mida tasub jälgida, on protsessimootor ise. Camundas on sisse ehitatud võimekus [pakkuda meetrikaid](#) erinevatel kujudel. Meetrika andmeid talletab Camunda oma mälus ning monitooringu süsteem peab need mingil viisil kätte saama selleks, et neid agregeerida, kuvada ja vajadusel ka otsus langetada (nt kas on vaja saata välja teade probleemi ilmnemisest). Seda saab teha kahel viisil – monitooringu süsteem küsib andmeid (*polling*) või andmed saadetakse monitooringu süsteemile (*pushing*). Andmete küsimiseks on Camundas implementeeritud otspunkt, mis pakub andmeid Prometheusile sobival kujul. Andmete saatmiseks toetab Camunda OTLP protokoll. Selle lahenduse puhul saadab Camunda iga teatud aja tagant andmed FluentBiti, mis suudab andmeid APM Serverile ning Elasticsearchi edastada (loe täpsemalt mikroteenuste [jälgitavuse peatükist](#)). Elasticsearchis olevate andmete põhjal saab Kibanas luua erinevaid raporteid ja teavitusi, mis saadavad hoiatatusi süsteemis tekkivate tõrgete korral. OTLP protokoll kasutamise eelis seisneb selles, et Camunda jaoks ei ole enam oluline, millist spetsiifilist tööriista monitooringuks kasutatakse – Camunda saadab andmed lihtsalt kindla formaadi alusel edasi.



Joonis 24. Protsessimootoris toimuva monitoorimiseks vajalik arhitektuur

3.4.5 Arhitektuur

Lähtuvalt eelmistes peatükkides käsitletud lahendustest joonistub välja alljärgnev arhitektuuridiagramm. Arhitektuuri luues on silmas peetud, et võimalikult palju saaks kasutada ära juba TARVIKu projektis kasutusel olevaid ning mikroteenusarhitektuuriga kasutusele võetud lahendusi lahendusi nagu nt PostgreSQL, Redis, RabbitMQ, FluentBit, APM Server, Elasticsearch, Kibana.



Joonis 25. Protsessimootori koondarhitektuur

Arhitektuuris kasutatavate komponentide kirjeldus:

- Zeebe Gateway – Zeebe klatri komponent, läbi mille toimub suhtlus Zeebe klastriga. Käitub koormusjaoturina ning suunab päringuid edasi Zeebe Brokeritele.
- Zeebe Broker – Zeebe klatri põhikomponent, mis töötleb päringuid, tegeleb replikeerimisega, partitsioneerimisega ja andmete ekspordiga.
- RocksDB - Andmebaasisüsteem, mida Zeebe Broker kasutab sisemiselt andmete talletamiseks.
- Protsessimootori mikroteenus – Mikroteenus, läbi mille toimub suhtlus teiste mikroteenuste ja protsessimootori vahel.
- Protsessimootorit kasutav mikroteenus - Kõikvõimalikud mikroteenused, mis soovivad suhelda protsessimootoriga.
- X-tee mikroteenus – Mikroteenus, mille kaudu saab teha X-tee päringuid
- RabbitMQ - Sõnumiedastusvahend, mis aitab erinevatel rakendustel omavahel suhelda kasutades AMQP protokoll.
- Redis – Mälupõhine NoSQL võti-väärtus andmebaas, mida antud juhul kasutatakse protsessidega seotud sündmuste edastamiseks Zeebe Simple Monitorile.
- Zeebe Simple Monitor – Kommuuni poolt arendatud rakendus BPMN-is kirjeldatud protsessi instantside jälgimiseks ja haldamiseks.
- PostgreSQL – Relatsiooniline SQL andmebaas, kuhu Zeebe Simple Monitor talletab Zeebe Brokerist saadatud protsessidega seotud sündmused.
- Grafana – Andmete visualiseerimise tööriist, millega saab luua graafikuid, tabeleid ja töölaudu. TTJA e-teenuste protsessimootori detailanalüüsis on välja toodud, kuidas Grafanat

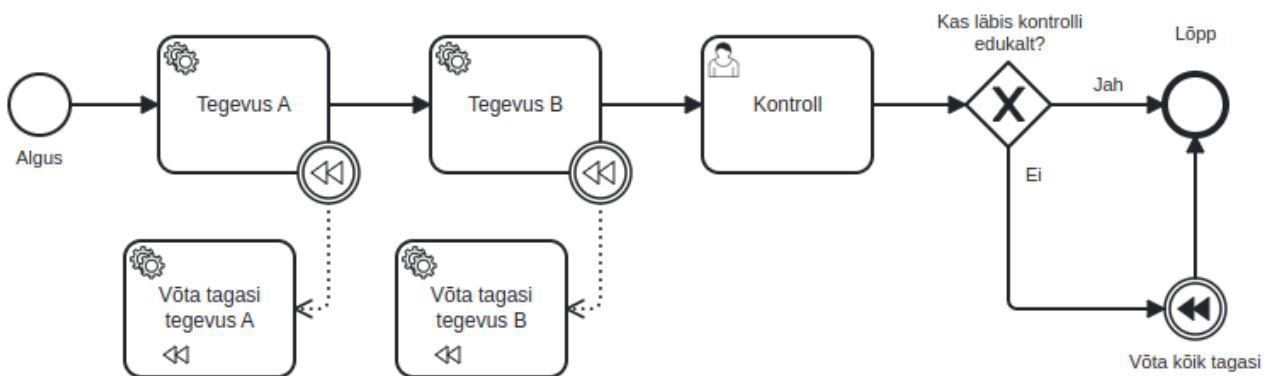
kasutades saab lihtsasti vaadata protsessidega seotud statistikat. Grafana ei sisaldu KEMITI IT-profiilis, kuid raportite loomiseks on mõistlik see kasutusele võtta.

- FluentBit - Tööriist, mis aitab erinevatest rakendustes koguda, töödelda ja edasi saata jälgimisandmeid nagu logid (*logs*), meetrikad (*metrics*) ja jäljed (*traces*).
- APM Server – Teenus, mis võimaldab vastu võtta OTLP protokolliga saadetud andmeid ning talletada need sobival kujul Elasticsearchi.
- Elasticsearch - Andmebaas ja otsingumootor, mis võimaldab efektiivselt otsida ja analüüsida suurtes kogustes andmeid.
- Kibana – Kibana abil on võimalik otsida rakenduste logisid, jälgida päringute vooge läbi hajutatud süsteemi ning visualiseerida meetrikaid. Lisaks on võimalik kirjeldada ära reeglid, millele meetrikad peavad vastama ning mittevastavuse korral saab välja saata [teavitusi](#) võimaldades kiiresti reageerida võimalikele vigadele.

3.4.6 Kompenseerivad tegevused

Protsesside korral tuleb tihti teha muudatusi mitmetes erinevates mikroteenustes, erinevates andmebaasides ja need muudatused võivad toimuda pika aja jooksul. Kui kõik läheb soovitud rada pidi, siis esmapilgul ei tundu probleemi olevat. Kui mõni toiming aga ebaõnnestub, siis võib olla vajadus tehtud muudatused võtta tagasi ka teistes mikroteenustes. Selle jaoks saab kasutada [SAGA](#) mustrit, mida on võimalik disainida ka BPMN diagrammina, mis annab hea ülevaate, kuidas süsteem veaolukordade ja erandite korral käitub.

Camundas saab SAGA mustrit kirjeldada kasutades [kompenseerivaid tegevusi](#). Need aitavad võtta tagasi tegevusi, mis eelnevalt edukalt tehtud said, kuid uue vea ilmnedes enam korrektsed ei ole. Järgnevalt on toodud näide, kus “Tegevus A” ja “Tegevus B” on varasemalt sooritatud, kuid pärast kontrolli ebaedukat läbimist kutsutakse välja kompensatsiooni nõudev sündmus “Võta kõik tagasi”, mille peale käivitatakse kompenseerivad tegevused “Võta tagasi tegevus B” ja “Võta tagasi tegevus A”.

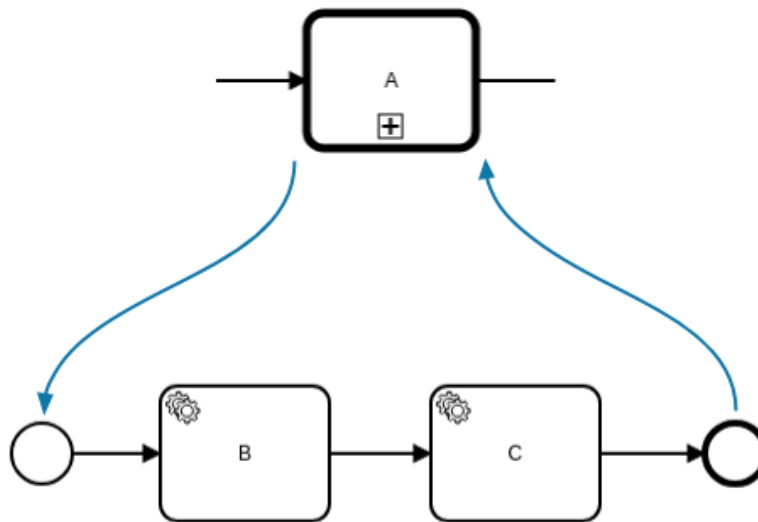


Joonis 26. SAGA mustri kasutamine BPMN diagrammis kompenseerivate tegevuste teostamiseks

3.4.7 Korduvad tegevused

Kogemuse põhjal julgeb Pakkuja väita, et erinevad protsessid sisaldavad tihtipeale sarnaseid tegevusi, olgu selleks siis üldine põhiprotsess, mis sisaldab teenuste põhilisi spetsiifilisi toiminguid, allkirjastamise protsess, mis peab alati järgima sama loogikat olenemata teenusest vms levinud toiming. Sellisel juhul ei ole mõistlik igas erinevas protsessidiagrammis seda loogikat eraldi kirjeldada vaid teha seda ühes konkreetses .bpmn failis ning teistes protsessides sellele viidata.

Selle jaoks on olemas spetsiaalne element [Call Activity](#). Kui Protsess jõuab sammu A, siis luuakse uus protsessi instants. Kui loodud protsessi instants jõuab lõppu (sammud B ja C saavad tehtud), siis viimati loodud protsessi instants lõpetatakse ning eelmine protsessi instants jätkab tööd.



Joonis 27. Alamtegevuste väljakutse BPMN diagrammis

3.4.8 Rakendamine

Kui keskkond on püsti ja protsessidiagramm on loodud, siis on vaja see protsessimootoris sisse laadida. Seda saaks teha soovi korral ka otse Desktop Modeleri kaudu, kuid see pole soovitatav. Selle asemel tuleks loodud mudelid talletada koodihoidlasse, mis võimaldab mudeleid versioneerida ning veenduda, et igasse keskkonda jõuab täpselt sama mudel. Samuti võimaldab see katta loodud mudelid automaattestidega veendumaks, et loogika erinevate stsenaariumite korral paika peab. Seetõttu tuleks diagramm lisada “Protsessimootori mikroteenus”-esse, kus sellele automaattestid saab kirjutada, ning mis koordineeriks mudelite tarnimist kasutades @Deployment annotatsiooni.

```
@Deployment(resources = {
    "classpath:bpmn/juurprotsess.bpmn",
    "classpath:bpmn/allkirjastamine.bpmn",
    "classpath:bpmn/loataotlus.bpmn",
})
```

Joonis 28. BPMN mudelite tarnimine kasutades @Deployment annotatsiooni

Uue protsessi käivitamiseks on vaja luua diagrammi põhjal uus instants. Selleks tuleb protsessimootorile teada anda, millise protsessi id-ga protsessi soovitatakse käivitada. Protsessi ID on ära määratud .bpmn failis ning peab olema unikaalne üle kõikide protsesside. Protsessi instantsi loomise järel saadud protsessi instantsi id on soovitatav talletada andmebaasi ning siduda see kirjega, mille põhjal instants loodi. See võimaldab näiteks vajadusel lihtsasti üles leida mis seisus mingi konkreetne taotlus parasjagu on.

```
<bpmn:process id="loataotlus" isExecutable="true">
```

Joonis 29. BPMN diagrammi lähtekoodis olev viide protsessi identifikaatorile

3.4.9 Üleminekuplaan

Protsessimootori kasutuselevõtt hõlmab mitmeid tegevusi nii infrastruktuuri kui ka rakenduse poole pealt. Järgnevalt käime läbi etapid, mida on vaja teostada selleks, et protsessimootor TARVIKus kasutusele võtta. Antud juhul lähtub Pakkuja visioonist, kus kasutusele võetakse Zeebe Simple Monitor, et hoida kulud minimaalsena.

