Arhitektuuri visioon

Originaalfailid: TTJA arhitektuur v1.zip

1. Sisukord

2. Äriarhitektuur

Tarbijakaitse ja Tehnilise Järelevalve Amet (edaspidi TTJA)[1] on loodud 2019 aastal mitme riigiasutuse ja ameti liitmisel ja tegutseb põhimääruse[2] alusel. Nimetatud konsolideerimise tulemusena sündis ühendametkond, millel on üsna lai tegevuste spekter, ja seetõttu tähtis roll Eesti riigis ja ühiskonnas.

Ameti põhitegevuseks on ohutusjärelevalve, tururegulatsioon ning seadusest tulenevate kohustuste täitmise kontrollimine järgmistes valdkondades:

- elektrooniline side, sagedushaldus ja meediateenused;
- raudteetransport ja EL struktuurivahendite rakendamine;
- eripädevusnõuetega tööde ning seadmete ja toodete ohutus;
- ehitised, taristu ja energiatõhusus;
- tarbijaõigused.

TTJA pakub oma ülesannete täitmisel järgmisi teenuseid:

- tegevus- ja kasutusõiguse andmine (tööstus-, ehitus- ja raudteeohutus, elektrooniline ja raadioside);
- riiklik järelevalve (tööstusohutus, ehitusvaldkond, energiatõhusus, raudteeohutus, elektrooniline side ja meediateenused, raadiosageduste kasutamine, tarbijaõigused);
- nõustamistegevus;
- tarbijavaidluste lahendamine.

TTJA asutamisega võttis organisatsioon üle erinevatelt asutustel ka nende asutuste protsessid ja protsesse teotavad infotehnoloogilised lahendused sh. infosüsteemid ja andmebaasid.

TTJA strateegiliste eesmärkide[3] elluviimiseks on tarvilik reformida nii olemasolevaid äriprotsesse, kui ka neid toetavaid infosüsteeme. Planeerida süsteemi muudatused vastavalt käesoleva aja parimetele infotehnoloogilistele praktikatele ja kaaluda ka tulevikus potentsiaalsete innovaatiliste lahenduste kaasamist infotehnoloogia arendamisel, testimisel, juurutamisel ja haldamisel. Olemasolevad süsteemid on nii moraalselt, kui ka tehniliselt vananenud ja nende edasiarendamine on osutunud ebamõistlikuks.

3. Andmearhitektuur

Käeolevas peatükis tutvustame süsteemi andmearhitektuuri, klassifitseerime andmed ja tutvustame põhilisi süsteemis kasutatavaid andmesalvestustehnoloogiaid. Samuti nende rakendamist süsteemi osade disainimisel.

Andmed üldiselt jagunevad mitmesugusteks. Käesolevas dokumendis klassifitseerime andmed loogilisteks gruppideks ja käsitleme igat gruppi eraldiseisvalt, näiteks protsessi andmed, binaarandmed, suurandmed, avaandmed ja äriprotsesside põhiandmed.

3.1. Põhiandmed

Põhiandmete all mõistame andmeid, mis on eelkõige kirjeldatud õigusaktides näiteks seadus, andmekogu põhimäärus, ministri määrus, korraldus vms. Nende andmete seas on äriprotsessi käigus kogutud ja tekkinud põhilised andmed näiteks avaldused, otsused, õigused jne. Nendel andmetel on määratud andmeturbest lähtuv ISKE turvaklass[4], kogumise alus ja säilitamise tähtaeg. Põhiandmetega toimub äriprotsess (menetlus), mille käigus olemasolevaid andmeid muudetakse või tekib juurde uusi põhiandmeid. Põhiandmed omavad ka väljaspool IT süsteemi tähtsust ja tähendust. Põhiandmeid jagatakse ühiskonnale välja avaandmetena ja teiste riigi äriprotsessidele üle riikliku andmevahetusplatvormi.

TTJA vaates on baasandmed kirjeldatud alljärgnevates õigusaktides:

MTR

- Majandustegevuse seadustiku üldosa seadus (lühend – MSÜS)[5]

JvIS

- Seadme ohutuse seadus https://www.riigiteataja.ee/akt/123032015004?leiaKehtiv
- Tarbijakaitse ja Tehnilise Järelevalve Ameti järelevalve infosüsteemi põhimäärus https://www.riigiteataja.ee/akt/124032020010

NBA

- Elektroonilise side seadus (ESS) https://www.riigiteataja.ee/akt/112122018033?leiaKehtiv
- Majandus- ja kommunikatsiooniministri määrus "Nõuded numbri liikuvuse tagamiseks sideettevõtja vahetamisel" https://www.riigiteataja.ee/akt/126022019012?leiaKehtiv

- Majandus- ja kommunikatsiooniministri määrus "Numbri broneerimise tingimused" https://www.riigiteataja.ee/akt/103072015016?leiaKehtiv
- Directive 2002/22/EC of the European Parliament and of the Council on universal service and users' rights relating to electronic communications networks and services (Universal Service Directive) https://ec.europa.eu/digital-single-market/en/news/directive-universal-service-and-users-rights-relating-electronic-communications-networks-and
- Directive 2002/58/EC concerning the processing of personal data and the protection of privacy in the electronic communications sector and Regulation (EC) https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=celex%3A32002L0058
- No 2006/2004 on cooperation between national authorities responsible for the enforcement of consumer protection laws (Text with EEA relevance). https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex%3A32004R2006

SASS

- Elektroonilise side seadus (ESS) https://www.riigiteataja.ee/akt/112122018033?leiaKehtiv
- Majandus- ja kommunikatsiooniministri määrus "Raadiosageduste kasutamise tingimused ja tehnilised nõuded sagedusloast vabastatud raadioseadmetele" https://www.riigiteataja.ee/akt/126022019014?leiaKehtiv
- Majandus- ja kommunikatsiooniministri määrus "Tehnilised nõuded sagedusloa alusel kasutatavatele raadioseadmetele" https://www.riigiteataja.ee/akt/117062014008?leiaKehtiv
- Komisjoni otsus teabe kättesaadavuse ühtlustamise kohta seoses raadiospektri kasutamisega ühenduses 2007/344/EÜ https://eur-lex.europa.eu/legal-content/ET/TXT/?uri=CELEX:32007D0344

GIS

- Elektroonilise side seadus (ESS) § 1002 https://www.riigiteataja.ee/akt/112122018033?leiaKehtiv
- Sideteenuse katvuse, kasutuse ja võimaluste kaardistuse infosüsteemi põhimääruse kinnitamine https://adr.mkm.ee/?id=72B0329323B341B2C2258081002C4D39

Antud nimekiri pole ammendav ega lõplik.

3.2. Protsessi andmeid

Protsessiandmete all mõistame andmeid, mis tekivad või muutuvad äriprotsessi (menetluse) läbimise käigus, kuid kirjeldavad ainult konkreetset äriprotsessi ja selle omadusi. Nendeks võib lugeda protsessi andmeühikuks millal protsess käivitus (algatati menetlus), milline asutuse töötaja või osapool konkreetselt antud protsessis osaleb, mis toiminguid protsessis tehakse, äriprotsesi staatus jne. Protsessi tehniline kirjeldus (masin-töödeldav algoritm või meta-keelne teisend) võivad ise ka olla protsessi andmed (BPMN notatsioonis kirjeldus). Protsessi andmete alla võib lugeda ka äriprotsessi seadistusi ja muutujate väärtuseid.

3.3. Binaarandmed

Binaarandmete all mõistame andmeid, mille esitluskuju on binaarne, üldjuhul ei ole binaarandmed mõistlikul või üldlevinud viisil masin loetavad ja -töödeldavad. Neil ei ole primaarset teksitilist esitluskuju. Eelkõige on binaarsel kujul failid näiteks pildifailid, PDF dokumendid, heli ja videoklipid jne.

3.4. Suurandmed

Suurandmeteks nimetatakse selliseid andmekogumeid, mis on nii suured ja keerukad, et nende töötlemiseks tuleb kasutada uusi tehnoloogiaid[6]. Neid andmeid iseloomustavad üldjuhul suur maht, suur juurdetekkimise kiirus ja suur variatiivsus[7].

TTJA-s tegeleb suurandmetega antud dokumendi loomise ajal projekt "TTJA Andmeait".

Käesolevas dokumendis kirjeldatud mikroteenustel arhitektuuri elluviimisel muutub oluliselt süsteemi andmemudel sh. võivad kasutsuele tulla uued tehnoloogiad ja andmesalvestuse vahendid. Andmeaida arendamisel peab silmas pidama uute andmemudelite ja andmeformaatide temaatikat ja vajadusel tegema koostööd uue platvormi välja töötajatega.

3.5. Avaandmed

Avalikud avaandmed (valitsusandmed) tähendavad andmeid, mille on kogunud, tootnud või mille eest on tasunud avaliku sektori asutused ning mis on muudetud vabalt kättesaadavaks ja mis tahes otstarbel taaskasutatavaks. Kasutamistingimused on täpsemalt määratletud litsentsis.[8]

Avaandmetega seonduv on juhendmaterjalina avaldatud Avaandmete loomise ja avaldamise juhendis[9] ja Eesti avaandmed on koondatud Eesti avaandmete portaali[10].

Käesolevas dokumendis kirjeldatud süsteemis luuakse ja jagatakse avaandmeid. Avaandmete loomise ja avaldamise tehniline lahendus on kirjeldatud peatükis 4.2.12.

3.6. Andmemudelid ja andmete hoidmise tehnoloogiad

Käesolevas alapeatükis kirjeldame lähemalt võimalikku andmete säilitamise ja töötlemise põhimõtteid ja lahendusi. Põhiliselt käsitleme kolme olulisemat andmete säilitamise viisi a) relatsiooniline, b) dokument orienteeritud ja c) blokkhoidla. Eelkõige keskendume füüsilisele andmemudelile, mitte loogilisele. Loogiline andmemudel oma üleüldiselt joondub rohkem relatsioonilise andmemudeliga.

3.6.1. Relatsiooniline andmemudel

Relatsiooniline andmebaas põhineb relatsioonilisel mudelil ehk baasi loogiline struktuur koosneb relatsioonide kogumist. Relatsioonilise andmemudeli kontseptsiooni esitas 1970. aastal Edgar Frank Codd. Sünonüümselt võib relatsiooni nimetada ka tabeliks, mille struktuur koosneb olemi atribuutidest ja andmekogumitest, teisisõnu veergudest ja ridadest, kus iga veerg saab hoida ainult ühte tüüpi andmeid[11]. Traditsioonilistes andmebaasi mootorites on relatsiooniline andmemudel implementeeritud tabelite, mis koosnevad veergudest ja ridade gruppidest, mis on loogiliselt omavahel seotud viidete ja viitetabelite (vahetabelite) kaudu. Füüsilise adnmemudeli saamiseks tuleb loogilist andmemudelit normaliseerida.

3.6.2. Dokumentandmemudel

Viimase dekaadi jooksul on infotehnoloogias kogunud populaarsust termin NoSQL[12], mis on saanud sünonüümiks dokument orienteeritud andmehoidlatele. Dokument andmemudeli üheks omaduseks võib olla, et otseselt ei eksisteeri objekti iseloomustavat tabelit, vaid eksisteerib mingis tekstilises andmevormingus näiteks JSON andmekomplekt, mida nimetatakse „dokumendiks". Dokument orienteeritud andmebaasi süsteemid on näiteks Apache CouchDB, MongoDB. Nimetud andmebaasimootorite kohta kehtiks küll termin NoSQL, kuna nendes kasutamiseks ei kasutata SQL süntaksit. Küll aga on dokument orienteeritud andmete säilitamine võimalik ka klassikaliste relatsiooniliste andmebaasimootoritega nagu näiteks PostgreSQL. Sealt edasi on võimalik ehitada ka hübriidlahendusi, kus vastavalt vajadusele on võimalik mõlemat andmemudelit kombineerida.

Kuna JSON andmeformaat on *de facto* andmevorming IT-maailmas, siis käsitletakse dokument andmebaasides praktiliselt ainult JSON andmekomplektide hoiustamist. Muus vormingus dokument andmemudelid on marginaalsed. Näiteks on ajalooliselt XML andmete andmebaasis hoidmine olnud võimalik pea samal viisil, kui JSON andmete, kuid sellel on olnud sisulised ja fundamentaalsed puudused, mistõttu ei ole selle kasutamine juurdunud. JSON andmeformaadis andmekomplekti näidis on kujutatud joonisel 1 ja JSON Schema[13] samale andmekomplektile on kujutatud joonisel 2.

Oluline on ka asjaolu, et dokument-andmemudel ei asenda relatsioonilist andmemudelit kõigis stsenaariumites ja dokument andmemudel ei sobi kõigi ärijuhtude tarbeks andmete säilitamiseks. Kuna relatsiooniline andmemudel on eksisteerinud pikka aega on andmebaasi mootorid optimeeritud töötama relatsiooniliste andmetega. On spetsiifilised funktsioonid ja operatsioonid mille saavutamine dokument orienteeritud andmebaasis võrreldes relatsioonilisega on oluliselt keerulisemad.