Sõltumata konkreetsetest äriprotsessidest saab koheselt pihta hakata protsessimootori ning sellega seotud teenuste püstipanekuga. Selleks on vaja kasutusele võtta [protsessimootori arhitektuuri](#) peatükis välja toodud uued komponendid, mida üldises mikroteenusarhitektuuri koosseisus pole veel paigaldatud: Zeebe Gateway, Zeebe Broker, Zeebe Simple Monitor, Grafana. Kuna tegu on üldiste komponentidega, mille paigaldamine ei eelda eriarendusi, siis saab sellega tegeleda nii KeMITi poolne IT, kui ka Pakkuja.

Paralleelselt paigaldustöödega on võimalik täiendada Zeebe Simple Monitori funktsionaalsust (protsessi sammude tagasivõtmine, instantside üleviimine järgmisesse versiooni, rollide haldamine) ning töötada välja vajalikke raporteid Grafana-s. Siiski soovitab Pakkuja Tellijal kriitiliselt üle vaadata, mis funktsionaalsus on reaalselt vajalik, et vältida asjatuid arenduskulusid.

Eelnevatest tegevustes sõltumatult saab hakata välja arendama "Protsessimootori mikroteenust". See hõlmaks endas mikroteenuste ja protsessimootori vahelise API kirjeldamist ja implementeerimist, äriprotsesside tarnimisloogika juurutamist, "X-tee mikroteenuse" liidestamist. Kuna tegu on TARVIKu süsteemis ühe keskse komponendiga, tuleb kogu teenus katta põhjalikult automaattestidega.

Protsessimootorit on võib kasutada nii protsesside puhul, mis võivad kesta kaua, nt toimub selle menetlemine kui ka lühemate protsesside puhul, mis koosnevad mitmest sammust, võimaldades anda hea ülevaate sammudes toimuvast. Kõigepealt tuleb analüüsi käigus koostöös Tellijaga kaardistada tegevused, mille jaoks on protsessimootorit mõistlik kasutada. Leitud tegevused tuleb seejärel modelleerida kasutades BPMN notatsiooni. Loodud BPMN diagrammid on sisendiks arendajale.

Vastavalt sisenddiagrammile ja analüüsidokumendile loob arendaja vastava teenuse. Vajadusel täiendab arendaja loodud BPMN diagrammi, lisades väljakutsed teistele teenustele. Korduvad tegevused tõstetakse eraldi BPMN diagrammidesse ning nendele viidatakse sisenddiagrammidest. BPMN diagrammid tarnitakse koos teenusega. Sarnane arendusprotsess toimub ka äriprotsesside muutmise korral.

3.4.10 Riskid ja maandamismeetmed

Alljärgnevalt on välja toodud protsessimootori kasutuselevõtiga seotud riskid ja maandamismeetmed.

Risk	Tõenäosus	Mõju	Maandamismeetmed
Arhitektuuri keerukuse kasv	Keskmine	Keskmine	Põhjalik dokumentatsioon, mis annab ülevaate protsessimootori tööst ja põhimõtetest.
Keeruline leida vigu	Kõrge	Keskmine	Kasutusele võetakse vastavad logimise ja jälgitavuse tööriistad, mis agregeerivad andmed tsentraalselt kokku pakkudes kiire ja lihtsa ülevaate süsteemis toimuvast.
Tööprotsesse ei saa jooksutada, kui	Keskmine	Kõrge	Protsessimootori jooksutamine mitme öla peal suurendamaks jõudlust ja kättesaadavust.

protsessimootor pole kättesaadav			
Integreerimise keerukus olemas oleva süsteemiga	Keskmine	Keskmine	Alustada lihtsamatest protsessidest. Refaktoreerida vajadusel olemas olevat koodi, et see sobituks uue arhitektuuriga.
Kogu protsessi testimine on keeruline	Keskmine	Keskmine	Testitakse üksnes protsessi. Suhtlus teiste väliste teenustega <i>mockitakse</i> .
Aktiivsete protsesside muudatused	Keskmine	Keskmine	Camundasse sisse ehitatud protsessi versioneerimise kasutamine. Migreerimisstrateegia paikapane ja dokumenteerimine.

3.5 Mikroesitluskihid

TARVIKu visiooni järgi koondatakse TTJA, JVIS ning NBA süsteemide funktsionaalsus, sh majandustegevuslubade ja -teadete kuvamine, taotlemine ning menetlemine aga ka administreerimine jt alamsüsteemid ühele platvormile, et

- parandada kasutajakogemust,
- lihtsustada menetlejate igapäevatööd,
- vähendada praegusest mitme süsteemi ülalpidamisest tulenevat halduskoormust.

Süsteemide liitmisel on aga oht monoliitrakendustele omastest tihedatest seostest tulenevate riskide realiseerumisele, näiteks:

- ühe (klienditoode) muutmine eeldab teiste (menetlustoode) muutmist,
- ühe toote muudatuste testimine eeldab ka teiste toodete testimist,
- üht toodet pole võimalik tarnida teisi tooteid tarnimata,
- admintoote tehnoloogiaid ei ole võimalik valida sõltumatult kliendi- ja menetlustoote tehnoloogiatest.

Eelpool nimetatud probleemide leevendamiseks pakume visioonis mikroesitluskihtidel (*micro front-end*) põhinevat arhitektuuri. **Mikroesitluskihtidega** on võimalik rakenduse esitluskiht jagada komponentideks selliselt, et **iga toode on** teistest **sõltumatu** ehk on **avatud muudatustele, eraldi testitav** ja **tarnitav** ning **tehnoloogilistes valikutes vaba**.

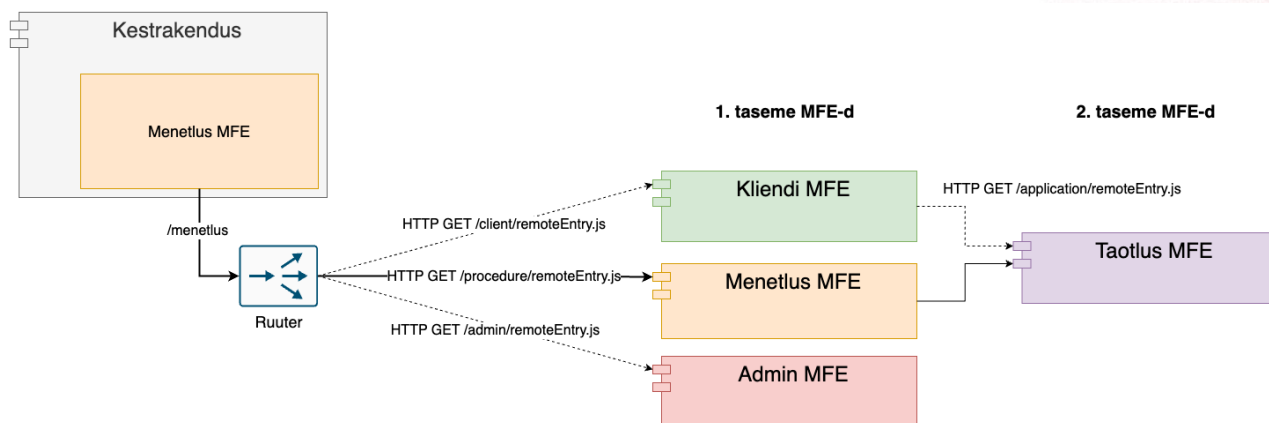
3.5.1 TARVIKu mikroesitluskihtide üldarhitektuur

Pakutav mikroesitluskihtidel (MFE-del) põhinev üldarhitektuur koosneb loodavast kestrakendusest ning igale tootele vastavast MFE-st.

Kestrakendus (*shell/host*) on vajalik MFE-de **orkestreerimiseks**. Selleks tuleb kõik loodavad (1. taseme) MFE-d registreerida kestrakenduses teekondadena (*route*). Samuti soovitame koondada võimalikult palju kesksel funktsionaalsust nagu autentimine, rakenduse keele seadistus jms kestrakendusse.

MFE on **autonoomne ärioloogiline** komponent, millel on **selgelt piiritletud äridomeen**. Näiteks, klienditoote, menetlustoote ja admintoote ärifunktsionaalsus ei tohiks omavahel põimuda üheks MFE-ks, sest sel juhul ületataks selgelt äridomeenide piirid, küll aga sobib neid esitada eraldi mikroesitluskihi komponentidena:

- Kliendi MFE,
- Menetlus MFE,
- Admin MFE.

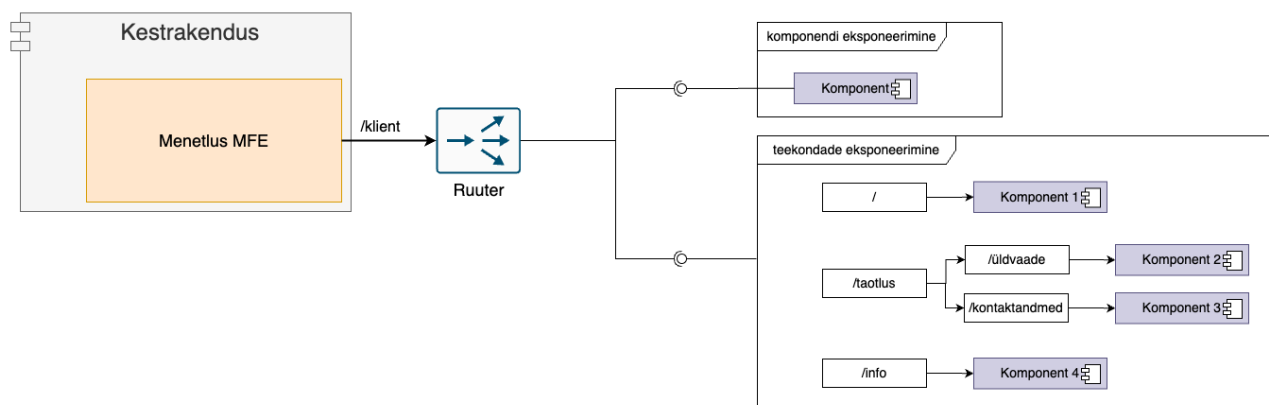


Joonis 30. TARVIKu portaali kestrakendus ja MFE-deks jagunemine

Iga MFE on võimeline orkestreerima alam-MFE-sid. Nii on võimalik komponente **taaskasutada**. Näiteks, Kliendi MFE ja Menetlus MFE mõlemad võivad tarbida Taotlus MFE-d, et kliendi taotluse vaadet kuvada. 2. ja kõrgemate tasemete MFE-de loomisel soovitame aga olla ettevaatlik, eriti kui MFE kasutamine ilma sisendparameetriteta pole võimalik. MFE-des sisendparameetrite kasutamine loob jätku seoseid komponentide vahel, mis vähendab nende autonoomsust.

Näiteks, Taotlus MFE taotluse id parameetri nime või andmekuju muutumisel ei ole võimalik Taotlus MFE-d tarnida ilma, et tehtaks muudatusi ka tarbivates Kliendi MFE-s ja Menetlus MFE-s. Selle asemel soovitame primitiivseid sisendandmeid lugeda URL-ilt, sest teekondade (*path*) kuju on võimalik Taotlus MFE-st juhtida.

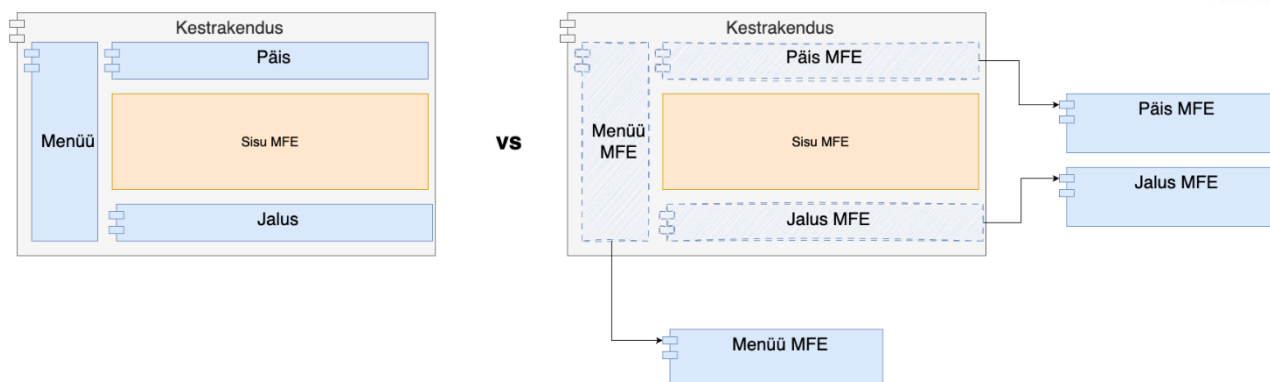
MFE-sid on võimalik laadida nii teekondade kui ka [iseseisvate komponentide](#) eksponeerimise kaudu. Näiteks, 1. taseme MFE-sid on otstarbekas eksponeerida teekondadena, sest võivad koosneda mitmest alamlehest. Kui MFE-d on vaja kuvada aga mõne teise MFE sees (näiteks Taotlus MFE Kliendi MFE sees), siis võib piisata iseseisva komponendi eksponeerimisest. Sellegipoolest, soovitame võimalusel alati **eelistada [teekondade eksponeerimist](#)**, sest nii on edaspidi MFE rohkem avatud funktsionaalsuse laiendamisele.



Joonis 31. Komponenti ja teekondade eksponeerimine

Kuigi **päist, vasakmenüüd ja jalust** on samuti võimalik MFE-deks jagada, soovitame siiski TARVIKu puhul need **lisada otse kestrakendusse**. Päise, vasakmenüü ja jaluse eraldi arendamise ja

tarnimise võimalus suurt eelist ei anna, seega pakutud lahendus aitab vältida liigset keerukust ning hoida tarnitavate esitluskihi rakenduste arv madalamana



Joonis 32. Päis, menüü ja jalus kestrakenduses või MFE-dena

Täpne MFE-deks jagunemine ning granulaarsus selgub analüüsi tulemusena, seda on täpsemalt kirjeldatud peatükis „4. Nõuete ja skoobi täpsustamine“.

3.5.2 Tehnoloogiad

Järgnevalt kirjeldame erinevaid MFE-de loomiseks võimalikke tehnoloogilisi lahendusi.

Iframe ehk sisepaneel on pilt-pildis põhimõttel toimiv küps tehnoloogia. *Iframe* sobib hästi kolmandate osapoolte sisu kuvamiseks MFE-na, kuid meie hinnangul **TARVIKusse baaslahendusena ei sobi**, sest autentimise, stiilide, sõltuvuste jms jagatud sisu propageerimine on keeruline. Samuti on *iframe*-id otsingumootorivaenulikud ning ka WCAG nõuete täitmine on raskendatud.

Web Components tehnoloogia sobib hästi kolmandate osapoolte sisu kuvamiseks, kus tarbijal on vaja kontrolli sisu üle. Siiski, meie hinnangul **TARVIKusse baaslahendusena ei sobi**, sest ei võimalda jagada sõltuvusi, mis võib muuta portaali komponentide laadimise kulukaks. Samuti eeldab nende kasutamine enamasti versioneerimist, mis tooks kaasa vajaduse kestrakenduses käsitsi iga muudatuse peale versioone muuta ja kaoks võimalus komponente autonoomselt tarnida.

Webpack Module Federation sobib meie kogemusel suurepäraselt **TARVIKusse MFE-de esitamiseks**. Toetudes *module federation* kogemusele eesti.ee riigiportaalis, kirjeldame selle peamiseid eeliseid:

- *Module federation* võimaldab MFE-sid arendada ja tarnida muust äriefunktsionaalsusest sõltumata ning **ei eelda versioneerimist**.
- *Module federation* toetab [sõltuvuste jagamist](#) (näiteks Angular) kestrakenduse ja MFE-de vahel, kuid ei nõua seda. Samuti toetab Webpack [laiska laadimist](#) mida on võimalik kombineerida Angulari teekondade laiska laadimisega ning hoida seeläbi aktiivne rakendus võimalikult õhuke.
- Webpack Module Federation on [laialt levinud, hästi hooldatud ning küllaltki küps](#) tehnoloogia, mille kohta leidub ohtralt [dokumentatsiooni](#), [koodinäiteid](#) ning [juhtumiuurimusi](#), eriti Angular lahendusteks.

Webpack Module Federationi peamine **puudus** on otsene sõltuvus Webpackist ning küllalt suur keerukus, mille peamisi väljakutseid käsitleme peatükis „Väljakutsed“.

Webpack Module Federationi **juurutamiseks** on võimalik seda kasutada kas [otse](#) (1) või kolmandate osapoolte pistikute kaudu, neist enim levinud on [Nx](#) (2) ja [Angular Architects](#) (3). Korrektses seadistuses korraldab Webpack Module Federation risti kõigi eelnevalt mainitud kolme lahenduse vahel, kuid TARVIKus soovime piirduda neist ühega. Meie kogemusel pistikprogrammide kasutamine lihtsustab *module federationi* seadistamist oluliselt võrreldes selle otse kasutamisega. Soovime *mono-repodega* valida Nx ning *multi-repodega* Angular Architects lahendus. **Webpack Module Federationi kasutamine on eelduseks eesti.ee portaaliga liidestumiseks.**

[Native Federation](#) on oma olemuselt väga sarnane Webpack Module Federation lahendusele, kuid ei eelda Webpack kasutamist. Varasemalt eesti.ee portaalile teostatud tehnilise analüüsi põhjal jõudsimme aga järeldusele, et **tehnoloogiline sõltumatus ei kaalu hetkel üle suuri puuduseid**:

- vigaderohked ja ebastabiilsed toodangu versioonid,
- vähe dokumentatsiooni,
- kestrakenduses on nõutud kõikide tasemete MFE-de registreerimine (mitte ainult esimese, nagu Webpack Module Federationiga).

3.5.3 Andmevahetus

Hoolikalt valitud jaotus MFE-deks tagab süsteemi komponentide maksimaalse autonoomsuse. Siiski mõningane andmevahetus kestrakenduse ja MFE-de vahel on vältimatu, mõnel juhul isegi eelistatud. Tagamaks head kasutajakogemust, soovime luua **kestrakendusse üldise veahalduse**. Selliselt on võimalik keskselt lahendada juhud, kus MFE või MFE toimimiseks hädavajalike andmete laadimine ebaõnnestub, puuduvad õigused sisu nägemiseks jms veaolukorrad. Keskse veahalduse kasutamine aitab lihtsustada veahalduses muudatuste tegemist ka tulevikus.

Selleks on vaja luua võimalus andmete edastamiseks MFE-de ja kestrakenduse vahel. Järgnevalt kirjeldame veebibrauserite reaktiivseid tehnoloogiaid, nende puudusi ning pakume lahenduse.

Redux, Angular Service ja **RxJS** on levinud tehnoloogiad Angular rakendustes andmete hoidmiseks ja voogedastamiseks, kuid mikroesitlustkihtidele üle viidud (MFE-statud) rakenduses eeldab see *singleton* instantside kasutamist, mis võib hilisemalt seada liigselt piiranguid ning kõrget sidusust konkreetsete tehnoloogiatega, seetõttu **ei soovita** me sellist lahendust.

Brauseri sündmused (Event) ja **PostMessage API** on raamistike jm tehnoloogiate suhtes agnostilised ning küllaltki lihtsad kasutada, kuid neil puudub sisse-ehitatud võimalus luua jäika versioneeritud andmekuju. Suurte ja keeruliste süsteemide puhul on meie hinnangul vastupidava andmevahetuse loomiseks see aga väga oluline. Näiteks, versioneerimist kasutamata pole kestrakenduse veateadete andmekuju muutumisel MFE-d uuest andmekujust teadlikud. Kestrakenduse ja MFE-de vaheliseks **otsesuhtluseks** nimetatud tehnoloogiad **kasutada me ei soovita**, kuid järgmises peatükis pakume välja lahenduse, kuidas kirjeldatud probleeme lahendada.

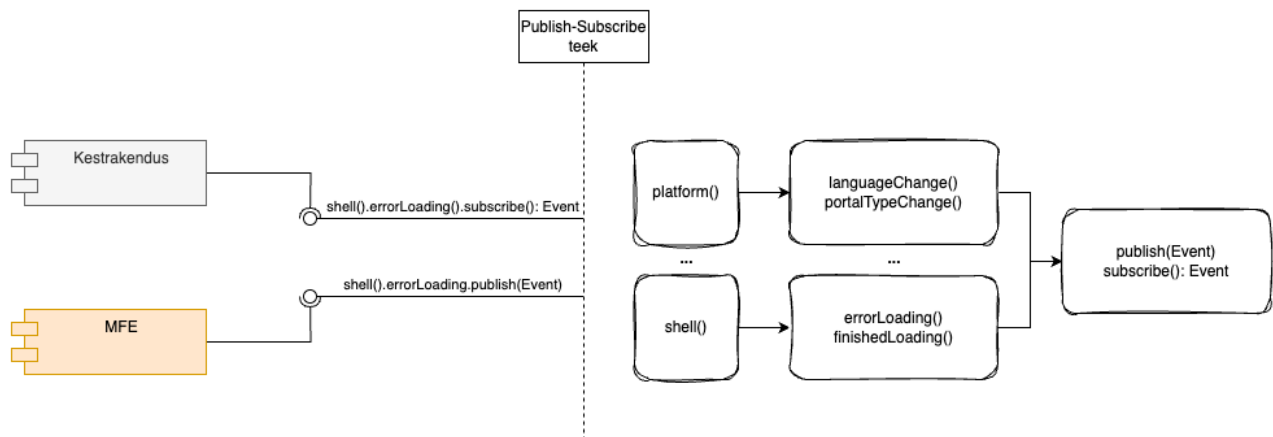
Riigiportaali **eesti.ee kogemusel** soovime luua TARVIKu **andmevahetuseks** eraldi **teegi**, mis põhineb **publish-subscribe** põhimõttel. Näiteks, MFE-siseste andmete laadimise ebaõnnestumisel publitseerib MFE teegi kaudu veateate, mida kestrakendus kuulab ning oskab sellele reageerida.

Teegi loomise ja kasutamise peamised **eelised** on:

- **jäik andmekuju** (publitseerijad ja tarbijad teavad täpselt milliseid andmeid oodata ning on võimalik luua tagasiulatuva ühilduvus),
- **võimalus** andmete hoiustamise **tehnoloogiat muuta** (*window object, global store, localStorage, sessionStorage* jne),

- võimalus luua **selge andmete hierarhia** (platvormi sõnumid, MFE-spetsiifilised sõnumid jne),
- võimalud **versioneerida** ja paketeerida kasutades npm-i,
- võimalus teeki integreerida andmevahetuseks ka PostMessage API vms tehnoloogiaga.

Peamine **puudus** sellisele lahendusele on tõsiasi, et see tuleb välja arendada ning kõik MFE-d on sunnitud aeg-ajalt teegi versiooni uuendama, mis võib autonoomsust vähendada, kuid tagasiulatava ühilduvusega on võimalik seda riski maandada.



Joonis 33. Andmevahetus kestrakenduse ja MFE vahel publish-subscribe põhimõttel

Kuigi keel jms süsteemiparameetrid on tihti URL-ilt kättesaadav, soovitame neid ka *publish-subscribe* põhimõttel propageerida, et iga MFE ei peaks üldiste parameetrite lugemist ise eraldi implementeerima ja kestrakendusele jääks võimalus nende loogikat muuta.