Dokument orienteeritud andmemudelit on võimalik defineerida läbi meta-skeemi. JSON andmevormingu puhul on võimalik defineerida andmekomplekti kasutades JSON Schema funktsionaalsust. Käesoleval hetkel ei toeta PostgreSQL JSON Schema kasutamist otse andmebaasi funktsionaalsel tasemel, seega tuleb andmekomplekt kirjeldada andmebaasi väliselt või hüübriidmudeliga. PostgreSQL JSON Schema toega on olemas mitmeid tarkvaraarenduse teeke, mis selle probleemi näiteks Java keeles ära lahendavad.

```
{
    "eesnimi": "Rein",
    "perekonnanimi": "Põld",
    "vanus": 50
}
```

Joonis 1. Näidis JSON andmekomplekt

```
{
    "$schema": "https://json-schema.org/draft/2019-09/schema",
    "$id": "http://ttja.ee/example.json",
    "type": "object",
    "title": "Põhiobjekt",
    "required": [
        "eesnimi",
        "perekonnanimi",
        "vanus"
    ],
    "properties": {
        "eesnimi": {
            "type": "string",
            "title": "Isiku eesnimi"
        },
        "perekonnanimi": {
            "type": "string",
            "title": "Isiku perekonnanimi"
        },
        "vanus": {
            "type": "integer",
            "title": "Isiku vanus"
        }
    }
}
```

Joonis 2. näidis JSON skeem

Antud näite JSON skeemist järeldub, et andmekomplektis on kolm kohustuslikku atribuuti eesnimi, perekonnanimi ja vanus. Samuti on kirjeldatud nende lihttüüp ja määratud JSON Schema lisaatribuut „*title*”, mida saab kasutada, et kirjeldada atribuuti. JSON Schema-s on lisaks veel mitmeid atribuute, mida ära kasutada JSON objekti kirjeldamiseks. JSON andmekomplekti on võimalik vastu skeemi valideerida ja kontrollida.

3.6.3. Blokkandmehoidla

Andmeid, millised on kirjeldatud peatükis 3.3 tuleb hoida blokkandmete hoidlas. Käesoleval ajal on kujunenud üleüldiseks standardiks kasutada blokkandmete hoidlateks tehnilisi lahendusi, mis pakuvad andmete salvestamist ja ligipääsu andmetele kasutades Amazon S3 blokihoidmise lähenemisega. S3 on üldtunnustatud protokoll binaarandmetele ligipääsuks. Riigipilv pakub blokkandmehoidla teenust Pilw.io StorageVault[14]. Blokihoidla puuduseks on asjaolu, et selles, ei saa hoida binaarandmete-failide kohta käivat metainfot näiteks faili tüüp, laiend, omanik, märksõnad, viited indeksitele jne. Selleks tuleb luua süsteemi komponent, mis haldab binaarandmete metainfot.

3.7. Andmemigratsioon ja sünkronisatsioon

Uuele platvormile üleminekul peab samal ajal hoidma töös ka vana süsteemi. Kuna vanas süsteemis on ka palju andmeid tuleb neid hakata järk järgult üle viima uuel platvormil arendatud süsteemi.

Detailanalüüsi käigus tuleb täpselt spetsifitseerida, millised andmed, millal ja mis viisil üle kolitakse ja milliseid andmeid sünkroniseeritakse.

Andmemigratsioon on ühekordne tegevus, mida tehakse vanast süsteemist uude süsteemi. Seda tehakse, kas spetsiaalselt loodava mikroteenusega, SQL migratsiooni skriptidega või mõne valmistarkvaraga. Tuleb hinnata eraldi, millisesse andmehoidu lahendusse andmed üle kantakse ja mis migratsiooni strateegiat tuleks antud olukorras kasutada. Relatsiooniliselt andemudelilt dokument orienteeritud andmemudelile üle viimisel on soovituslik kasutada mikroteenust, kus tegevuste käigus toimub ka andmete valideerimine ja vajadusel andmete parandamine.

Andmete sünkroniseerimisel on sama andmestik olemas mõlemas süsteemis, nii vanas kui ka uues. Sünkroniseerimiseks kasutatakse eraldi arendatud mikroteenust, mille ülesanne on hoida uue ja vana süsteemi andmebaasid samade andmetega, kui selline vajadus on.

Andmete sünkroniseerimisega tegelev võimalik mikroteenus on kirjeldatud peatükis 4.4.6.

Andmete migreerimiseks on mõstlik kasutada vabavaralisi tööriistu näiteks *pgloader*, replicatsiooni puhul nt *pglogical*.

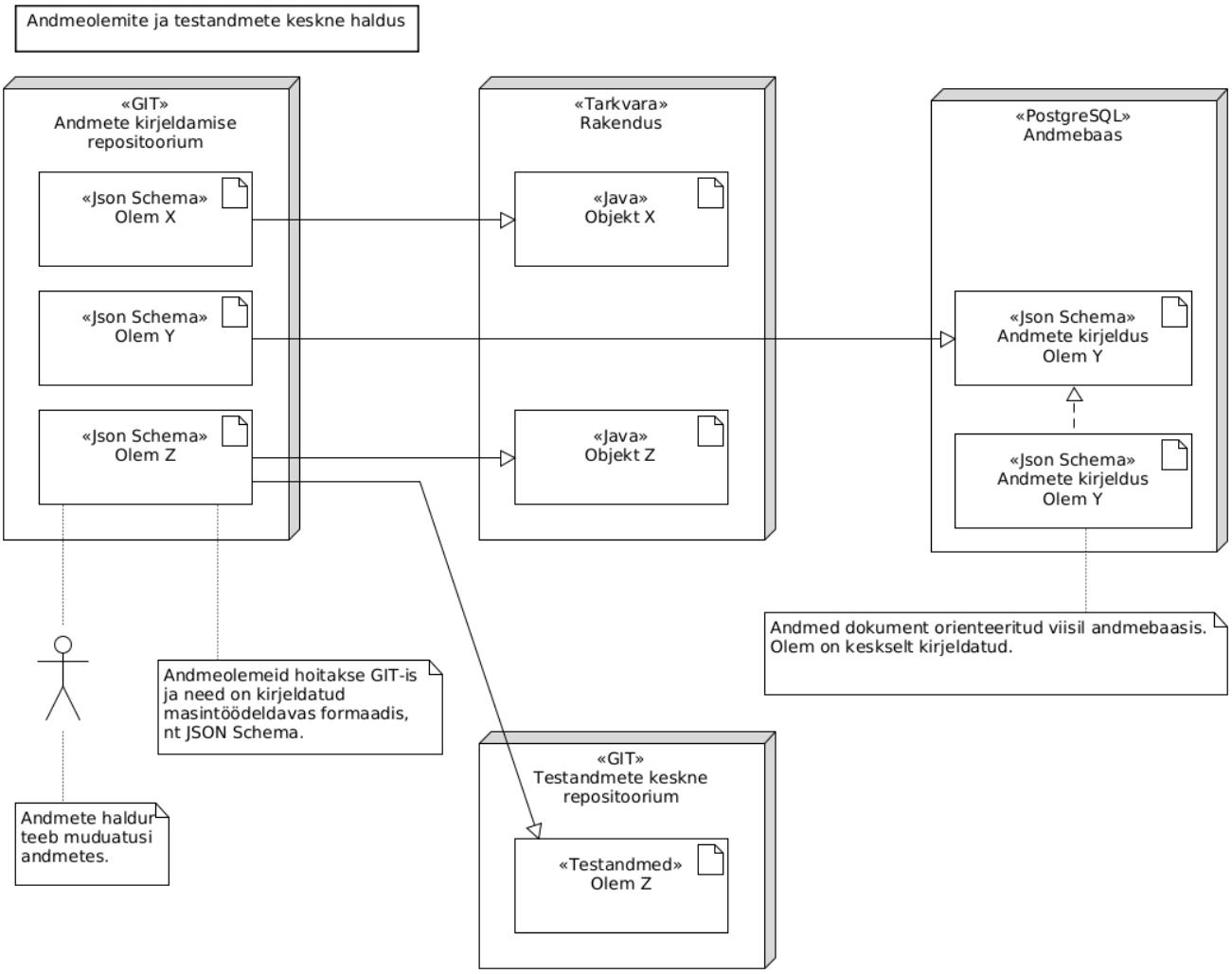
3.8. Andmeolemite keskne haldus

Suures organisatsioonis, kus käitatakse palju erinevaid äriprotsesse on kasutusel paralleelselt palju erinevaid andmeolemeid. Andmeolemid läbivad kõiki kihte, kuni tehnilise kihini välja. Andmeolemite üle tuleks pidada arvestust ja neid keskelt hallata. Keskselt hallatud andmeolemid tuleks kirjeldada kasutades JSON Schema standardit ja hoiustada GIT koodihoidlas. Mõned andmeolemite keskselt haldamise eelised ja põhimõtted on alljärgnevad:

- Annab hea ülevaate süsteemi andmeolemitest;
- On võimalik kasutada dokumentatsioonina;
- Mainsloetaval kujul andmeolemitest ja nende sostest on võimalik genereerida diagramme;
- Masin loetaval kujul hoiustatud andmeolemeid on võimalik konverteerida tarkvara lähtekoodi klassideks. (Java puhul *POJO*-deks);
- Masin loetaval kujul hoiustatud andmeolemitest on võimalik genereerida testandmeid ja näidisandmeid;
- Kõigil osapooltel on võimalik pääseda ligi andmeolemite kirjeldustele.

Lisaks andmeolemite kesksele haldusele tuleks koos andmeolemitega keskselt hallata ka testandmeid ja näidisandmeid.

Joonisel 3 on kujutatud andmeolemite ja testandmete keskset haldust.



Joonis 3. Andmeolemite ja testandmete keskne haldus

4. Tarkvara arhitektuur mikroteenustel

Selles peatükis kirjeldame süsteemi kontseptuaalne arhitektuuri ja selgitame detailsemalt arhitektuuri osi. Süsteemi kirjeldamiseks oleme kasutanud sõna „moodul", mis on antud dokumendi raames mikroteenuse sünonüüm. Kogu süsteemi oleme klassifitseerinud viieks moodulite blokiks 1) Tugimoodulid, 2) Andmevahetusmoodulid, 3) Andmemoodulid, 4) Äriprotsessi moodulid ja 5) Kasutajaliideste moodulid. Igas alampeatükis on detailsemalt selgitatud vastava domeeni moodulite omadused ja sisu. Samuti on eraldi alampeatükis selgitatud moodulite vahelised andmevahetus põhimõtted.

4.1. Kontseptuaalne arhitektuur

Käesolevas alapeatükis pakume välja loodava süsteemi kontseptuaalse arhitektuuri, esitledes süsteemi komponente ja mooduleid, nende funktsioone ja käitamise asjaolusid.

Mikroteenuste arhitektuuri põhimõtted on alljärgnevad:

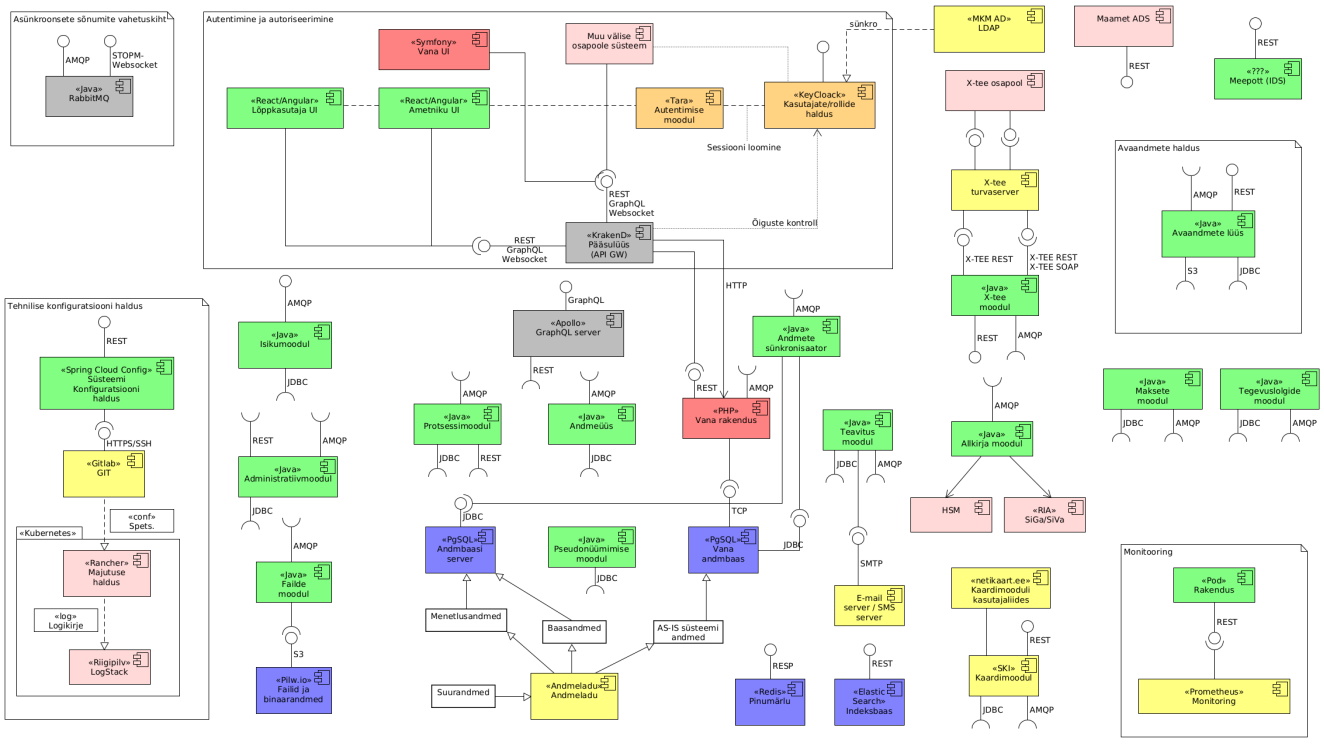
- Mikroteenus on ühte või mitut sarnast funktsiooni realiseeriv eraldi seisvalt paigaldatav autonoomne süsteemi komponent;
- Mikroteenus võib realiseerida ühte funktsiooni või mitut sarnaselt grupeeritud funktsiooni;
- Mikroteenus on teistest komponentidest otseselt mitte sõltuv iseseisev komponent;
- Mikroteenus on majutuskohas iseseisvalt skaleeritav ja hallatav komponent;
- Mikroteenuste arhitektuur võimaldab pakendatud funktsionaalsust sõltumatult, eraldiseisvalt arendada ja testida;
- Mikroteenusteks võib jagada funktsioone ka administratiivsete või projektorganisatoorsete asjaolude järgi;
- Mikroteenusel on enda andmebaas või pole seda üldse;
- Mikroteenus pöördub teise mikroteenuste (andmete) poole läbi programmliidese (API) või keskse andmesiini;
- Mikroteenuste arhitektuur käsitleb üldjuhul tagarakenduste ja kommunikatsioonirakenduste skoopi, mitte aga kasutajaliideste skoopi.