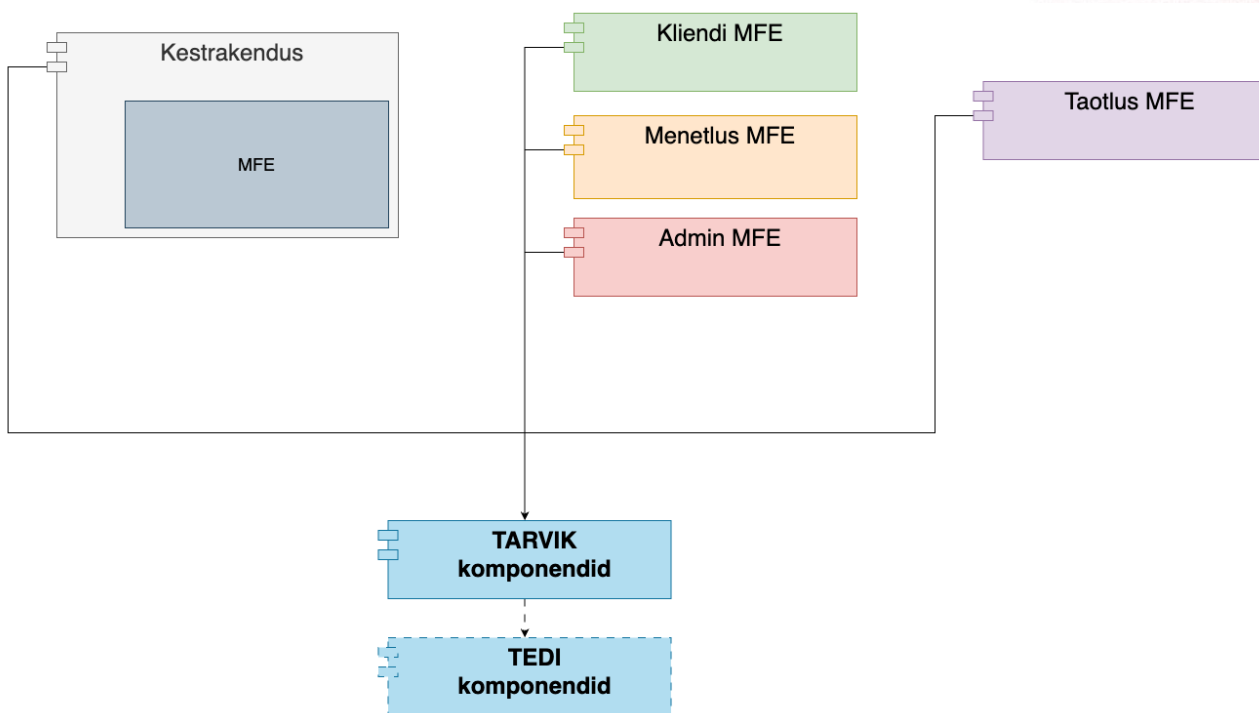
MFE-de omavahelist suhtlust (ka Publish-Subscribe teegi kaudu) soovitame vältida, sest see on märk kehvasti disainitud MFE-dest ning sel juhul tuleks analüüsida, kas MFE komponent on liiga kitsa ulatusega.

3.5.4 Disain

Ühtne disain kestrakenduse ja MFE-de vahel on kriitilise tähtsusega, et tagada hea kasutajakogemus. Võib leida esitluskihi komponente, mida on vaja riskasutada kestrakenduses või mitmes MFE-s. Ei ole otstarbekas implementeerida üldlevinud komponente nagu tabel, nupud, modaalid jms kestrakenduses ning igas MFE-s eraldi.

Soovitame luua esitluskihi komponentide teek *npm* pakina ning kaaluda ka [TEDI](#) kasutuselevõttu. TEDI on TEHIKu poolt arendatav, [Veera](#) disainisüsteemi järgiv, Angulari ja Reacti kasutajaliidese komponentide kogumik. Nii on võimalik tulevikus oluliselt arendus- ja hoolduskuludelt säästa ning portaali kujundus ühtsena hoida.

Lahendus eeldab olemasolevate komponentide asendamist teegi komponentidega, mis võib olla üsna tömahukas. Esialgu võib kaaluda ka korduvate komponentide kopeerimist erinevatesse MFE-desse, mida hiljem järk-järguliselt asendada korduvkasutavate komponentidega, et ülemineku keerukust vähendada.



Joonis 34. TARVIK ja TEDI komponentide taaskasutamine

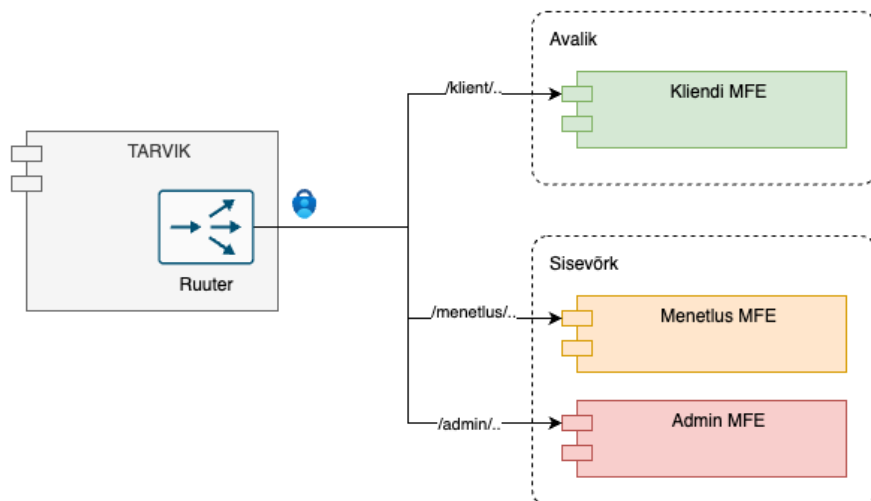
3.5.5 Autentimine, autoriseerimine ja infoturve

Autentimiseks soovitame kasutada [GovSSO](#)-d ning autoriseerimiseks GovSSO silda [Pääsukese](#) pääsuhaldussüsteemi. GovSSO võimaldab kasutajat autentida kasutades [TARA](#) ning võimaldab sessiooni jagada kõigi GovSSO-ga liitunud süsteemide vahel (näiteks eesti.ee portaali ja TARVIKu vahel navigeerides). Samuti on võimalik GovSSO väljastatud ID Tokenist lugeda kasutaja esindusõigused ning neid autoriseerimiseks rakendada. Kasutajarolle on võimalik laadida Pääsukesest ning vajadusel teisendada TARVIKu privileegideks.

Kuna pakutud lahenduse MFE-de *route*'ide juured on täielikult kestrakenduse kontrolli all, on võimalik kestrakenduse kaudu juhtida ka [route garde](#) ja nendega rakendada autentimisest ja autoriseerimisest tulenevaid piiranguid. Kui mõni alam-*route* Kliendi MFE sees vajab erikohtlemist, siis on võimalik seda teha MFE sees.

Autentimisest ja autoriseerimisest esitluskihi tasemel ei piisa, seda tuleb teha ka taustteenustes ning kestrakendus ja MFE-d peavad olema valmis vastavatele veakoodidele reageerima, et kasutajat ebaõnnestunud päringutest teavitada ning sisu kuvamist piirata.

Autorite visiooni järgi võivad Menetlus MFE ja isegi Admin MFE olla orkestreeritud sama kestrakenduse poolt, mis Kliendi MFE. Kui mõne MFE (näiteks Admin MFE) andmeteta esitluskiht võib liigselt reeta sisemiste tööriistade ülesehituse kohta, siis on võimalik vastavad moodulid muuta ainult sisevõrgust kättesaadavaks.



Joonis 35. Tundlike MFE-de piiramine sisevõrgust kättesaadavaks

3.5.6 Testimine

Süsteemi suurtest muudatustest põhjustatud probleemide vältimiseks soovitame olemasolevad süsteemi komponendid katta automaattestidega, integratsioonide katvusele ei ole meie hinnangul otstarbekas keskenduda, sest need võivad MFE-stamise käigus muutuda.

MFE-de testimiseks soovitame järgida mikroteenuste testimise [põhimõtteid](#). Soovitame ühiktestida ühe MFE piires ning integratsioone testida kestrakenduse ja iga MFE vahel. MFE-de omavahelisi integratsioone soovitame vältida, kuid nende esinedes tuleks neidki testida, sest **muudatused MFE-de integratsioonide vahel ei avaldu kompileerimise hetkel** ning nende muutudes võivad vead ebapiisava testidega katmise korral jõuda **märkamatu** toodangusse.

End-to-end testimise vajadus **MFE-de vahel** viitab [kehvasti disainitud hajussüsteemile](#) ning autonoomsuse puudumisele. Siiski ühe MFE piires on see meie hinnangul suurepärase vahend MFE *smoke* ja *sanity* testimiseks.

Regressioonitestimiseks soovitame kasutada automatiseeritud tööriistu: ühiktestimiseks [Jasmine](#) ning *end-to-end* testimiseks [Playwright](#). Mõlemad on küpsed, laialt levinud ning regulaarselt hooldatud. Soovitame jälgida testide katvuse head tava: 80% funktsionaalsetest komponentidest.

3.5.7 Kasutuselevõtt

Järgnevalt kirjeldame olulised sammud ja tegevused Webpack Module Federation mikrositluskihide tehnoloogia juurutamiseks ning kirjeldame lahendusi levinud probleemidele. Pakutav kasutuselevõtu plaan lähtub CI/CD põhimõtetest, et võimaldada tihedaid tarneid ning järkjärgulist üleminekut. Siiski tuleks arvestada, et iga loodav MFE peaks läbima ka turvatesti, mis võib seada tellijale piiranguid toodangusse samm-sammult liikumiseks. MFE-de kasutuselevõtt ei sõltu otseselt mikroteenustest, seega ei eelda peatükis „Mikroteenuste arhitektuur“ ettepanekute elluviimist.

I faas: analüüs

Analüüsi faasis tuleb välja selgitada äridomeenide vahelised piirid ning nende vahelised seosed. Pöörame tähelepanu, et esitluskähi äridomeenid ei pruugi kattuda peatükis „Domeenipõhine disain“ kirjeldatud domeenidega, kuigi samad põhimõtted kehtivad siingi.

Samuti peaks analüüsima riskasutust vajavaid komponente, näiteks kliendi taotluse koondandmete vaade menetlejale või lubade ja teadete nimekirja vaade riigiportaalis eesti.ee kuvamiseks. Esimeses faasis on oluline kaardistada lisaks uuele loodavale funktsionaalsusele ka juba olemasolev.

Äridomeenide vaheliste piiride kaudu on võimalik kirjeldada täpsed MFE-d ning nende puutepunktid. Soovitame MFE-sid hoida suurte loogiliste komponentidena, kuna liiga palju väikseid MFE-sid võib tõsta süsteemi keerukuse ebamõistlikult kõrgele. Autorite hinnangul on hästi koostatud MFE sisendparameetrite arv 0, sest siis on võimalik MFE-d vabalt muuta ilma, et tarbija peaks muudatusi tegema. Sellega on tagatud täielik autonoomsus ning maksimaalselt lõdvdad seosed.

Näiteks, geneeriline komponent Nupp ei ole hea kandidaat MFE-ks, sest eeldab sisendparameetreid (tekst, ikoon, värv jne), mille kuju võib tihti muutuda ja sobib seetõttu paremini *npm* pakina esitamiseks.

Küll aga klienditoode sobib hästi eraldi MFE-ks, sest koondab enda alla ühe äridomeeni (kliendiga seotud ärifunktsionaalsuse) ning on võimeline teistest äridomeenidest sõltumatult toimima.

II faas: olemasoleva funktsionaalsuse MFE-stamine

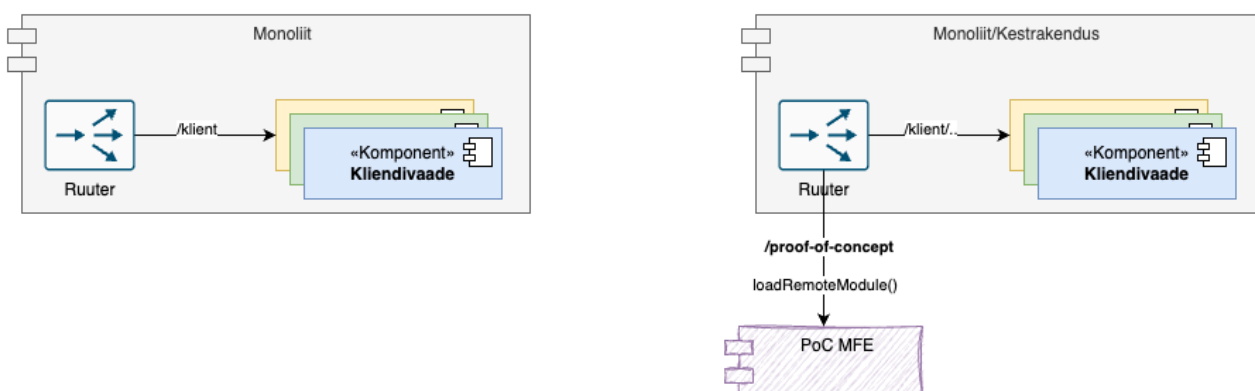
Olemasoleva funktsionaalsuse MFE-stamine ei eelda peatükis “Menetlustoote kasutajaliidese arendus” kirjeldatud ärianalüüsi valmimist.

I samm. *Module federationi* kasutuselevõtuks soovitame esmaselt värskendada praeguse TARVIKu esitluskihi rakenduse sõltuvused uusimatele versioonidele. Seejärel paigaldada Webpack Module Federation ning [seadistada TARVIKu rakendus hostiks](#). Lihtsuse huvides soovitame esialgu seadistada kõik sõltuvused [singletonideks](#) ning mitte nõuda rangeid versioone (brauseri konsoolis kuvatakse selle kohta hoiatusi). Nüüd on TARVIK võimeline teekondadel (Angular *route*) põhinevaid MFE-sid laadima ja kuvama ning seda võib nimetada **kestrakenduseks**. Hiljem eraldatakse ärifunktsionaalsus praegusest TARVIK monoliit rakendusest.

Pärast 1. sammu on võimalik TARVIK toodangusse tarnida.

II samm. Järgmiseks soovitame luua *proof-of-concept (PoC)* MFE projekti ning seadistada see *remoteks*. Sarnaselt kestrakendusele kasutada lõtvade versioonireeglitega *singleton*. Lisada Angular *route(d)/module/component* ning [eksponeerida](#) see `remoteEntry.js` failina, kusjuures ühest *remotest* on võimalik eksponeerida ka mitu olemit. Pärast eksponeerimist on võimalik kestrakendusel seda tarbida.

Lisada TARVIKusse juurde uus teekond, millega MFE [sisse laetakse](#). Korrekse seadistuse puhul sellest piisab, et näidata MFE-de sisse laadimise ja TARVIKus kuvamise võimalikkust. See projekt saab olema etaloniks järgmistele loodavatele MFE-dele.



Joonis 36. *Proof-of-concept MFE lisamine*

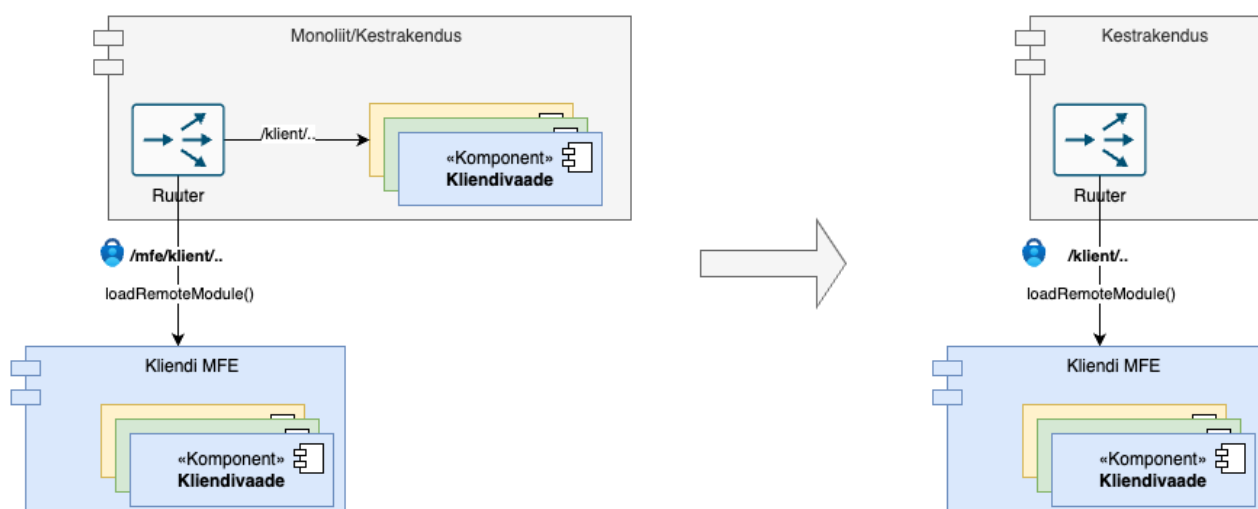
2. sammus loodud teekonnad ning *PoC* rakendust toodangusse mitte tarnida.

III samm. Nüüd on võimalik hakata olemasolevat funktsionaalsust MFE-stama. Näiteks, kui analüüsis jõuti järeldusele, et Kliendi MFE on sobilik autonoomne komponent, siis on võimalik:

1. olemasolev funktsionaalsus tõsta eraldi Git repositooriumisse,
2. eksponeerida kestrakendusele tarbimiseks,
3. luua CI/CD töövoog ehitamiseks ning paigaldamiseks.

Näiteks, kui praeguses lahenduses jaguneb kliendivaade mitmeks Angular *routeks*, siis on võimalik need kõik eksponeerida ning lisada kestrakendusse.

Soovitame MFE-lahendusega toodangusse minna peidetult ehk jätta esialgu alles praegune avalik teekond klientidele ning peidetud MFE-statud teekond testimiseks. Täiendavalt on võimalik lisada ka piirangud, et teekond on kättesaadav ainult sisevõrgust.



Joonis 37. Olemasoleva funktsionaalsuse MFE-stamine

Alternatiivselt võib ka ajutiselt tarnida 2 eraldi rakendust, kus MFE-statud rakendus asub TARVIKu alamdomeenil, et lihtsustada CORS jms turvareeglite rakendamist, võimaldamaks toodangusse tarnida nii MFE-statud kui ka monoliitrakendus korraga. Kui MFE-statud rakenduse töökindluses on jõutud veendumusele, on võimalik toodangu monoliitrakendus täielikult asendada MFE-statud rakendusega lõppkasutajale märkamatuks.

Ülemineku perioodil soovitame olemasolevat funktsionaalsust muuta minimaalselt, et monoliitrakenduses ning MFE-statud rakenduses ei peaks muudatusi dubleerima.

III faas: uue funktsionaalsuse lisamine MFE-dena

Uue funktsionaalsuse lisamine MFE-dena eeldab ärianalüüsi. Menetlustoote jaoks on seda kirjeldatud peatükis “Menetlustoote kasutajaliidese arendus”.

Enne uue funktsionaalsuse lisamist soovitame põhjalikult veenduda vähemalt ühe toimiva MFE kasutamises mitmes keskkonnas (näiteks Kliendi MFE). Seejärel on võimalik lihtsama vaevaga hakata looma uusi MFE-sid võttes esimest eeskujuks ning järgides juba mainitud tegevusi. Kõiki järgmisi MFE-sid on nüüd võimalik arendada ka samaaegselt.

Iga uue MFE rakenduse lisamine eeldab olemasolevasse CI/CD töövoogu ehitus- ja paigaldusskriptide lisamist.

Väljakutsed

Mikroesitluskihtide areng on olnud kiire. Praegused veebilahendused tihtipeale ei arvesta MFE paradigma põhimõtetega. Siin peatükis toome välja peamised probleemid, mis riigiportaali eesti.ee kogemusel võivad vajada lahendamist ka TARVIKu MFE-stamisel.

Probleem I: stiilide (CSS) lekkimine.

Enamik rakendusi kasutavad nii-nimetatud globaalseid stiile, mis rakenduvad kas mõnele HTML elemendile (nt `<body>`, `<h1>`, `<p>` jne) või kasutavad juurskoobis geneerilist klassinime (nt „modal“, „button“ jne).

Soovitame alati kõik CSS klassinimed skoopida komponendi piires või kui see ei ole võimalik, kasutada eesliidet, et vältida nimekonflikte. Näiteks CSS klassi „modal“ asemel kasutada „tarvik-modal“.

Vaikimisi [Angular skoobib komponentide stiilid](#) kasutades `ViewEncapsulation.Emulated`, millega rakendatakse ka vanem-komponentide stiilid (näiteks kestrakenduse stiilid MFE-le). Kui soovitakse täielikku stiilide isoleeritust, tuleks kasutada `ViewEncapsulation.ShadowDom` ja/või Web Component tehnoloogiat. Soovitame vältida `ViewEncapsulation.None` kasutamist, sest siis rakendatakse alam-komponentide stiilid ka vanem-komponentidele, näiteks kui MFE määrab tekstivärvi, muutub tekstivärv ka kestrakenduses.

Probleem II: globaalsed komponendid.

Leidub kolmandate osapoolte (näiteks Ng Bootstrap) komponentide implementatsioone, mis toimimiseks lisavad DOM-i lõppu HTML-i, näiteks modaali, *overlay*’d jms. Selliste valmislahenduste kasutamine MFE-s ilma `ViewEncapsulation.None` kasutamata võib osutuda keeruliseks.

Sel juhul soovitame kolmandate osapoolte implementatsioone mitte kasutada, vaid arendada enda (kasutada TEDI) komponente, mis arvestavad ka MFE-de omapäradega. Tuleks arvestada, et olemasolevate komponentide asendamine võib olla üsna ajamahukas.

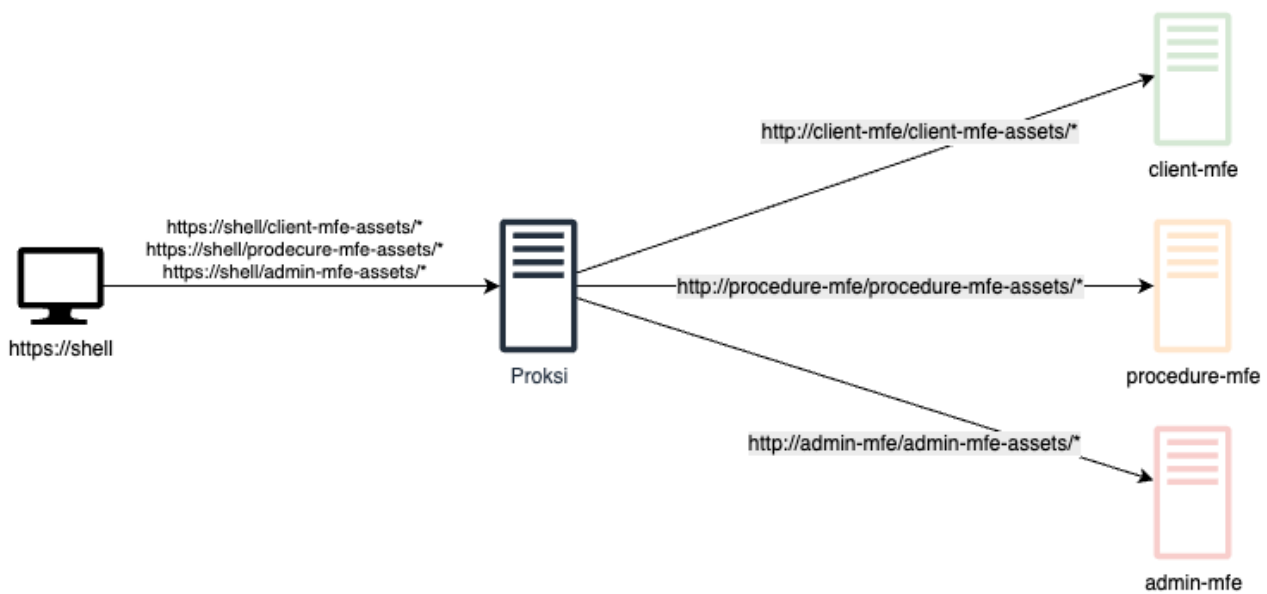
Probleem III: innukas cache’imine.