Eelnev nimekiri pole lõplik ega ammendav.

Süsteemi kontseptuaalne arhitektuur on kirjeldatud joonisel 4. Joonise paremaks loetavuseks on süsteemi osad tähistatud erinevate värvidega alljärgnevalt

- a) roheline – uued komponendid, mis tuleb arendada (või taaskasutada mingit tarkvara),
- b) kollane – süsteemid, mis eksisteerivad,
- c) oranž – autentimise ja autoriseerimise moodulid,
- d) sinine – andmebaas või andmehoidlad,
- e) punane – olemasolevad taakvarad (*legacy software*),
- f) roosa – muu väline osapool või tugisüsteem,
- g) hall – kommunikatsiooni moodulid, mis tuleb luua.

Joonisel toodud seostes on päringu teenindaja märgitud joone ja mummuga või kriipsjoone ja noolega, päringu tarbija (võib olla päringu algataja) on märgitud poolkaarega, teenuse pakkuja on märgitud mummuga. Seose peale võib olla märgitud andmevahetuse protokoll. Eraldi äärisega on tähistatud valdkonnad, millistel on täpsemad joonised käesoleva dokumendi alapeatükkides. Noolega on tähistatud andmet sõltuvuse suund või päringute liikumise suund.



Joonis 4. Süsteemi kontseptuaalne arhitektuur

Rõhutame, et näidatud arhitektuur on kontseptuaalne. Arenduste ja süsteemi loomise käigus võib tekkida vajadusi luua juurde mooduleid, milliseid pole võimalik käesolevas dokumendis ette näha. Küll aga tuleb arvestada, et moodulite arvu suurenemisel suureneb ka halduskeerukus ja süsteemi evitusega senduvad väljakutsed. Mida rohkem on mikroteenuseid seda keerulisem on evitus.

4.2. Tugimoodulid

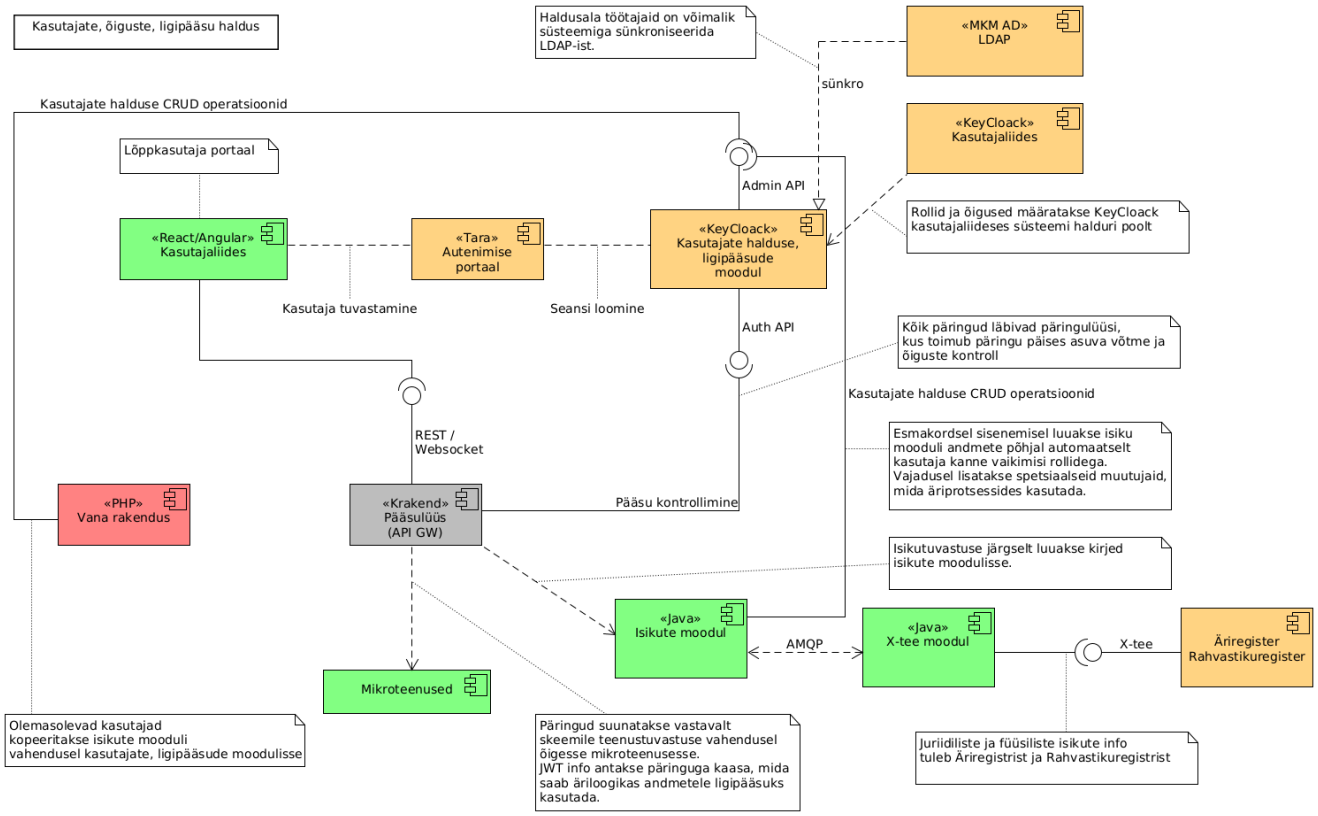
Käesolevas peatükis käitleme detailsemalt tugimooduleid, nende funktsioone ja tehnilisi lahendusi.

4.2.1. Identiteedi ja pääsuhalduse moodulid

Kasutajate, rollide, õiguste ja nende omavahelise halduse, pääsu saladuste jne hoidmiseks tuleb luua eraldi moodul Kõik süsteemis kasutusel olevad õigused ja rollid asuvad käesolevas moodulis. Kõik süsteemi tehtavad päringud autoriseeritakse vastu seda moodulit, et kontrollida pääsuõiguse kehtivust. Moodul genereerib, haldab ja väljastab JWT võtmeid[15].

Käesoleva mooduli tarkvaraliseks lahenduseks tuleks kasutada Keycloak[16] tarkvara, mis on üks levinum avatud lähtekoodiga identiteedi ja pääsuhalduse valmistoode.

Joonisel 5 on kujutatud täpsemalt identiteedi ja pääsuhalduse korraldus mikroteenuste arhitektuuris.



Joonis 5. Kasutajate, õiguste, ligipääsude haldus mikroteenuste arhitektuuris

Kasutaja interaktsiooni esimene samm toimub kasutajaliideses, kust suunatakse kasutaja riiklikusse autentimisportaali. Autentimisportaal edastab tuvastatud identiteedi andmed kasutajate halduse, ligipääsu moodulisse, kus seotakse need kasutajaga. Kasutaja puudumisel luuakse kasutaja ja seotakse vaikimisi rollidega isikute mooduli andmete põhjal. Tuvastatud kasutaja Internetisirvikule edastatakse JWT võti[17], mis sisaldab informatsiooni kasutaja identiteedi kohta. Edaspidistele päringutele süsteemi suunas paneb sirvik JWT võtme kaasa mistõttu, saab pääsulüüs, või mis tahes mikroteenus kontrollide päringu vastu võtmisel JWT võtme autentsust ja päringu tegija õigust päringut sooritada. Kasutajate halduse moodulisse saab lisada kasutaja juurde andmeid, mille abil on võimalik seostada kasutajat-isikut mingi juriidilise kehaga või mingite andmetega. Kuna JWT võtit saab kaasa anda ka mikroteenustele, siis on võimalik seoste informatsiooni põhjal andmetele ligipääsu kontrollida.

4.2.2. Autentimise moodul

Autentimise mooduli ülesanne on korraldada kasutajate autentimist ja vahendada kasutajale sessiooni tunnust koos vajaliku meta-informatsiooniga.

Autentimise mooduliks kasutame RIA TARA lahendust, mis on iseseisvalt hallatav valmis tarkvaratoode. Moodul sisaldab tuge mitmesugustele autentimislahendustele ja on integreeritav Keycloak pääsuhalduse tarkvaraga.

4.2.3. Isikute moodul

Isikumooduli ülesanne on hallata ja säilitada informatsiooni sh. ajalugu süsteemis eksisteerivate juriidiliste- ja füüsiliste isikute kohta. Isikute moodul suhtleb süsteemi väliste osapooltega, mis sisaldavad isikute infot. Näiteks Äriregister – juriidiliste isikute andmed, Rahvastikuregister – füüsiliste isikute andmed. Andmevahetus nimetatud registritega käib üle X-tee vastava mooduli vahendusel. Isikute moodul on isikuandmete „tõe allikas" süsteemi jaoks.

Isikute moodul peaks olema Java keeles arendatud tarkvara või TEHIK TEIS projekti arendatud *persons-service*[18] kloon või võrdväärne.

4.2.4. Administratiivmoodul

Administratiivmooduli ülesanne on koondada süsteemi administratiivsed funktsioonid. Nendeks on näiteks:

- Klassifikaatorite haldus;
- Äriliste parameetrite haldus;
- Tõlgete haldus;
- Abitekstide haldus;
- Infotekstid;
- Aadresside, asukohtade haldus (ADS) – käesolevas dokumendis toodud välja ka eraldi moodulina.

Äriliste parameetrite halduse ja teenustele pakkumise funktsionaalsuse saaks delegeerida ka tehnilise konfiguratsiooni moodulile.

Administratiivmoodul moodul peaks olema Java keeles arendatud tarkvara.

Kuna administratiivmooduli käsitletud andmed on üldjuhul üsna staatilised, siis tuleks need võimalikult palju hoida rakenduse või minumälu baasis. Nii on võimalik tagada päringutele kiired vastamised.

4.2.5. Maksete moodul

Maksete mooduli ülesanne on koondada maksete teostamise ja riigilõivude kogumise funktsioonid. Nendeks on näiteks:

- Riigilõivude informatsiooni hankimine;
- Riigilõivude hinnakirja haldamine;
- Viitenumbrite genereerimine;
- Makseplatvormi pakkumine;
- Liidestumine maksevahendajaga, pangalinkidega;

Maksete moodul peaks olema Java keeles arendatud tarkvara. Näidiseks saab võtta näiteks TEHIK TeIS projekti käigus arendatud tarkvaramooduli payments-service[19].

4.2.6. Failide moodul

Failide mooduli eesmärk on vahendada teistele moodulitele binaarandmeid. Failide salvestamist oleme käsitlenud peatükis 4.2.6. Failide mooduli andmestikus salvestatakse ka failide metainfo.

Allkirja moodul peaks olema Java keeles arendatud tarkvara või TEHIK TeIS projekti arendatud *signing-service*[20] kloon.

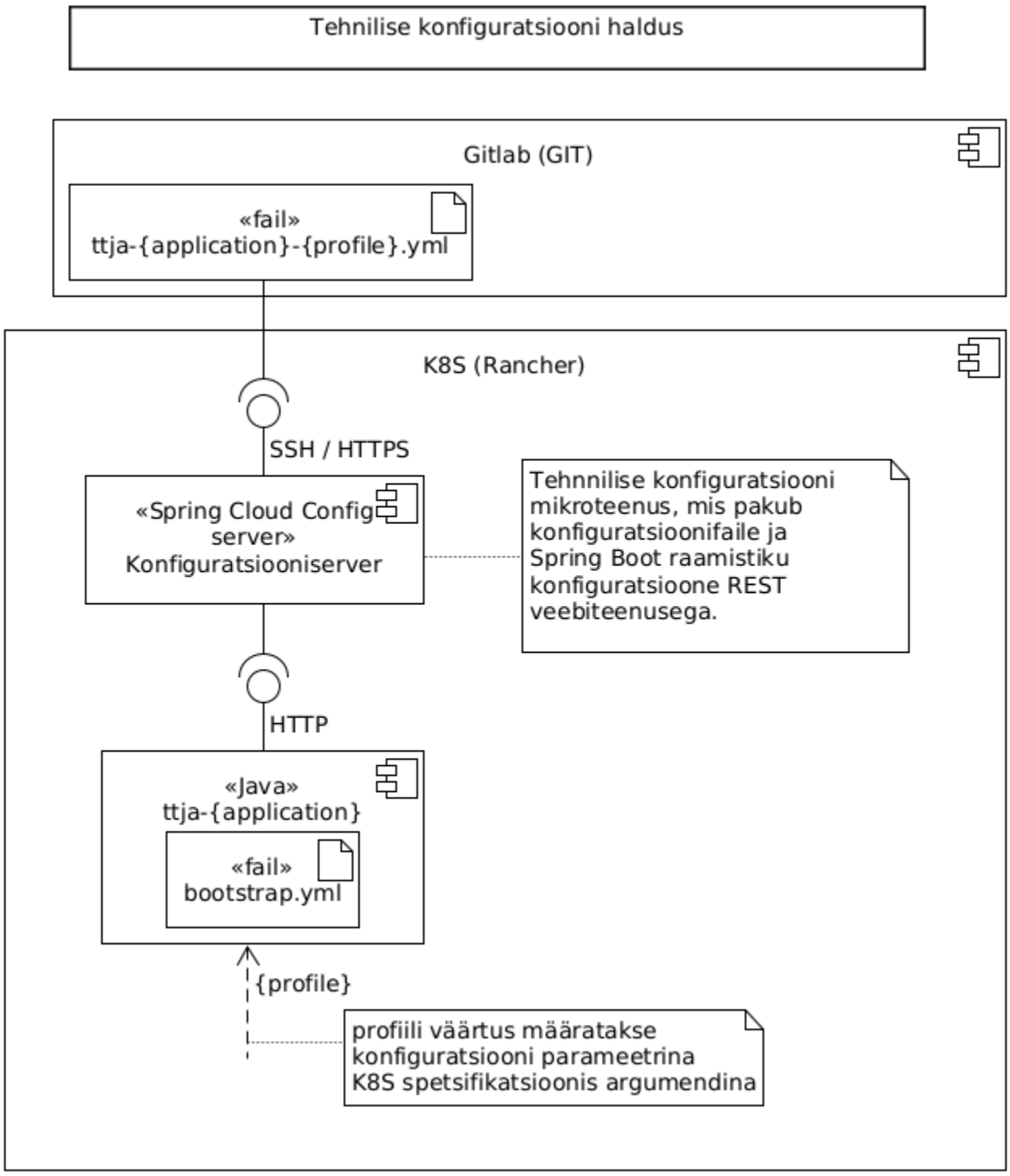
4.2.7. Tehnilise konfiguratsiooni moodul

Tehnilise konfiguratsiooni moodul on vajalik süsteemi teiste moodulite tehnilise rakendusseadete üle programmliidese pakkumiseks. Süsteemis kasutatavad Spring Boot tarkvaraarenduse raamistikul arendatavad rakendused vajavad spetsiifilisel kujul tehnilist konfiguratsiooni. Üks võimalus nende konfigureerimiseks on konfiguratsiooniserver-klient andmevahetus. Mikroteenuse rakenduse käivitamisel küsib see tehnilise konfiguratsiooni moodulilt endale seaded.