Kasutajakogemuse tõstmiseks *cache*’ivad tänapäevased lahendused võimalikult palju staatilisi ressursse, et vältida nende mitmekordset laadimist. Kuna MFE-sid laetakse staatiliste ressurssidena - JavaScript (js) või Module JavaScript (mjs) failidena - ning nende aadress ei sisalda unikaalset *hash*i ega muud strateegiat *cache* uuendamiseks, siis soovitame rakendusserveris seadistada `remoteEntry.js` failidele [no-cache](#). Kui kasutusel on mõne kolmanda osapoolte lahendus, näiteks Cloudflare, siis tuleb ka sealseid seadistusi kontrollida. Kui `remoteEntry.js` *cache*’imine ei ole välja lülitatud, on **esitluskihi rakenduse töö pärast tarneid paljude kasutajate jaoks suure tõenäosusega häiritud**.

Probleem IV: staatiliste ressursside laadimine.

MFE staatilised ressursid ehk pildid, ikoonid jms ei ole kestrakenduses kättesaadavad, sest asuvad MFE aadressil.

Soovitame iga MFE staatilised ressursid paigutada unikaalse nimega kataloogi ning lisada kestrakenduse veebiserverisse staatiliste ressursside proksimine.



Joonis 38. MFE-de staatiliste ressursside proksimine

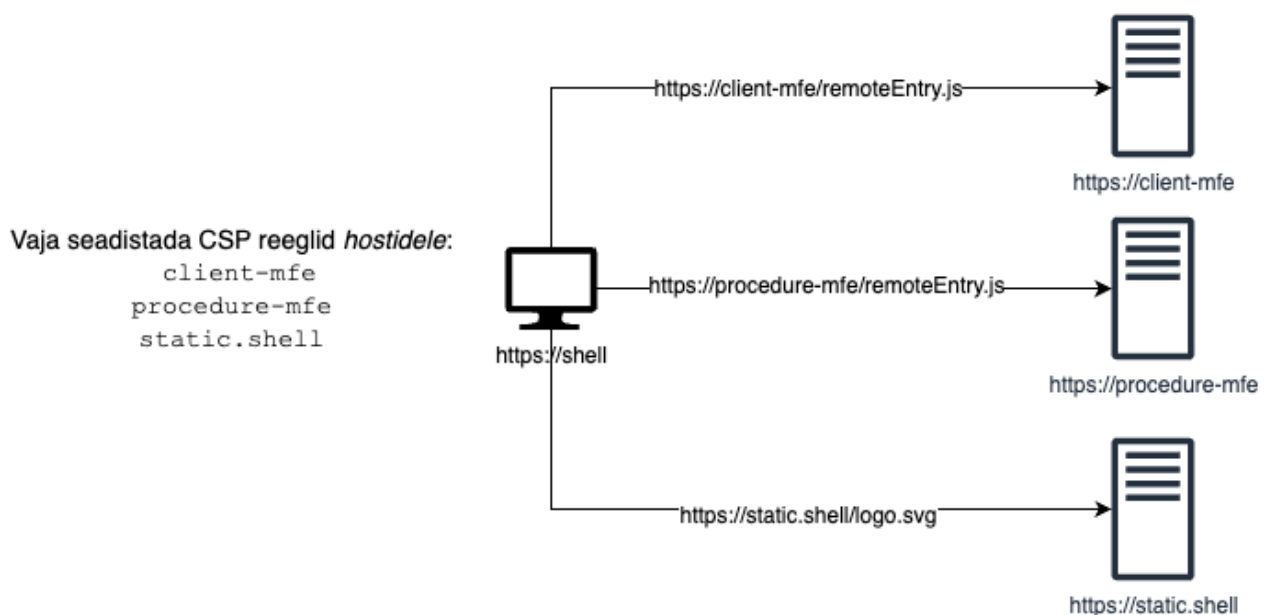
Lisaks soovitate seadistada Webpack [publicPath](#) (Angular Architects pistikprogramm teeb seda vaikimisi `''`). Siis lisab Webpack staatilistele ressurssidele MFE-aadressid automaatselt. Paraku **koodis refereeritud ressursse Webpack dünaamilisest laadima ei ole võimeline**, näiteks MFE piltide laadimine ebaõnnestub, kui kasutada relatiivset teekonda pildini:

```
.
```

Mainitud probleemi lahendust kirjeldame peatükis „MFE laadimine“.

Probleem V: turvapäistest tulenevad piirangud.

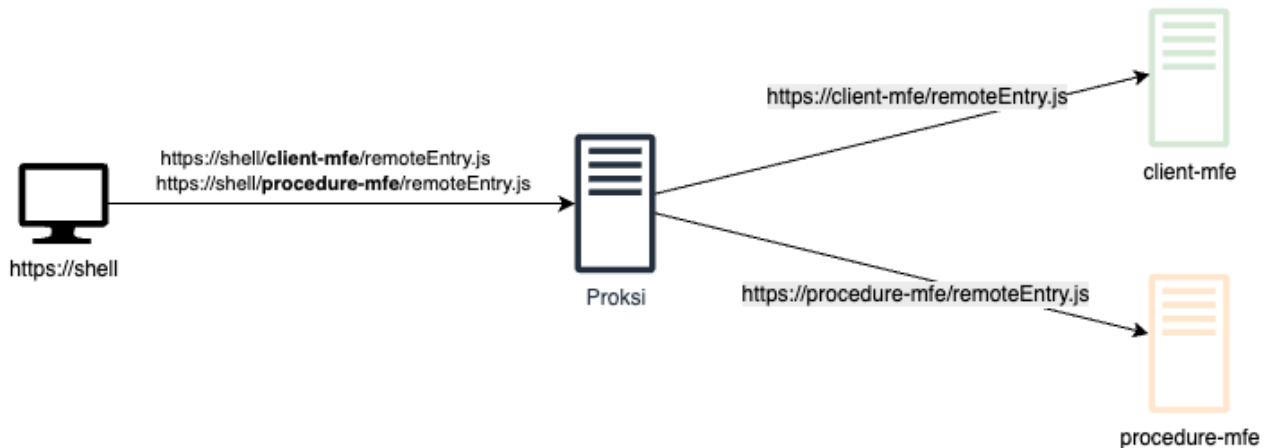
Veebirakenduste ressursside kaitsmiseks kasutatakse levinud klientrakenduste turvapäiseid, näiteks CSP ja CORS reegleid. CSP-d kasutatakse peamiselt failide (skriptid, pildid jne) laadimise piiramiseks, CORS-i aga taustteenustega suhtlemise piiramiseks. Kasutades kestrakenduse ja MFE-de jaoks erinevaid (alam)domeene, muutub kestrakenduse turvapäiste haldamine keerulisemaks.



Joonis 39. MFE skriptide ja muude ressursside laadimiseks CSP reeglite seadistamine

Näiteks, kui Kestrakendus asub domeenil `shell` ning Kliendi MFE asub domeenil `client-mfe`, siis tuleks (iga) MFE domeeni lisada turvapäiste erandite hulka, et kestrakendusel oleks lubatud MFE-d laadida. Ka alamdomeenide kasutamine ei aita, sest kõikide alamdomeenidele erandi lisamine on tülikas ning *wildcard* (*) kasutamine domeenireeglite lisamisel pole piisavalt turvaline, sest laiendab oluliselt ründepinda.

Selle asemel soovitame MFE-de puhul kasutada põhidomeeni alamkatalooge koos proksiga. Selliselt piisab põhidomeenile (`shell`) turvapäiste seadistamisest, turvalisuse osas järeleandmisi tegemata.



Joonis 40. MFE staatiliste ressursside laadimine läbi proksi

Probleem VI: Sõltuvuste konfliktid

Webpack Module Federation võimaldab defineerida igale MFE-le oma sõltuvused ja/või nende versioonid. Nii on võimalik tagada täielik autonoomsus, kuid ühe asutuse piires soovitame siiski tuumsõltuvuste (nt Angular) versioone hoida sünkroonis. *Singleton*-konfiguratsiooni kasutamisega on võimalik maandada vigade tekkimise tõenäosust ning see aitab vältida vigade avaldumist alles pärast paigaldamist. Siiski, ka *singletonidega* võib eriversioonide kasutamisel [vigu esineda](#). Selleks, et neid varakult avastada, soovitame MFE-de integratsioonid kestrakendusega katta automaattestidega.

Singletonide kasutamisel tuleks arvestada sellega, et nendele sõltuvustele ei ole võimalik teostada *tree shaking* optimeerimist. Näiteks, kui sõltuvusena on kasutusel suuremahuline ikoonide pakk, millest päriselt kasutatakse väikest arvu ikooni, siis *singleton* sõltuvusena laetakse ikkagi kogu paki sisu, mitte ainult tarvilik.

Väliste süsteemide integreerimisel või muul juhul vältimatul versioonide/raamistike lahknevusel tuleks iga MFE skoopida, kasutades [Web Component ja Shadow DOM tehnikaid](#). Need võimaldavad MFE täielikult isoleerida kestrakendusest ja teistest MFE-dest. Sellise lähenemisega tuleks arvestada, et iga MFE jaoks laetakse **kõik** sõltuvused alati uuesti ehk rakenduse laetavate ressursside maht kasvab oluliselt määral.

3.5.8 Integratsioon väliste süsteemidega

Mikroesitluskihtide üks paljudest kasuteguritest on võimalus esitluskihi komponente taaskasutada nii lokaalses kui ka välistes süsteemides. Riigiportaali eesti.ee on MFE-valmidusega, mis loob potentsiaalse võimaluse TARVIKu komponente seal kuvada. Selles peatükis käsitleme integratsiooni väliste süsteemidega eesti.ee portaali näitel ning sellega kaasnevaid väljakutseid.

MFE laadimine

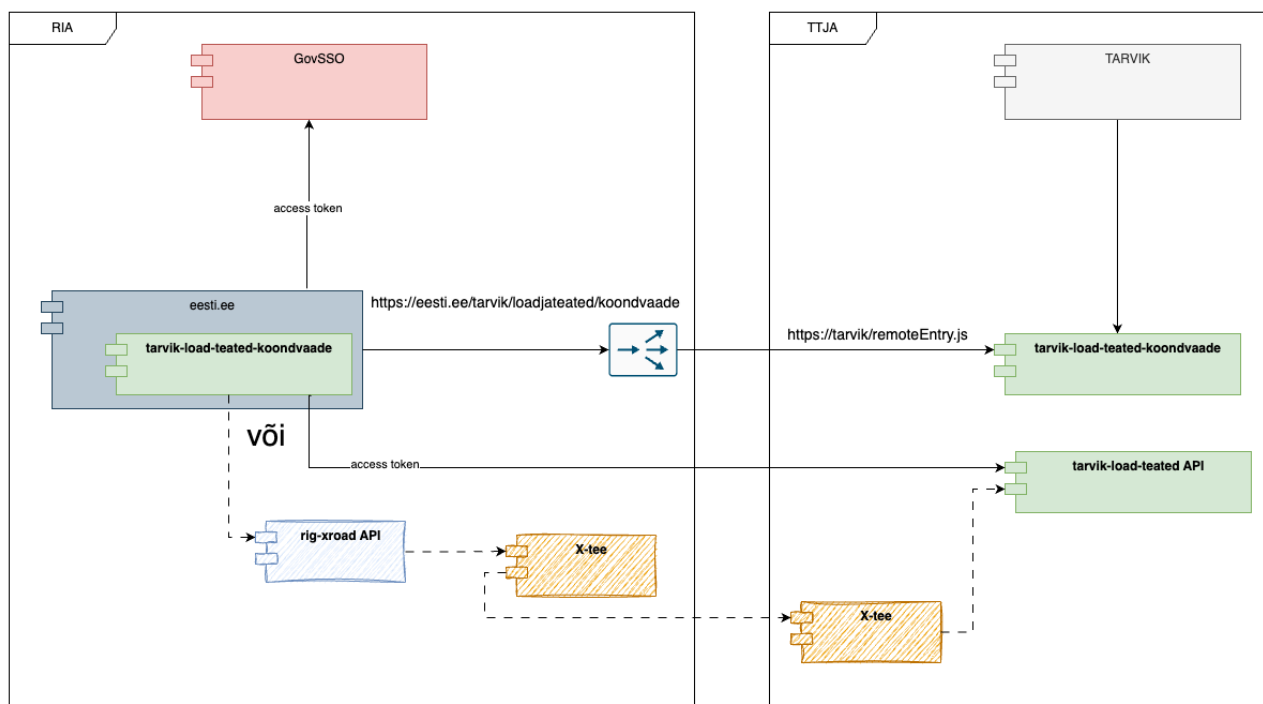
Kuna TARVIKu MFE asub eesti.ee portaalist erineval domeenil, on vaja eesti.ee portaalis lubada `remoteEntry.js` ning MFE staatiliste ressursside laadimine TARVIKu domeenilt. Alternatiivselt on võimalik lisada proksi eesti.ee domeenilt TARVIKusse, kuid ideaalis soovitame neid välises süsteemides vältida, et vähendada süsteemide-vahelisi jäike seoseid. Peatükis “Probleem IV: staatiliste ressursside laadimine.” selgitati täpsemalt miks staatilisi ressursse ei ole võimalik dünaamiliselt MFE-st laadida ilma proksita. Proksi vältimiseks on võimalik kasutada koodis [Webpack publicPath](#) staatilise ressursi relatiivse teekonna ees:

```
.
```

Andmevahetus

Kasutades *module federationit*, laetakse MFE kood alati orkestreerivasse (kestrakenduse) komponenti ja seetõttu **MFE kood käivitatakse alati orkestreerivas (host) rakenduses**. Avaliku sektori infosüsteemid kasutavad andmevahetuseks enamasti X-tee, kuid MFE-de kasutamiseks tähendaks see eesti.ee portaalis TARVIKu X-tee teenustega liidestumist. Taoline lahendus loob meie kogemusel liigseid jäiku seoseid TARVIKu ja eesti.ee portaali vahel. Näiteks, taotluse andmekuju muutmiseks API muudatuste korral oleks vaja ka eesti.ee portaalis teha arendusi rig-xroad API komponendis, et uus X-tee teenus kasutusele võtta.

GovSSO-l on valminud OAuth [Access Tokenil](#) põhinev lahendus, millel on potentsiaal vältida X-tee liidestusi ning teostada päringud otse TARVIKu REST APIde kaudu ning TEHIKu lisaarendustega võimaldaks seda kasutada mustri [Backend for Frontend](#) (BFF). Toimiv [Token Mediation](#) lahendus on eesti.ee ja Riikliku Postkasti vahelise integratsiooni jaoks toodangus juba ka kasutusel.



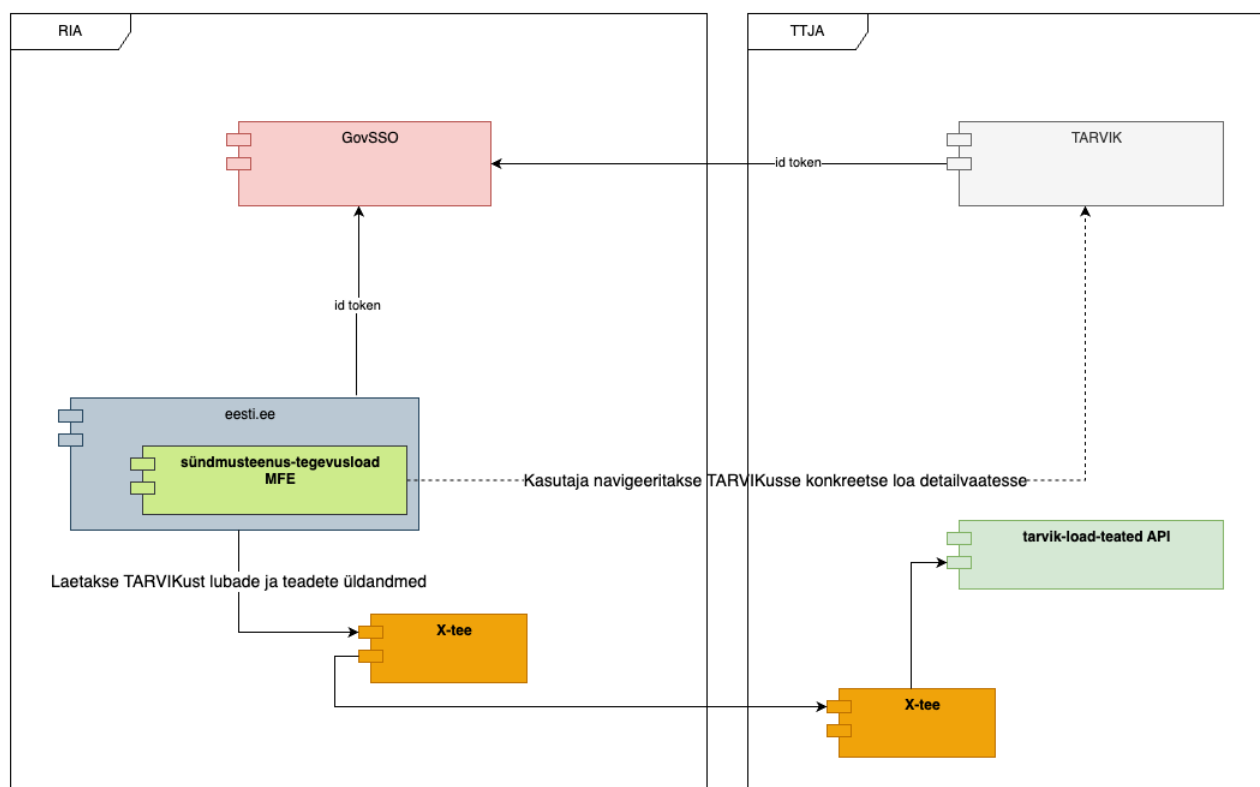
Joonis 41. Turvalise andmevahetuse alternatiivid eesti.ee portaali ja TARVIKu vahel: access token või X-Tee

Alternatiivina võib kaaluda ka [Server Side Rendering](#) (SSR) lahendust, millega puudub vajadus andmevahetuse integratsiooni loomiseks väliste süsteemide vahel, kuid võib potentsiaalselt tekitada suurt koormust TARVIKu füüsilisele serveriressursile.

Otstarbekus

Soovitame analüüsida TARVIKusse koondavate lubade ja teadete (aga ka muude potentsiaalsete TARVIKu komponentide) MFE-na eesti.ee portaalis kuvamise otstarbekust. Riigiportaalis on juba valminud majandustegevuslubade ja -teadete mitmetest asutustest agregeerimise sündmusteenus, millega liidestumiseks piisab ühe REST otspunkti lisamisest TARVIKusse. Veelgi enam, TARVIKu komponent ei annaks lõppkasutajale ammendavat nimekirja tema ettevõttele väljastatud lubadest ja teadetest, sest ei sisalda andmeid teistelt asutustel. Meie **hinnangul arendamise keerukus ei kaalu saadud kasu üle**.

Selle asemel soovitame TARVIKus kasutada GovSSO-d, mis võimaldab kasutajal hõlpsasti riigiportaali ja TARVIKu vahel navigeerida ning autentida. Niiviisi on riigiportaali tegevuslubade sündmusteenusest jõudmine TARVIKusse (lubasid vaatama aga ka muutma või esitama) lihtne.



Joonis 42. TARVIKu liidestamine olemasoleva eesti.ee Tegevusload Sündmusteenusega üle X-tee

3.5.9 Riskid ja maandamismeetmed

Selles peatükis kirjeldame mikroesitluskihtide juurutamisega seotud riske ning võimalusi nende maandamiseks.

Risk	Tõenäosus	Mõju	Maandamismeetmed
Esitluskihi keerukuse kasv.	Kõrge	Kõrge	Vältida süsteemi liigset tükeldamist; vältida jäike seoseid MFE-de vahel; vältida erinevate raamistike kasutamist vähemalt ühe asutuse piires; analüüsida põhjalikult MFE komponentide taaskasutuse otstarbekust; serveerida MFE komponente samalt domeenilt;

			kasutada jäika, kuid tagasiulatuva ühilduvuse võimalusega andmevahetuskihti kestrakenduse ja MFE-de vahel.
Ebaühtlane disain ja kasutajakogemus.	Keskmine	Keskmine	Viia olemasolev rakendus üle Veera disainisüsteemile; kasutada kesket komponentide teeki (TEDI); rakendada kesket veahaldust läbi kestrakenduse ja andmevahetuskihi. Täpsemalt loe peatükist „3.5.4 Disain“.
Kasvav halduskoormus väliste rakendustega integreerumisel (eesti.ee portaal).	Keskmine	Kõrge	Maksimaalse isoleerituse tagamine: Web Componenti kasutamine, GovSSO <i>access tokeni</i> juurutamine X-tee asemel. Kaaluda MFE kuvamise asemel süvalingiga kasutaja suunamist portaalide vahel. Täpsemalt loe peatükist „3.5.8 Integratsioon“.
Eesti.ee portaaliga liidestumise takerdumine GovSSO liidestuse puudumisel või GovSSO pooleliolevate <i>access tokeni</i> arenduste tõttu.	Keskmine	Madal	Prioritiseerida GovSSO juurutamist olemasolevatesse TARVIKu süsteemidesse; <i>access token</i> lahenduse puudumisel implementeerida integratsioon üle X-tee. Loe täpsemalt peatükist „Andmevahetus“.
Infoturbe seisukohalt tundlike (andmeteta) esitluskirhi komponentide avalik kättesaadavus.	Madal	Madal	Muuta tundlikud MFE-d (Menetlus ja Admin MFE) kättesaadavaks vaid sisevõrgust. Loe täpsemalt peatükist „3.5.5 Autentimine, autoriseerimine ja infoturve“.
Vead MFE-de sõltuvuste erinevate versioonide/raamistike integreerimisel ja tarnimisel (CI/CD).	Kõrge	Kõrge	<i>Singleton</i> sõltuvuste kasutamine; kestrakenduse ajakohasena hoidmine; MFE-de skoopimine Web Componentiga. Loe täpsemalt peatükist: “Probleem VI: Sõltuvuste konfliktid”.
MFE-de liigsest põimumisest tekkinud sõltuvuste tõttu autonoomsuse vähenemine (hajusa monoliidi tekkimine).	Madal	Keskmine	Domain-Driven Design põhimõtete järgimine ning põhjalik eelanalüüs äridomeenide jaotumiseks. Loe täpsemalt peatükist „Domeenipõhine disain“ ning „3.5.1 TARVIKu mikroesitluskihide üldarhitektuur“.
Suurest arvust MFE-dest põhjustatud versioonihalduse keerukuse kasv (palju repositooriume).	Madal	Madal	Domain-Driven Design põhimõtete järgimine ning põhjalik eelanalüüs äridomeenide jaotumiseks. Loe täpsemalt peatükist „Domeenipõhine disain“ ning „3.5.1 TARVIKu mikroesitluskihide üldarhitektuur“.

Olemasoleva funktsionaalsuse regressiooni testimise suur keerukus ja koormus Tellijale.	Kõrge	Kõrge	Olemasoleva funktsionaalsuse katmine automaattestidega. Loe täpsemalt peatükist „3.5.6 Testimine“.
Rakenduse laetavate ressursside mahu suurenemine.	Madal	Kõrge	Seadistada jagatavad ressursid <i>singletoniks</i> ning lõdvendada versioonide kattuvuse nõudeid Webpack konfiguratsioonis . Pöörata tähelepanu ka tree shaking piirangutele module federationiga .
Stiilide lekkimine kestrakenduse ja MFE-de vahel.	Keskmine	Madal	Rakendada strateegiaid peatükist “Probleem I: stiilide (CSS) lekkimine.”.