Tehniliselt tuleb kasutada Java keeles arendatud *Spring Cloud Config*[22] ja *Spring Cloud Config Client*[23] tarkvaraarenduspaketti, mille andmehoidlaks on näiteks GIT (Gitlab repositoorium piiratud ligipääsuga). Lisaks GIT-ile on toetatud ka mitmed teised andmehoidlad.

Mikroteenused, millised vajavad tehnilist seadistamist kasutavad *Spring Cloud Config Client* teeki ja on seadistatud laadima sisse endale seaded konfiguratsioonimoodulist vastavalt spetsifikatsioonis määratud profiilile.

Joonisel 6 on sealhulgas kujutatud tehnilise konfiguratsiooni mooduli asukoht ja suhtlus osapooltega.



Joonis 6. Tehnilise konfiguratsiooni haldus

Tehnilise konfiguratsiooni mooduli võib ka ära jätta, kui on täielikult rakendatud käesolevas dokumendis peatükis 9.2 kirjeldatud GitOps põhimõtted ja kasutusel süsteemi majutuse halduse operaator tarkvara Argo CD või samaväärne. Sellisel juhul on tehniline konfiguratsioon siiski hallatud GIT-is, kui see antakse rakendusele ette läbi Kubernetes platvormi võimaluste. Pole vaja jagada tehnilisi seadistusi rakendustele üle REST teenuse.

4.2.8. Allkirja moodul

Allkirja mooduli ülesanne on signeerida andmeid ja kontrollida signatuuride korrektsust. Allkirja moodul suhtleb väliste osapoolte ja süsteemidega, mis osalevad signeerimises või signatuuri kontrollimises. Näiteks vajadusel SiVa ja SiGa teenus, riistvara turvamoodul (HSM), ajatempliteenus jne. Kasutades PKC11 protokolli saab moodul suhelda ka riistsvaralise turvamooduliga HSM (Harware Secure Module).

Allkirja moodul peaks olema Java keeles arendatud tarkvara või TEHIK TeIS projekti arendatud *signing-service*[24] kloon.

4.2.9. Teavitus- ja kommunikatsioonimoodul

Teavituste moodul koondab endas funktsioone, millised on seotud kommunikatsiooniga inimeste ja institutsioonide vahel viisil, mis infovahetus pole masintöödeldavas formaadis.

- Tekstimallide haldus;
- PDF faili mallide haldus;
- Lõpptekstide loomine mallidest ja muutujate väärtustest;
- PDF failide loomine;
- E-kirjade loomine ja välja saatmine;
- SMS-ide välja saatmine;
- Osapooltega vahetatud kiirsõnumite haldus.

Loetelu pole lõplik. Konkreetne funktsioonide nimekiri kinnitatakse detailanalüüsiga.

Teavitus- ja kommunikatsioonimoodul moodul peab olema Java keeles arendatud tarkvara, mille puhul võib aluseks võtta näiteks TEHIK TeIS projekti arendatud *messages-service*[25] kloon.

4.2.10. Ärilogi moodul

Mitmel juhul on vaja, et ärilistest tegevustest jääks maha tegevuste logi, mille tähtsaim eesmärk on, et oleks hiljem võimalik üheselt aru saada mis toimus, mis andmeid vaadata või mis andmeid muudeti. Põhimõtteliselt on võimalik ärilogi või ka auditlogi hallata kahel viisil 1) Logikanded salvestatakse andmebaasi, 2) sarnaselt nagu tehniline logi, kuid eraldiseisva andmekomplektiga. Tehnilise logi käitlemine on kirjeldatud peatükis 5.2.4. Äri- ja auditlogi tuleks tehnilisest logist logi pinus (*logstack*) füüsiliselt eraldada, et neid oleks võimalik eraldiseisvalt töödelda ja erineva tähtajaga säilitada.

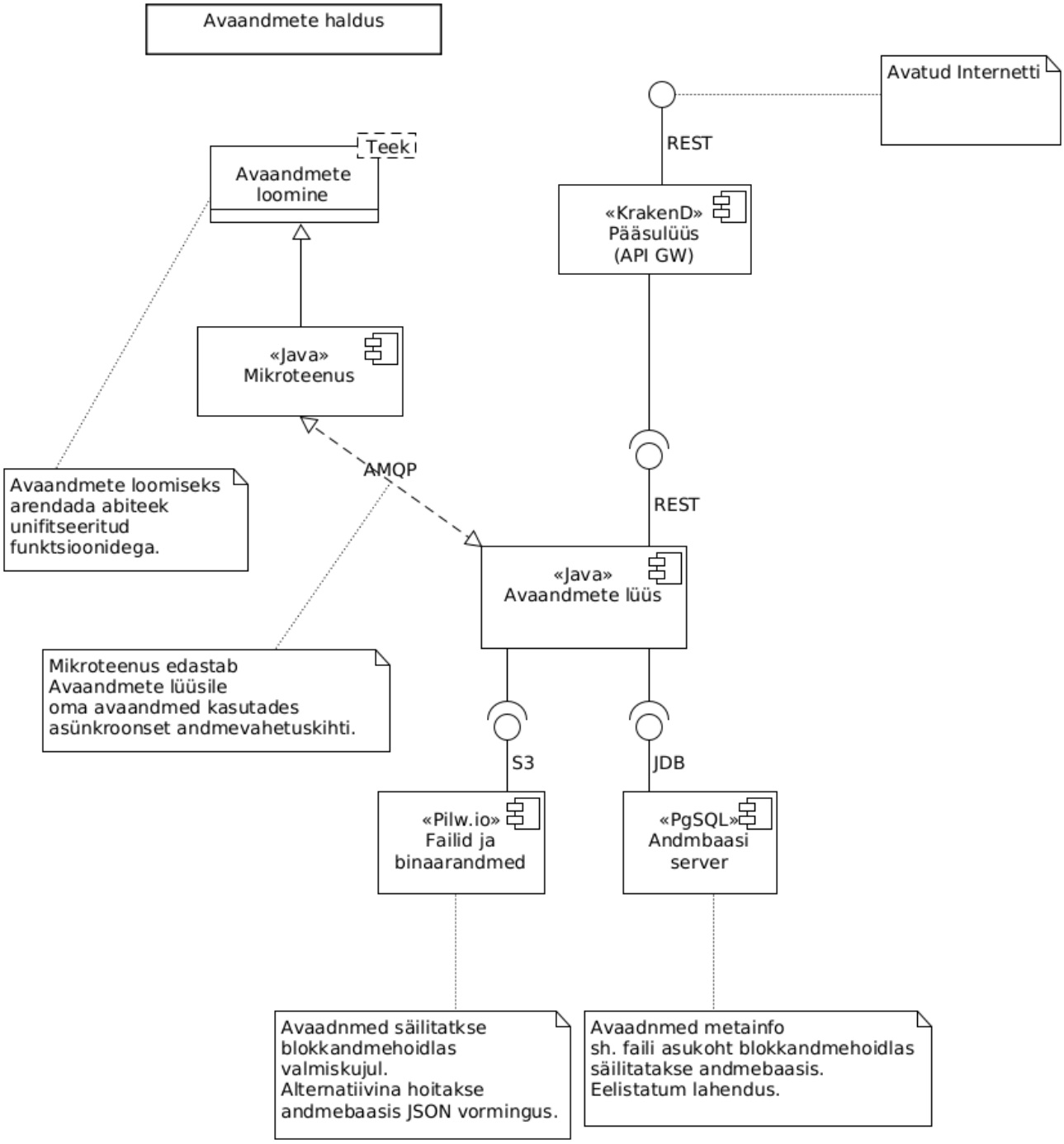
Logide salvestamisel andmebaasi tuleks jälgida, et logide andmebaasi tabelid oleks mõistlikult partitsioneeritud ja oleks võimalik kiired päringud ja tabelid ei läheks ebamõistlikult suureks. Temaatilised logid võib jagada äriliste protsesside mikroteenuste andmebaasidesse.

Kuna tegevuste logi võib tekkida väga palju, siis tuleks eelistada teist varianti. Logide vaatamiseks on võimalik kasutada logipingu koosseisu kuulutav kasutajaliidesega tööriista või arendada kasutajaliides, mille tagarakendus on integreeritud logipinuga läbi programmliidese.

Kuna auditlogide haldamiseks eraldi mikroteenust pole mõistlik teha, siis tuleks unifitseerida ja standardiseerida sääraste logide implementatsioon ja käsitleda seda eraldi tarkvarateegina. Auditlogi andmekomplekt peaks olema unifitseeritud ja sisaldama vajalikke andmevälju. Auditlogi kirjeid peaks olema võimalik seostada infosüsteemi dokumentatsiooniga, et auditi läbiviijatel teadmata täpselt äriloogikat oleks võimalik logikanne siduda äriprotsessi sammuga.

4.2.11. Avaandmete lüüs

Avaandmete lüüs on moodul, mille ülesanne on serveerida Interneti avaandmeid. Joonisel 7 on kujutatud avaandmete halduse tehniline lahendus.



Joonis 7 Avaandmete haldamine ja publitseerimine

Avaandmete lüüs peaks olema mikroteenus, mille ülesanne on süsteemi avaandmeid hallata. Avaandmete loomise eest vastutab valdkondlik mikroteenus (äriprotsessi mikroteenus vms.) Unifitseeritud ja sarnane funktsionaalsus tuleks koondada ühte tarkvarateeki ja seda kasutada avaandmete loomiseks valdkondlike andmete baasilt. Süsteemi äriprotsesside mikroteenused loovad oma valdkonna avaandmeid (vastavalt juhendile[26]) ja edastavad neid avaandmete lüüsile kasutades asünkroonset andmevahetusplatvormi. Avaandmete lüüs vastutab andmete säilitamise ja edasise esitamise eest. Avaandmeid saaks hoida staatilisel, ilmutatud kujul, kas plokkandmehoidlas või andmebaasis JSON vormingus. Esimesel juhul tuleb lisaks andmetele andmebloki salvestamisel säilitada andmebaasis bloki metaandmed. Andmebaasis andmete hoidmine võib olla siiski paindlikum ja anda tulevikus võimalusi näiteks otsinguid teostada avaandmete baasilt. Lõpliku ligipääsu avaandmetele päringute pinustamise ja piiramise eest vastutab juba programmliideste pääsulüüs.

Käesoleval viisil avaandmete säilitamine ja avaldamine võimaldab teostada enne andmete säilitamist isikuandmete anonümiseerimist ja võimaldab eraldiseisvalt operatiivandmebaasist neid andmeid väljastada. Andmebaasis andmete hoidmine kasutades dokument orienteeritud andmesalvestustehnoloogiat võimaldab ka üle andmete otsinguid teostada.

4.2.12. Kaardimoodul

Kaardimoodul on mikroteenus mille eesmärk on tegeleda georuumiliste (*geospatial*) andmetega.

Kaardimoodul on olemasolev tarkvara TTA SKI, millel on oma andmebaasid loogikakiht ja kasutajaliides. Antud süsteem tuleb integreerida piisavas mahus käesolevas dokumendis kirjeldatud mikroteenuste arhitektuurida kasutades samu põhimõtteid. Lõppkasutaja kasutajaliies saab uuendada ja taasluua kasutades loodava süsteemi disainimustrit ja arendamise põhimõtteid.

4.3. Andmevahetusmoodulid

Käesolevas peatükis tutvustame kommunikatsiooni ja andmevahetusmooduleid ja põhimõtteid, samuti pakume välja komplekti andmeahetusega seotud mooduleid ja nende kirjelduse.

Mikroteenuste arhitektuuri mustri arenedes on kogunud populaarsust REST (*Representational State Transfer*) andmevahetus põhimõte, mis baseerub HTTP protokollil. Mikroteenuste laiemal levikul sai sellest andmevahetuspõhimõttest standardmuster. Mikroteenuste arendus ja uute tehnoloogiate tekkimisega koos on loodud uusi protokolle ja andmevahetuse mustreid, mis koguvad üha populaarsust näiteks AMQP protokoll (Advanced Message Queuing Protocol), STOMP (*Simple Text Orientated Messaging Protocol*), GraphQL, *Websocket.*

4.3.1. Süsteemi komponentide andmevahetuse põhimõtted

Käesolevas peatükis sõnastame olulisemad süsteemi komponentide vahelised põhimõtted järgnevalt:

- Süsteemi komponendid peaks eelistama omavaheliseks suhtluseks asünkroonseid protokolle sünkroonsete protokollide ees, näiteks tuleks kasutada AMQP protokolli;
- Süsteemi komponentide vahelises asünkroonses andmevahetuses tuleb luua standardiseeritud sõnumi formaat, mis sisaldab infot sõnumi osapoolte kohta, seansi metainfot ja vahetatav informatsiooni;
- Vahetatav info peaks olema JSON vormingus;
- Kui süsteemi komponendid kasutavad suhtlemiseks REST andmevahetust, siis toimub suhtlus läbi programmliidese pääsulüüsi;
- Süsteemi kasutajaliidesed peaks tagarakendustega suhtlema keerukamate andmevajaduste puhul eelistatult GraphQL seejärel REST andmevahetuse mustri abil, lihtsamate andmevajduste puhul piisab REST protokollist;
- Süsteemi kasutajaliidese ja tagarakenduste asünkroonseks suhtluseks sobiks kõige paremini STOMP või Websocket tehnoloogiad;
- REST teenused peaks olema lihttüüpidega (äriloogiliste olemitena), kusjuures tuleks rangelt välitda komineeritud lihttüüpide kohta pöörduspunktide tegemist, mis muudab programmliidese keerukaks ja raskesti hoomatavaks.

4.3.2. Süsteemi väliste osadepoolte andmevahetuse põhimõtted

Käesolevas peatükis sõnastame olulisemad süsteemi komponentide ja väliste süsteemide vahelise andmevahetuse põhimõtted järgnevalt:

- Riiklikud süsteemid suhtlevad omavahel kasutades X-tee taristut;
- Käesoleval ajal on eelistatud kasutada X-tee taristul asünkroonset REST andmevahetust, kus vahetatakse andmeid JSON formaadis;
- Kõik seni SAOP protokollile arendatud X-tee teenused mida organisatsioon pakub viiakse arendusprotsessi kõigus üle REST lahendusele, selle kohta tuleb teisi osapooli teavitada piisavalt varakult, et need saad muudatusega adapteeruda;
- Süsteemi elukaare jooksul ei looda enam uusi SOAP teenuseid;
- SOAP teenuseid teenindatakse nende üleminekuni kasutades olemasolevaid tarkvarasid;
- Loodaval süsteemil on põhimõtteliselt valmisolek võtta kasutusele ka asünkroone X-tee andmevahetuspraktika, kuid selle arendamine ja välja töötamine X-tee platvormil alles käib.
- Süsteemi osapoolte, kes ei kasuta X-tee päringuid ja süsteemi vahel on võimalik kokku leppida ja avada ka programmliideseid, mingis konkreetses äriprotsessis osalemiseks.

4.3.3. Programmliideste pääsulüüs

Kogu süsteemi üks keskne kommunikatsiooni moodul on programmliidese pääsulüüs (*Application Gateway*), mille põhiliseks ülesandeks on vahendada andmevahetuspäringuid süsteemi komponentide vahel (v.a. asünkroonsed AMQP päringud). Detailsemad ülesanded ja võimalused on loetletud järgnevalt:

- Päringute vahendamine õigele ressursile (mikroteenusele);
- Päringute autoriseerimine – õiguse tuvastamine teha päringut;
- Päringute transformeerimine – andmete manipuleerimine päringus/vastuses vastavalt vajaduele;
- Päringute paigutamine pinusse – kiiremaks vastamiseks (*caching*);
- Protokolli transformeerimine – nt REST-GraphQL;
- Päringute filtreerimine;
- Päringute monitoorimine;

Nimekiri pole lõplik.

Programmliidese pääsulüüsi oluline funktsionaalsus on päringute autoriseerimine ja see funktsionaalsus peaks olema just programmliidese pääsulüüsis, mitte igas mikroteenuses eraldi. Mikroteenuses peaks olema päringute autoriseerimine ainult erandkorras.

Programmliidese pääsulüüsiks tuleks kasutada vabavaralist valmistarkvara KrakenD[27]. Kõnealusel tarkvaral on tugi mitmesuguste teiste kommunikatsiooni lahendustega koos töötamiseks sh. tugi autoriseerida päringuid vastu identiteedi ja pääsuhalduse moodulit Keycloak. Kõik kommutaatori (*router*) funktsioonid konfigureeritakse seadete failidega ja neid saab hoida ja versioneerida GIT repositooriumis. Rakendusele on võimalik kirjutada vajadusel laiendusi vastavalt organisatsiooni vajadustele, mis muudab tarkvara paindlikumaks.

Alternatiivina nimetatud valmistarkvarale on võimalik pääsulüüs ka arendada kasutades Java Spring Boot aredusraamistikku. Näitena võib kasutada TEHIK TeIS projektis arendatud tarkvara *common-api-gateway*[28].

4.3.4. X-tee moodul

X-tee mooduli ülesandeks on vahendad X-tee päringuid sisemiste komponentide ja välimiste süsteemide vahel. X-tee poolt vaadatuna on moodul, kui X-tee adapter, süsteemi seest vaadatuna on tegemist X-tee suhtlemist abstraheeriva pääsulüüsiga.

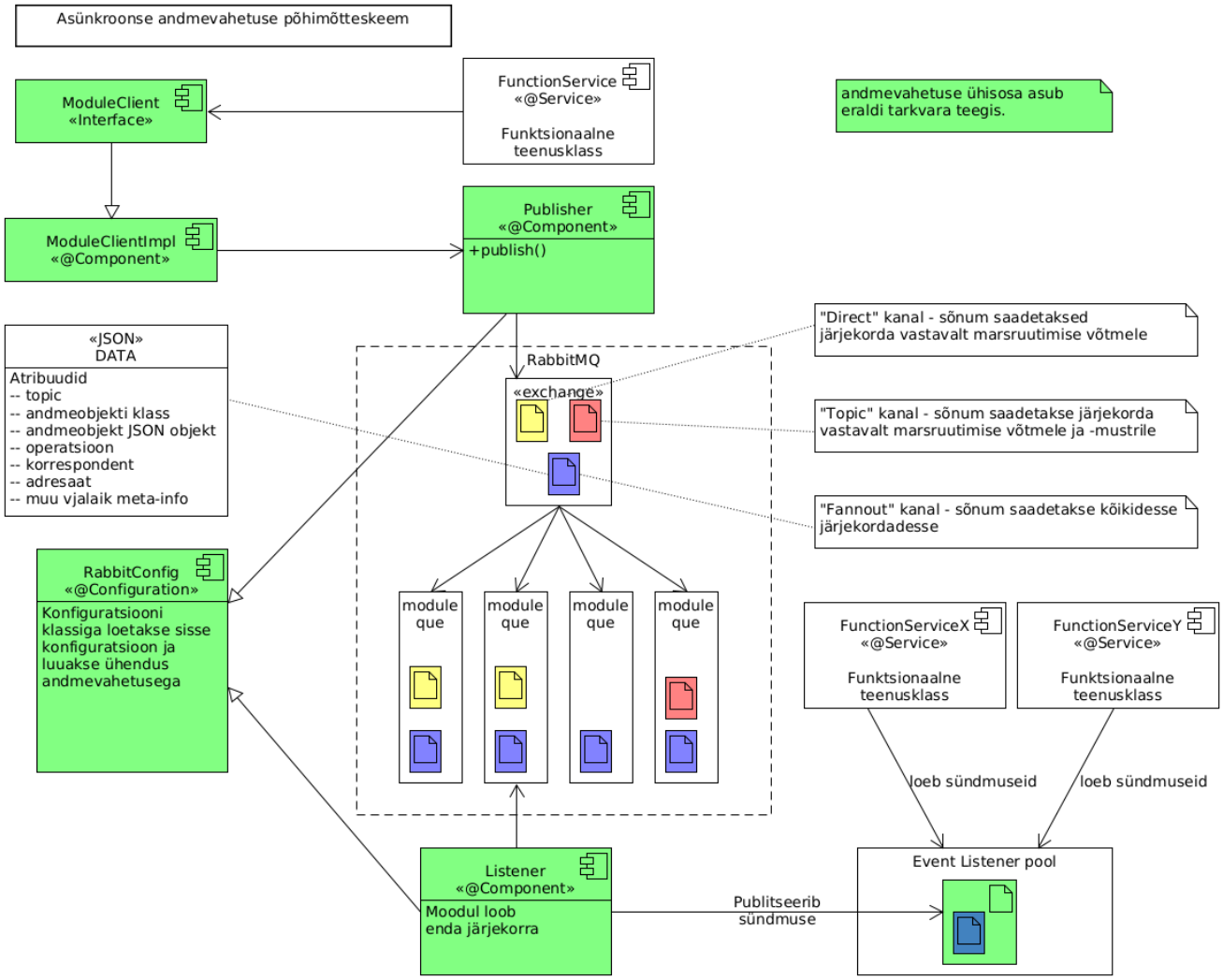
X-tee moodul sisaldab äriloogikat, mis on vajalik konkreetsete päringute teenindamiseks või informatsiooni kogumiseks. Näiteks eksisteerib hetkel selliseid x-tee teenuseid, kus andmete saamiseks on vaja kasutada kahte erinevat päringut. Ühte selleks, et andmete saamist algatada, teist selleks, et andmed alla laadida. Teisalt on selliseid päringuid, kus päringu esmakordsel tegemisel tagastatakse mingi identifikaator ja algatatakse info saamine ja sama päringu kordamisel koos identifikaatoriga on võimalik andmete tekkimisel andmed kätte saada. Sellisel juhul tuleb päringute korduv tegemine ja andmete ootamine realiseerida x-tee mooduli äriloogika protsessina. Antud tegevust on teoreetiliselt võimalik realiseerida ka protsessimootori kaasamisega, kuna antud tegevuste jada on küllaltki lihtne tegevuste algoritm. Selline lähenemine looks aga lisakeerukust moodulisse endasse.

X-tee moodul peaks olema Java keeles arendatud tarkvara või TEHIK TeIS projekti arendatud *xroad-gateway*[29] kloon.

4.3.5. Asünkroonsete sõnumite vahetuskiht

Asünkroonseks andmevahetuseks tuleb eelistada AMQP andmevahetusprotokolli, mille vahendamiseks tuleb süsteemi paigaldada RabbitMQ klaster. Klastri saab paigaldada Kubernetes platvormile.

Pakume välja ka asünkroonse andmevahetuse implementatsiooni, mis on kujutatud joonisel 8.



Joonis 8. Asünkroonse sõnumivahetuse arhitektuur *Spring Boot* raamistiku näitel

Süsteemis on keskne RabbitMQ vahendaja (*broker*). Temasse luuakse andmevahetuskanal (*exchange*) kuhu iga liitunud osapoole kohta luuakse sõnumijärjekord (*que)*. Süsteemi üleselt lepitakse kokku standardses andmevahetussõnumis, mis sisustatakse andmetega ja edastatakse andmevahetuskanalisse. Andmevahetussõnumisse pannakse kaasa ka saatja ja adressaadid. Moodulid mis on liitunud andmevahetuskanaliga võtavad sõnumi vastu, kui nad on adressaadid. Süsteemi siseselt toimub siis juba info edastamine õigele teenuse implementatsioonile. Java Spring Boot raamistikus on võimalik rakenduse sees saata sündmuseid (*event*) ja teenusklassid saavad vastavalt sündmuse tüübile (näiteks adressaadi poolt välja kutsutud operatsioon) andmed vastu võtta. Kogu sõnumite saatmise ja vastuvõtmise tarkvaraline implementatsioon peaks asuma keskses teegis, mida saab sõltuvusena tarkvaraprojektidesse lisades ja koheselt kasutusele võtta.

4.4. Andmemoodulid

Käesolevas alapeatükis käsitleme süsteemi komponente, mida saab klassifitseerida andmete põhiselt või on seotud andmete hoidmisega ja teistele mikroteenustele kättesaadavaks tegemisega.

4.4.1. Andmebaasimootor

Süsteemi põhiliseks andmebaasi mootoriks on PostgreSQL. Andmebaasi mootori evitus on kirjeldatud täpsemalt peatükis 5.

4.4.2. Põhiandmete pääsulüüs

Antud dokumendis toodud arhitektuur, ei sätesta lõpplikku mikroteenuste arvu seetõttu, et ei ole kuidagi reguleeritud mikroteenuste skoop, mis sisaldavad äriprotsesse. Kui sõnastada mikroteenus, kui eraldiseisev paigaldatav üksus, siis võib igat mikroteenust eksisteerida ka n-1 paigaldatud instantsi (skaleeritud rohkem kui üks instants). Äriprotsesse võib grupeerida mikroteenusteks liigi, või muude tunnuste järgi. Mikroteenuste arhitektuuri üks põhimõte on, et igal mikroteenusel on oma andmebaas või pole seda üldse. Teised mikroteenused pöörduvad mikroteenuse andmetele ainult läbi mikroteenuse programmliidese.

Kuna aga erinevad äriprotsessid vajavad ja toodavad põhiandmeid sõltumatult, tekib probleem kuidas andmeid hoida selliselt, et andmed ei oleks laiali üle äriprotsessi mikroteenuste andmebaaside ja andmete ligipääs oleks lihtne.