4 Tehisaru kasutamine

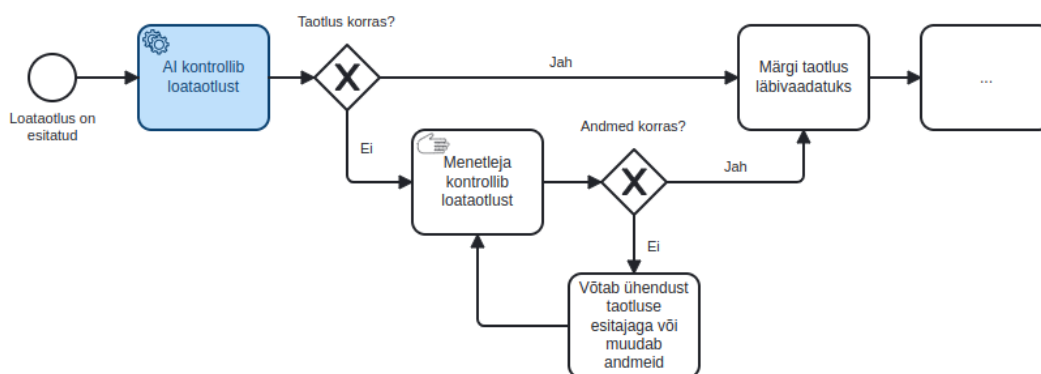
Kuna Hankija eeldab Pakkujalt AI komponentide kasutuselevõttu loodavates lahendustes (“Hanke tehniline kirjeldus” ptk 6 lõpp), siis antud pakumuse lisandväärtusena toob Pakkuja välja mõningad võimalikud kasutusjuhud, kus tehisaru kasutamine oleks võimalik. Pakutavate lahenduste loetelu ei ole lõplik.

Tehnoloogiat tuleks kasutada inimeste töö ja elu lihtsustamiseks nii palju kui võimalik. Lisaks loogilise ja mugava süsteemi disainimisele ja arendamisele on võimalik kasutada ka tehisaru (AI) protsesside kiiremaks ja lihtsamaks muutmiseks. Alljärgnevalt pakume välja mõned variandid, kuidas AI-d TARVIKu kontekstis ära saaks kasutada ning kuidas seda tehniliselt realiseerida.

4.1 Automaatne menetlemine

Menetlemise käigus otsustab menetleja hetkel sisendandmete põhjal, kas taotlusele saab menetluse käigus luua positiivse otsuse või mitte. Need sisendandmed võivad tulla nii kodaniku poolt täidetud vormilt kui ka võib ametnik vajalikku informatsiooni mujalt juurde hankida. Võttes aluseks lõpetatud menetluste sisendandmed ja otsuse tulemused, on **võimalik treenida AI mudeleid, mis suudavad menetleja eest võtta vastu otsuse juhul, kui süsteem on selles 100% kindel.**

Selle jaoks saab treenida otsustuspuit, mis käivitatakse menetlusprotsessi teatud sammus. Kui mudel on veendunud, et taotluse võib saata edasi järgmisesse sammu, saab ta selle automaatselt ära teha. Kui mudel aga ei ole selles täitsa veendunud, siis jätkatakse tavalise menetlusvooga, kus ametnik peab ise protsessi sekkuma. Alternatiivina võib mudel lihtsalt anda menetlejale omapoolse arvamuse, kas taotlus tundub positiivse otsuse loomiseks korrektne või mitte. Allpool on toodud näide [MTRi loataotlusprotsessi](#) algusest, kus saaks kasutada ära masinõppe võimalusi.

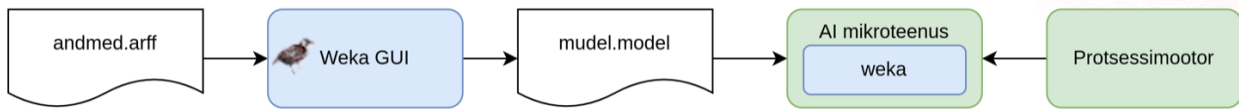


Joonis 43. Tehisaru kasutamine loataotlusprotsessi koosseisus

Tehniline lahendus

Mudeli treenimiseks ja jooksumiseks on ühe võimalusena kasutada nt [Wekat](#). Weka on avatud lähtekoodiga masinõppe tarkvarapakett, mille on välja töötanud Uus-Meremaa Waikato ülikool ning mis võimaldab andmete analüüsi, mudelite loomist ja visualiseerimist erinevate algoritmide abil. Kuna see on loodud kasutades Java programmeerimiskeelt, siis sobitub see ka KEMITi IT-profiiliga.

Selleks, et taotlust automaatselt kontrollida, luuakse esiteks vastav mudel. See eeldab piisavas koguses algandmete olemasolu, et loodav mudel oleks võimalikult täpne. Weka graafiline kasutajaliides võimaldab nii mudeli loomist kui ka loodud otsustuspuit graafilist visualiseerimist. Seeläbi on võimalik täpselt näha, mis tingimuste tõttu mingi otsus vastu võeti.



Joonis 44. Mudeli loomise ja kasutamise voog

Seejärel sisestatakse loodud mudel AI mikroteenusesse, mis sisendandmete põhjal tagastab tulemuse koos kindlustunde väärtusega, mis näitab protsentides, kui suure tõenäosusega tagastatav vastus on korrektne. Vastavalt kindlustunde väärtusele saab Protsessimootor teha otsuse, kas arvestada mudeli poolt tagastatud tulemust või mitte.

4.2 Klienditugi

TARVIK on üha kasvava funktsionaalsusega infosüsteem. Ka suurepärase kasutajakogemuse korral võib kasutajal tekkida küsimusi, nt millist taotlust on vaja mingi loa jaoks täita või mis väärtust sisendis oodatakse. Tõsi, selle jaoks võib luua põhjalikud juhendid, mida kasutaja läbi saab töötada, kuid palju mugavam oleks küsida vastust klienditoelt, mis koheselt vastaks. **Siinkohal tulevad appi keelemudelid.**

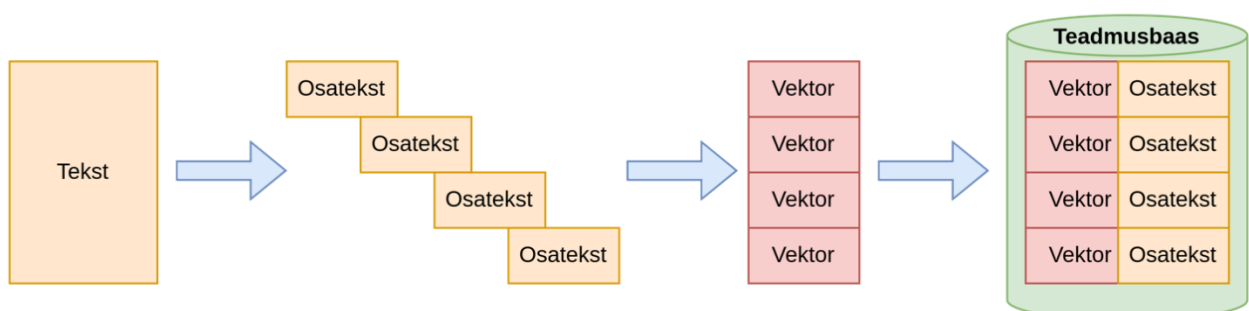
Probleem tekib aga keelemudeli teadmistes, kuna sellel puudub info TARVIKu kohta. Selle probleemi lahendamiseks kasutatakse tehnikat nimega RAG (*Retrieval-Augmented Generation*). RAG kombineerib infootsingu ja keelemudeli teksti genereerimise võimekuse. Kõigepealt otsitakse andmeallikast (TARVIKu juhendid) asjakohast teavet, mis antakse keelemudelile koos küsimusega kaasa, et mudelil oleks olemas kontekst, mille põhjal vastata.

Tehniline lahendus

Esmalt on vaja teadmusbaasist otsida üles relevantne info, mida keelemudelile kaasa anda. Iga päringuga keelemudelile kogu andmeallika kaasa andmine pole mõistlik ega praktiline, kuna see suurendab nii võrguliiklust kui ka keelemudeli kasutamise kulu.

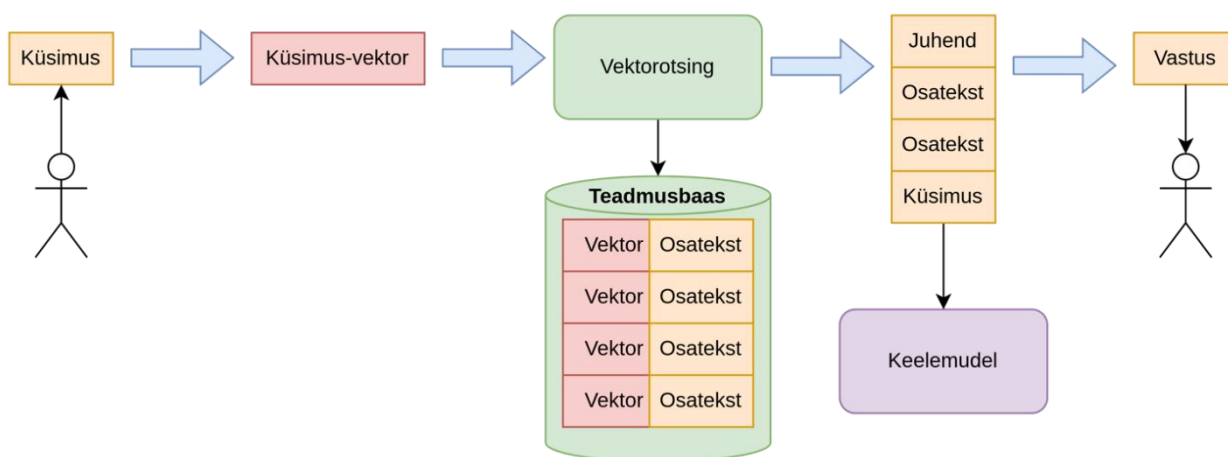
Teadmusbaasi loomine koosneb järgnevatest sammudest:

- Esmalt tükeldatakse dokumendid väiksemateks osadeks mõningase ülekattega säilitamaks konteksti. Tükeldamine on vajalik, kuna keelemudelitel on piirangud, kui pikka teksti nad suudavad vastu võtta.
- Seejärel genereeritakse dokumendiosadest vektorid (*embeddings*). Need vektorid talletavad teksti semantilist sisu, mitte sõnu. Sarnase sisuga tekstide vektorid on teineteisele lähemal.
- Vektorid koos tekstiga salvestatakse andmebaasi. Andmebaasisüsteemiks sobib KEMITi IT-profiilis välja toodud PostgreSQL koos [pgvector](#) laiendiga, mis võimaldab vektorandmeid talletada kui ka nende seast otsida. Tänu vektoritele on nüüd võimalik üles otsida relevantssed osatekstitid.



Küsimuse põhjal vastuse genereerimine koosneb järgnevatest sammudest:

- Kasutaja küsimusest genereeritakse vektor.
- Teadmusbaasist otsitakse üles küsimuse-vektorile lähedal asuvad osateksti-vektorid. Kuna antud vektorid talletavad endas teksti semantilist tähendust, siis leitud vektoritega seotud osatekstid sisaldavad relevantset informatsiooni. Leitud osatekstid annavad keelemudelile konteksti.
- Leitud osatekstide, küsimuse ning juhise põhjal luuakse sisend (*prompt*), mis saadetakse keelemudelile. Keelemudel genereerib antud sisendit arvesse võttes vastuse ja tagastab selle. Juhised on vajalikud selleks, et anda keelemudelile teada, mida temalt täpselt oodatakse.



Joonis 46. RAG. Küsimusele vastuse genereerimine vektorandmebaasis olevate andmete pealt

Keelemudeli poolt tagastatud vastuse sobivus sõltub olulisel määral keelemudelile saadetavast sisendist (*prompt*). Selleks, et suunata keelemudelit andma soovitud ja täpseid vastuseid kasutatakse [prompt-engineering](#) tehnikaid. See hõlmab õige sõnastuse, konteksti ja juhiste valimist, et maksimeerida mudeli kasulikkust konkreetse ülesande lahendamisel. Hästi kujundatud sisend võib oluliselt parandada mudeli täpsust, loogikat ja järjepidevust vastustes.

Keelemudeli poolt tagastatud vastus võib sama sisendi puhul iga kord erineda. Samuti ei saa alati kindel olla vastuse korrektsuses. Seetõttu ei sobi keelemudelite kasutamine suurt täpsust ja läbinähtavust vajavate protsesside (nt menetlemine) automatiseerimiseks. Küll aga sobib see suurepäraselt tekstist andmete otsimiseks ja selle põhjal küsimustele vastamiseks ning kasutaja abistamiseks.