Selleks on mõistlik luua eraldi mikroteenus, mis tegeleb põhiandmete hoidmise ja käsitlemisega. See mikroteenus ei sisalda äriloogikat ja vahendab ainult andmete salvestamist andmebaasimootorisse ja erinevaid otsinguid andmetest. Selline lahendus sobib eelkõige põhiandmetele, mis ei ole transaktsiooniliste omadustega või raamatupidamislikud. Eelkõige sobib selline lähenemine staatiliste andmete sh. dokument tüüpi andmekomplektide vahendamiseks ja hoiustamiseks.

Näitena tooks olukorra, kus on andmekomplekt „tegevusteade”. See on põhiandmete komplekt, milline on eraldiseisvalt põhiandmete komplekt, mis sisaldab olulist informatsiooni kogu äriprotsessi jaoks. Sarnase olemusega eksisteerib aga mitmeid andmekomplekte, mis samas on aga natukene erinevad. On olemas mikroteenused „kasutajaliides”, „menetlus-protsess”, „teavitus” ja „andmete lüüs”. Menetlus-protsess koostab kasutajaliidese päringu peale uue „tegevusteade” andmekomplekti, eel-täidab teatud andmeväljad ja edastab kasutajaliidesesse. Kasutajaliides täiendab andmekomplekti ja saadab tagasi menetlus-protsess mikroteenusele. Menetlus-protsess mikroteenus teab, mis tegevusi edasi peab tegema, kuid üks samm on, et tuleb salvestada tegevusteate andmed. Selleks saadab „tegevusteade” andmekomplekti andmete pääsulüüsi, mis paneb andmed andmebaasi. Lisaks pöördub menetlus-protsess ka veel „teavitus”, poole, et välja saata info uuest tegevusteatest. Sellega on protsess lõppenud. Protsess võib olla lõppenud, kuid kasutajaliides soovib uuesti saada kätte „tegevusteade” andmeid, selleks pöördub kasutajaliides otse andmete lüüsi poole ja saab sealt õige info kätte.

4.4.3. Pinumälu (*cache*)

Pinumälu eesmärk on ajutiselt salvestada andmeid kiireks pöördumiseks ja jagamiseks. Pinumälu peamiseks omaduseks on, see et andmeid hoitakse mälus mitte kettamassiividel, ning andmete saamine pinust toimub võrreldes andmete saamisega andmebaasist kiiresti. Pinumälu omaduseks on ka asjaolu, et selle poole saab pöörduda mitmesuguseid kliente kes kõik vajavad sama andmestikku.

Pinumäluks tuleb kasutada Redis[30] pinumälu süsteemi.

4.4.4. Andmete indekseerimine

Andmeid on vaja indekseerida eelkõige selleks, et kiirendada nendest osa leidmist. Praktikas näiteks täisteks otsingud, sõnaosa otsingud, täppisotsingud. Andmete indekseerimine ja indeksilt tehtavad otsingute funktsionaalsused eksisteerivad juba olemasolevas süsteemis ja see tuleb üle viia uude platvormi. Kuna olemasolevas lahenduses on kasutusel Elasticsearch[31] tarkvara, tuleb seda jätkuvalt kasutada ka uuel arhitektuuril. Indeksandmete hoidlaks tuleb kasutada Elasticsearch tarkvara. Elasticsearch pakub andmete sisestamiseks ja pärimiseks REST teenust ja erinevates programmeerimise raamistikes on üldjuhul Elasticsearchi tugi valmis kujul olemas. Nagu näiteks Java Spring Boot raamistiku puhul.

Sarnaselt Relatsioonilisele andmebaasile tuleks indeksandmebaas evitada virtuaalmasinale, mitte Kubernetes klastris.

4.4.5. Pseudonüümimise moodul

Pseudonüümimise moodul on mikroteenus, mille ülesanne on korraldada süsteemis andmete anonümiseerimist ja pseudonümiseerimist. Esineb olukord, kus andmed pärast andesubjekti surma või ärilise aktuaalsuse kadu tuleb pseudonümiseerida. See tähendab, et andmesubjekti isikuandmed asendatakse initsiaalidega või pseudo-informatsiooniga.

Käesolevas moodulis hoitakse pseudonümiseerimise ärireeglid ja moodul etteantud ajal käivitamisel läheb ise ette antud andmebaasi ja käivitab seal pseudonümiseerimise algoritmi, mille tagajärjel andmed anonümiseeritakse või pseudonümiseeritakse.

Moodul peaks olema Java keeles arendatud tarkvara. Tarkvara käivitatakse koordindeeritult vastavalt vajadusele ja see ei pea töötama kogu aeg.

4.4.6. Andmete sünkronisaator moodul

Olukorras, kus eksisteerib lisaks uuel platvormil arendatud süsteemile ka vana ja teatud funktsionaalsus on arendatud uuel, samas osa vanal peab andmeid süsteemide vahel sünkroniseerima. Sellisel juhul tuleks arendada sünkroniseerimise moodul, mis võtab andmeid uuest andmebaasist (uuel andmemudelil), teisendab andmed ja sisestab need vaba andmebaasi (vanal andmemudelil). Sünkroniseerimise sammuks võib olla ka andmete transformatsioon. Näiteks viiakse andmed üle JSON vormingust EVA mudelile ja vastupidi.

Andmete sünkronisaator moodul peaks olema Java keeles arendatud tarkvara.

4.5. Äriprotsessi moodulid

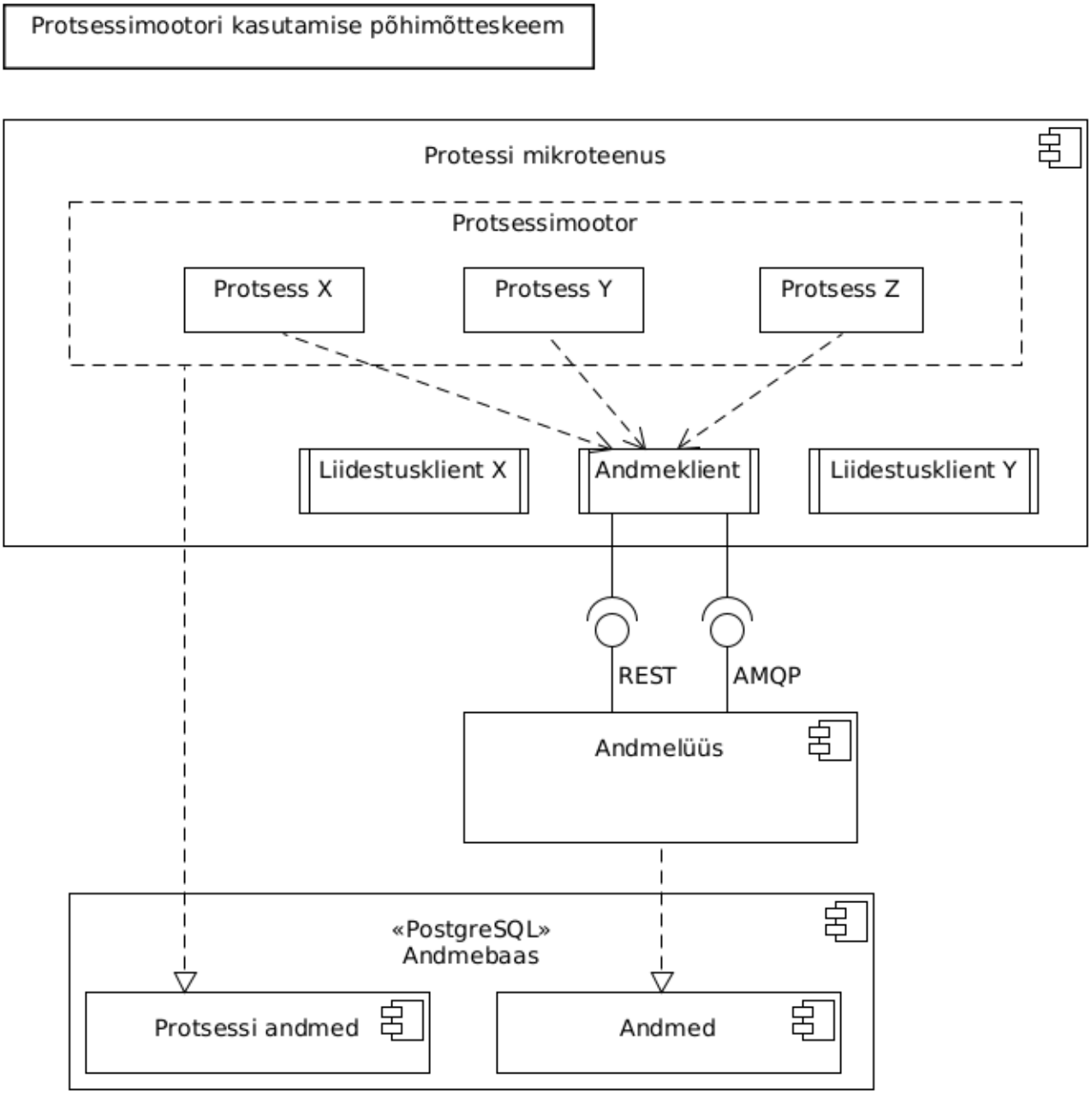
Käesolevas peatükis kirjeldame täpsemalt võimalikke äriprotsessi mooduleid, protsessimootorite kasutamise põhimõtteid ja äriprotsesside programmeerimist juhul, kui neid ei implementeerita protsessimootori abil.

4.5.1. Protsessimootori kasutamise põhimõtted

Protsessimootori kasutamise olulisemad põhimõtted on alljärgnevad:

- Protsessimootorit käitav mikroteenus peab sobituma käesolevas dokumendis kirjeldatud mikroteenuste arhitektuuriga;
- Protsessimootorit käitav rakendus oleks ühtlasi ise mikroteenus;
- Protsessimootorit võib rakendada mitmes rakenduses;
- Protsessimootori poolt käitavat protsessi peab olema võimalik laadida andmebaasi ja rakendus peab saama seda andmebaasist laadida;
- Äriprotsessi modelleerimiseks peaks olema võimalik kasutada ette antud funktsioone, mis on arendatud töötama koostöös protsessimootoriga;

Joonisel 9 on kujutatud, mil viisil võiks protsessimootorit käesolevas dokumendis kirjeldatud mikroteenuste arhitektuuril arendatud mooduli kasutada.



Joonis 9 Protsessimootori mikroteenuse moodulis kasutamise põhimõtteskeem

Üks levinud protsessimootori lahendus on Camunda[32], mida saaks integreerida käesolevas dokumendis kirjeldatud mikroteenuste arhitektuuril arendatud äriprotsessi moodulitega või keskse protsessimootori mooduliga. Camunda kasutamine sellisel juhul oleks manustatud (*embedded*) lahendusena.

4.5.2. Võimalikud eraldiseisvad ärimoodulid

Järgnevalt loetleme võimalikud äriprotsessi moodulid, mis tulenevad olemasolevate süsteemide äriprotsessidest ja andmestikust. Need on alljärgnevad:

- Aadresside haldus - moodul mis võimaldab pidada arvestust aadresside üle ja liidestuda Maameti ADS süsteemiga. Antud moodulit võiks ka kaaluda liita administratiivse vms mooduliga;
- Dokumentide haldus (DHS) - klassikalise dokumendihalduse moodul;
- Haridus ja tunnistused - moodul, kus hallatakse äriprotsessides registrites töödeldava haridusega seotud andmeid;
- Koolitusload - moodul, kus hallatakse koolituslubade andmeid ja vastavaid äriprotsesse;
- Keelud - Äriliste keeldudega seotud äriprotsesside ja andmete halduse moodulid;
- Statistika - Moodul mis võimaldab koguda ja andmelattu edastada statistilisi andmeid, põhifookus statitsilistel mudelitel peaks asuma andmeaidas;
- Kasutajate tagasiside - moodul kus hallatakse avalike kasutajate tagasiside andmeid;
- Taotlused - moodul mis töötleb erinevaid taotluseid ja vastutab andmete säilitamise eest;
- Teated - moodul mis töötleb erinevaid tegevusteadeid ja vastutab andmete säilitamise eest;
- Load - moodul mis töötleb erinevaid lubasid ja vastutab andmete säilitamise eest;
- Tegevuskohad - moodul mis töötleb erinevaid tegevusteadeid ja vastutab andmete säilitamise eest;
- Äriprotsessi logi - moodul kuhu koondatakse äriprotsessi logi;
- Numbriliikuvus - numbriliikuvuse moodul;
- Numbrilubade haldus - numbrilubade halduse moodul;
- Tarbijavaidluste haldus - tarbijavaidluse andmete ja protsesside haldamsie moodul;
- Raadiosagedused ja side - raadiosageduste ja side alaste protsesside moodul;
- Energiaaudit - erergiaauditite protsesside ja andmte haldamise moodul;
- Ettekirjutused ja järelevalve - ettekirjutuste ja järelevalve protsesside ja andmete moodul;
- Kemikaalid - kemikaalidega seotud äriprotsesside ;
- Sündmused - sündmuste keskse halduse ja äriprotsesside moodul;
- Aruandlus - Aruandluse andmete ja äriprotsesside haldamise moodul;
- Väärteomenetlus väärteamenetluste läbiviimise ja selelga soetud andmete moodul;

Nimekiri pole lõplik ega ammendav. Lähtuvalt olemasolevate süsteemide andmemudelitele, on võimalik ja tuleks tungivalt kaaluda sarnaste äriprotsesside andmete ja funktsionaalsuste konsolideerimist nii äriarhitektuuris, kui andmearhitektuuris.

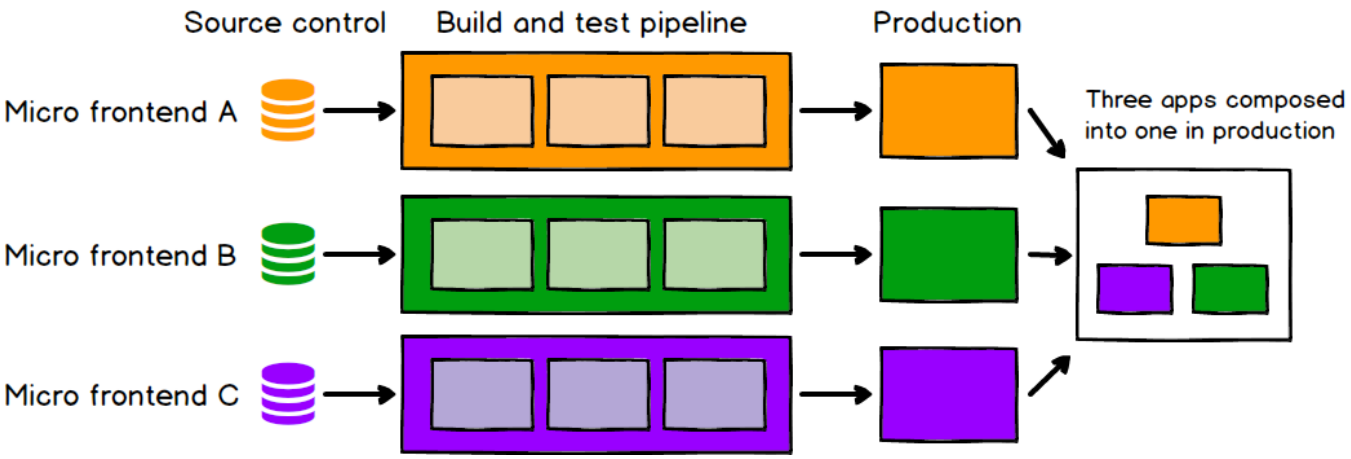
4.6. Kasutajaliidese moodulid

Käeolevas peatükis käsitleme kasutajaliidestega seonduvat. Käsitleme kasutajaliideste temaatikat mikroteenuste arhitektuuris, mikrokasutajaliideste mõistet, ksutajaliideste disainimustreid ja vahendeid, majutusarhitektuur ja andmevahetust tagarakendustega.

4.6.1. Mikrokasutajaliidesed

Mikrokasutajaliidesed (*Micro front ends*) on tihti segadus tekitav termin, mis meie hinnangul on ka tegelikult üsna laiali valguv määratlus. Mikroteenuste arhitektuuris on rõhk sõnal teenus, siis selle mõte on olnud alati jagada monoliitset infosüsteemi funktsionaalseteks osadeks (mikroteenusteks), mingite tunnuste, funktsioonide kaupa või organisatoorsete aspektide tõttu. Selle resultaat on, et tekib tagarakendus oma isikliku andmeaasiga. Seda võib IT maailmas nimetada nähtuste eraldamiseks (*separation of concerns*). Kuid kas sama praktikat peaks ja saaks rakendada ka kasutajaliidestes? Mitte alati, vähemalt mitte samal viisil.

Joonisel 10 on kujutatud mikrokasutajaliideste jaotus mingiks kujutletavaks nähtuseks. Siin on tegelikult jaotus enamjaolt tehtud mingite äriliste tööliinide kaupa, mitte mikro-funktsionaalsuste või olemite kaupa. Lihtsalt selline lähenemine, et on igal mikroteenuse tagarakendusel on ka oma mikroteenusest kasutajaliides ei toimi. Üldjuhul on vaja kasutajaliideses näidata andmeid ja töödelda neid kombinatsioonis mitme mikroteenuse andmestikuga. nt Isik ja tema Konto vms.



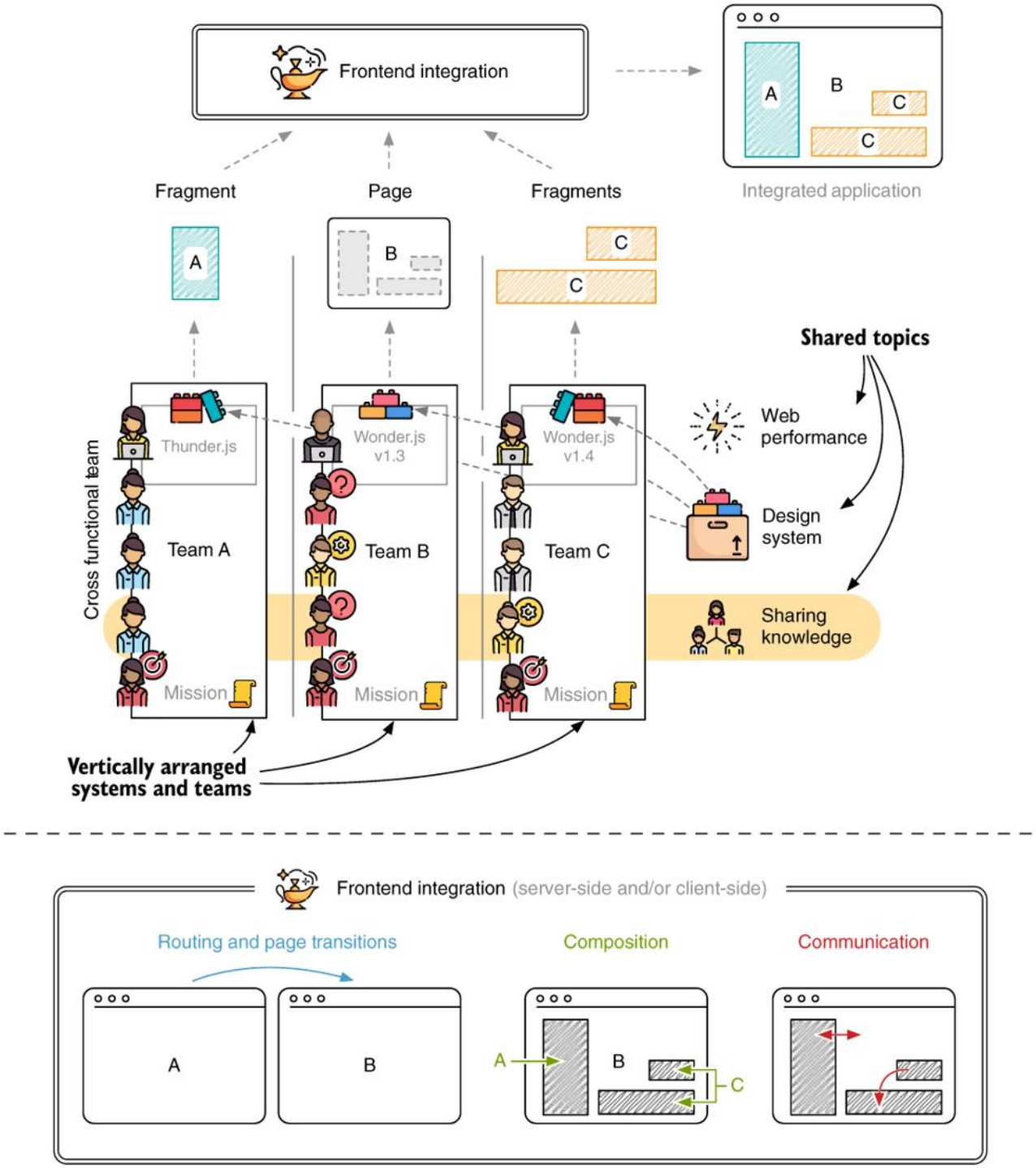
Joonis 10. Nähtuste eraldamine mikrokasutajaliideses.[33]

Oluline on aga silmas pidada, et kasutajaliideses ei pruugi toimida nähtusteks eraldamine samadel alustel, kui tagarakendustes ja andmebaasides. Kasutajaliideses tuleb tagada lõppkasutajale mugav ja arusaadav lahendus, mis aga tähendab, et mingite objektide või funktsioonide kaupa ei puurgi saada rakendust jagada. Seega liigub kasutajaliideste arendus üldse mikroteenustest eraldi ja kasutajaliidese arendamisel ei pruugi mikroteenuste olemasolu või jaotus üldse oluline olla, sest kontaktpunkt on üldse programmliides. Selles punktis on võimalik andmeid mitmetest mikroteenustest agregeerida, et vähendada liiklust kasutajaliidese ja mikroteenuste vahel.

Üks levinud arusaam, mis mikrokasutajaliidestega on kaasneb on, et mikrokasutajaliides on eraldi paigaldatav. Kuid pigem on oluline omadus efektiivne arendusvõimekus (organisatoorne). Omadus, et igat mikrokasutajaliidest võib arendada eraldi meeskond. Praktikas on mikrokasutajaliides lõpptarbija jaoks eraldiseisev JavaScript koodifail. Iga mikrokasutajaliides on pakitud eraldiseisvasse füüsilisse JavaScript faili ja seda käivitatakse ainult vastava funktsionaalsuse käivitamiseks. See ei pea muidugi tähendama, et antud mikrokasutajaliides JavaScript fail, koos muude lisadega on pakitud eraldiseisvasse paigaldatavasse konteinerisse.

Mikrokasutajaliideste jaotusel peab pigem arvestama asjaolusid, et erinevad meeskonnad saaks efektiivselt tegutseda ja oma äriprodukte arendada segamata üksteist ja omamata jäiku seoseid. Praktikas tähendab see, et on kasutusel mitmed Git repositooriumid ja ühiskasutatavad üldkomponendid on pakitud kasutajaliidese teekidesse ja kasutatud läbi sõltuvuste lahendamise tehnoloogia NPM (*Node Package Manager*).

Vertikaalselt meeskondadeks ja tööliinide gruppideks jaotamise näide on toodud joonisel 11.

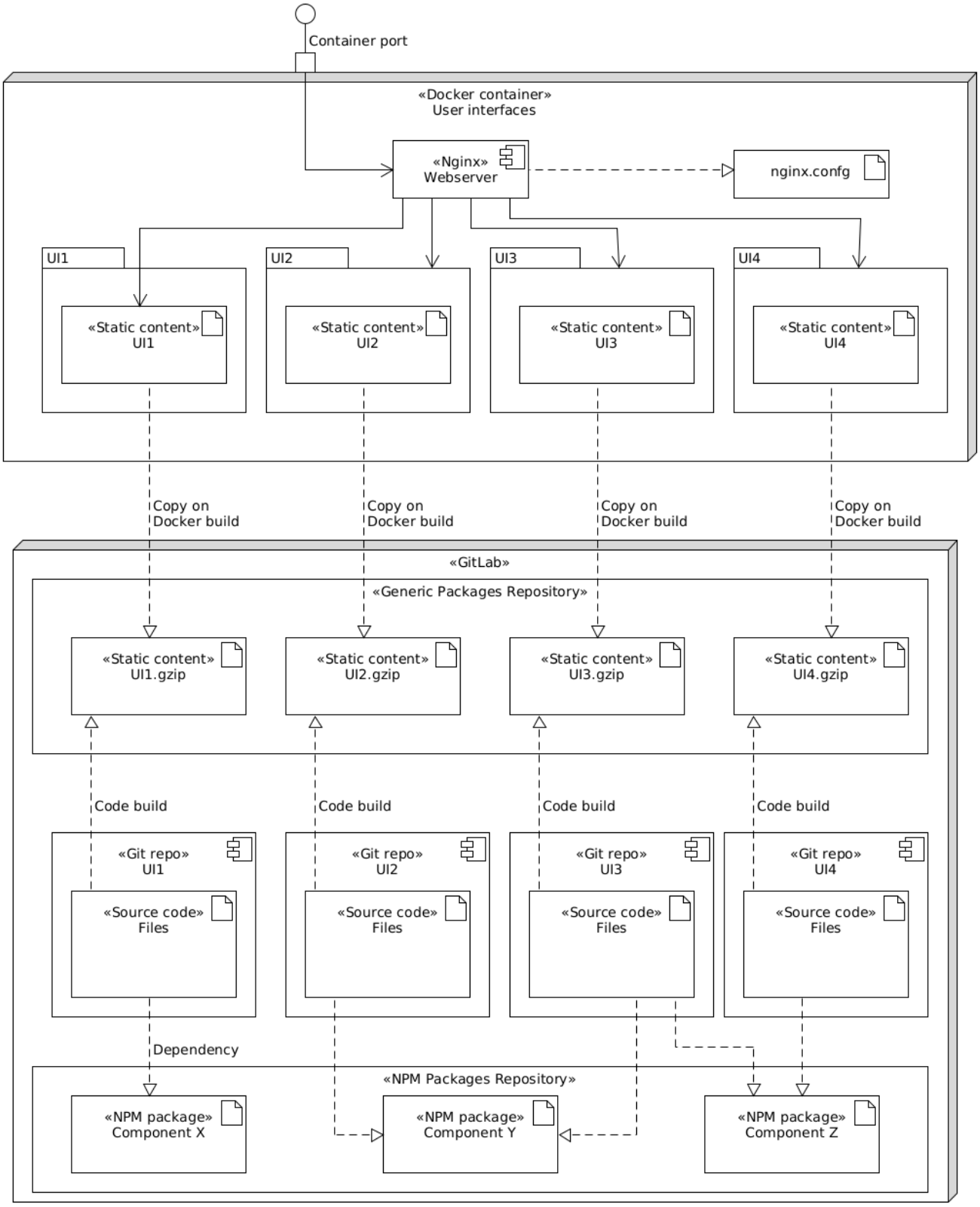


Joonis 11 Mikrokasutajaliidete vertikaalne jagunemine[34]

Kasutajaliideste arendamisel tuleks järgida organisatsioonis paika pandud disaini ja kasutajamugavuse mustreid, taaskasutada ja hallata ühtseid lihtsamaid komponente. Igal komponendil peab olema vastutaja (*code owner*).

Evituse vaatest ei pruugi mikrokasutajaliides tähendada eraldi seisvat paigaldatavat ühikut - Docker konteinerit. Liigse halduskeerukuse vältimiseks võiks mikrokasutajaliidesed siiski paketeerida üheks, kuni kaheks paigaldatavaks komponendiks. Paketeerimise aluseks võib olla ka asjaolu, et teatud funktsionaalsus on kasutatav ainult organisatsiooni sisevõrgus ja sellele ressursile ligipääs tuleb piirata võrgu tasemel.

Joonisel 12 pakume välja mikrokasutajaliideste põhiprintsiibid, kus oleme toonud ühele pildile nii, mikrokasutajaliideste jaotuse lähtekoodi hoidlas, kui ka kasutajaliideste evituse Docker konteinerina.



Joonis 12 Mikrokasutajaliideste põhimõtteline jaotus arenduses ja evitus majutusplatvormil.

Põhimõtted on järgnevad:

- Üldkasutatavad komponendid arendatakse ühes (monorepo) või mitmes repositooriumis ja paketeeritakse NPM pakiks. Pakki hoitakse artifaktooriumis;
- Üldkasutatavatel komponentidel on üks konkreetne koodiomanik (*code owner*). Muudatustesse võivad panustada kõik osapooled;

- Erinevates Git repositooriumites hoitakse erinevate lõpp-kasutajaliideste koodi;
- Koodi ehitamisel lõpptarkvaraks (staatiliseks koodiks) kaasatakse üldkasutatavad komponendid NPM artifaktooriumist sõltuvusena;
- Teistest moodulitest dünaamiliselt või staatiliselt koodi või funktsionaalsust ei kasutata;
- Lõplik staatiline kood paketeeritakse kokkusurutud pakendisse ja hoitakse artifaktooriumis;
- Paigaldatav konteineri pilt (Docker image) koostamisel laetakse sinna sisse artifaktooriumist mikrokasutajaliideste koodipakid, pakitakse lahti ja serveeritakse veebiserveriga;
- Kuna antud lähenemisel iga erineva mikrokasutajaliidese Interneti sirvikusse laadimisel uuendatakse kogu sisu, siis võimalik sessiooni info või muu oluline meta-info mida on vaja teenusest teenusesse üle anda salvestatakse sirvikus kasutades Redux[35] hoidla võimalusi.

Oluline on silmas pidada, et antud lähenemise tulemusena ei teki väga palju kasutajaliidese konteinereid, mis hõlbustab administreerimist. Arvestades seda, et kirjeldatud visiil pakitud kasutaliideste tarnimine ja uuendamine on lihtne ja ajaliselt lühike tegevus, siis pole mõistlik jagada mikrokasutajaliideseid eraldi konteineriteks. Suure hulga konteinerite tekkimine raskendaks üldist evitust ja selle administreerimist.

4.6.2. Kasutajaliidese disainimustrid ja -vahendid

Süsteemi kasutajaliideste disainimustriks tuleks kasutada Veera disainisüsteemi[36]. See on üks kolmest riiklikust disainisüsteemist, mille eesmärk on pakkuda kasutajale ühtest kasutajakogemust unifitseeritud stiiliraamistikus. Veerast on mitu versiooni ja paralleelseid koopiaid, nii *Angular* kui ja *React JavaScript* raamistikele.

4.6.3. Andmevahetus tagarakendustega

Andmevahetuseks tagarakendustega on kasutajaliidestel mitmeid võimalusi näiteks REST, Websocket ja GraphQL. Need kolm tehnoloogiat pole ükskõik sarnane, kuid võivad kombinatsioonis tõsta teenuse kvaliteeti ja süsteemi töökindlust.

REST teenus on enamlevinud ja väga populaarne lahendus. REST puhul tuleks igat olemit käsitleda ühe pääsupunkti grupina (vaatamine, muutmine, lisamine, kustutamine) mistõttu võib pöörduspunktide arv kasvada väga suureks ja seetõttu ka programmliideste spetsifikatsioon muutuda keeruliseks ja hoomamatuks. Kui ühe korraga on vaja küsida tagarakendustest (ühest) või mitmest korraga andmeid üheaegselt, siis REST puhul tähendab see mitme päringu paralleelset või järjestikulist käivitamist.

GraphQL on tehnoloogiana kiire, võimaldab unifitseeritud viisil teha andmemuudatusi ja päringuid, süsteemide suunas, kusjuures pole oluline milline on täpne tagarakenduste topoloogia. GraphQL tehnoloogia kasutamine võib lihtsustada programmliidest, kuid selle tehnoloogia enda kasutamine on mõnevõrra keerulisem kui REST puhul. Selle õppimine ja oskusteabe omandamine nõuab arendajatelt rohkem pingutusi kui REST puhul, kuid see pingutus toob kasu teistes kohades.

Websocket tehnoloogia kasutamist tuleb kaaluda kohas, kus kasutajaliidesele on vaja edastada infot tagarakenduse algatusel. Näiteks on selle tehnoloogia rakendusvaldkond reaalaja vestlusaknad.

4.7. Olemasolevad tarkvarad

JvIS – Järelevalve Infosüsteem. PHP Symfoni raamistikul arendatud monoliitne, PostgreSQL andmebaasiga.

MTR – Majandustegevuste Registri infosüsteem. PHP Symfoni raamistikul arendatud monoliitne, PostgreSQL andmebaasiga.

NBA/SASS – Numbribroneerimise ja numbrilubade halduse infosüsteem.

SKI (AlphaGIS) – Kaardirakendus, mida arendatakse ja täiendatakse eraldiseisva projektiga.

Andmeait - TTJA suurandmete andmeait, mida arendatakse ja täiendatakse eraldiseisva projektiga.

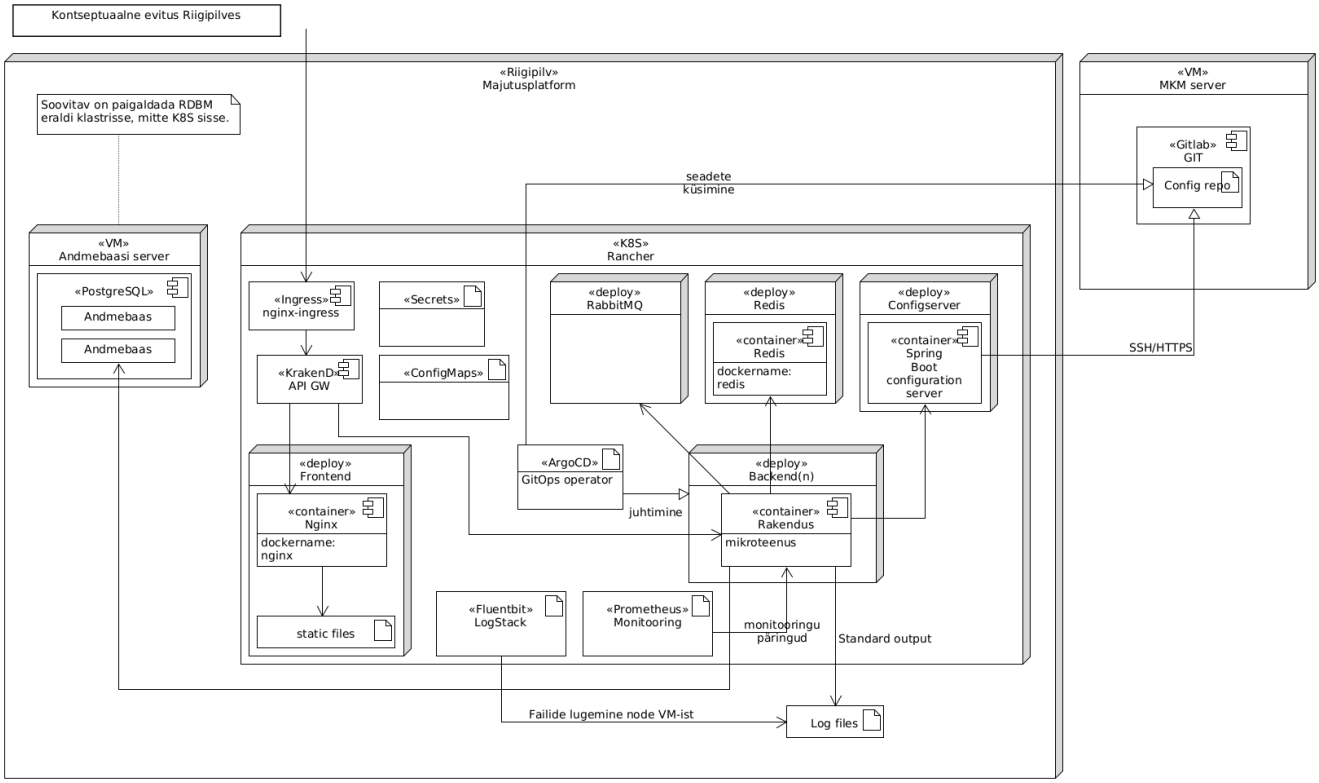
Kõesolevas dokumendis toodud mikroteenuste arhitektuuri praktilisel rakendamisel tuleks täiandada ja edasi arendada ka olemasolevaid süsteeme. Neile tuleks luua kesoelvas dokumendis nimetatud kommunikatsioonipaltvormide toed REST teenused ja AMQP suhtluse võimekus.

5. Majutusarhitektuur

Käesolevas peatükis kirjeldame süsteemi majutamist ja majutamisega seotud küsimusi, kontseptuaalset, evitust, majutusplatvormi ja -tööriistu.

5.1. Kontseptuaalne evitus

Kontseptuaalset evitust kirjeldab evitusdiagramm joonisel 13. Potentsiaalselt evitatakse süsteem virtuaalmasinatel ja Kubernetes klastris. Virtuaalmasinatel evitatakse andmebaasimootor PostgreSQL. Mikroteenused ja toetavad komponendid majutatakse Kubernetes klastris. Erinevad põhimõttelised komponendid või halduspartnerite vastutusalad eraldatakse Kubernetes nimeruumidega (N*amespace*). Rakendustele luuakse paigaldusüksused (*Deploym ent*), võrguteenused (services), konfiguratsioonid (*Configurations*). Käivitatud konteinerid (*Pod*) töötab Kubernetes õla (*Node*) serveri protsessina.



Joonis 13. Kontseptuaalne evitusdiagramm.

Evituses on kujutatud ka logimine ja monitooring, millest täpsemalt on kirjutatud peatükkides. 5.2.3. ja 5.2.4.

5.2. Majutustööriistad

Käesolevas peatükis kirjeldame detailsemalt süsteemi majutamiseks olulisi tööriistu ja tarkvarasid. Kirjeldame pilveplatvormi Kubernetes, Pideva paiggaldamise, monitooringu ja tehnilise logimise tarkvarasid.

5.2.1. Pilveplatvorm Kubernetes

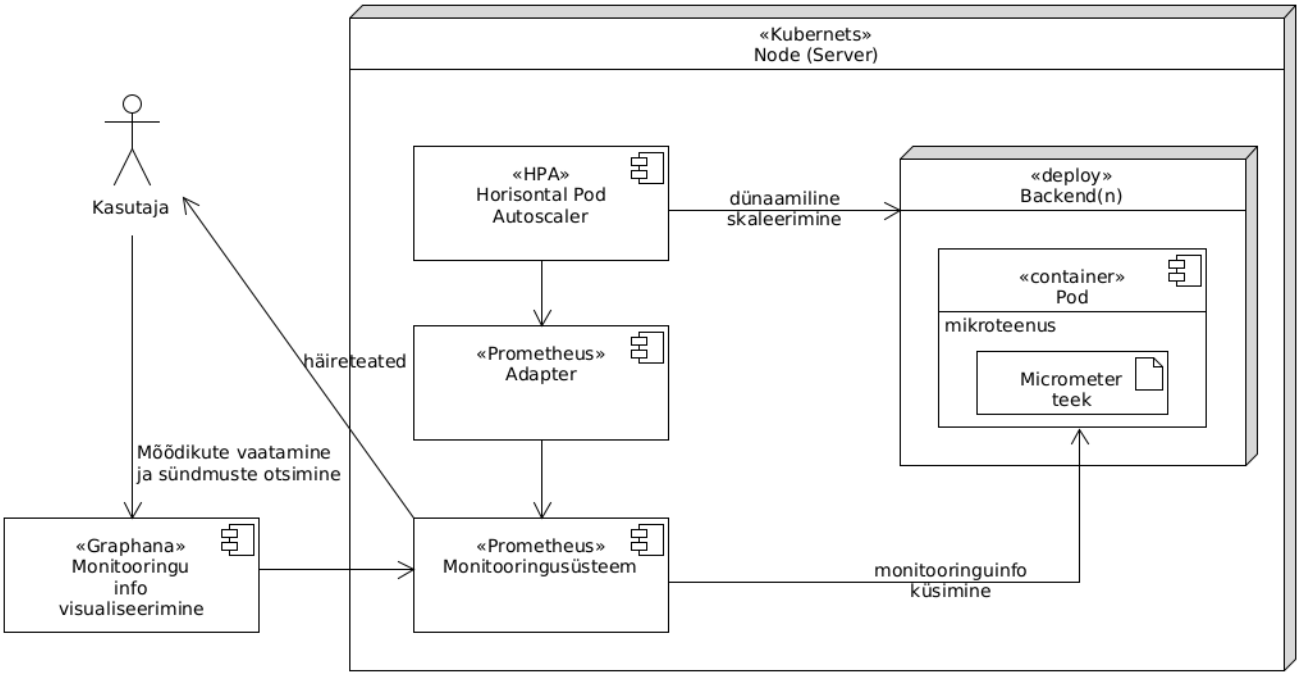
Kubernetes[37] on avatud lähtekoodiga konteineritesse pakitud rakenduste automaatse paigalduse, ressursi planeerimise, skaleerimise ja halduse süsteem. Kubernetes on muutunud *de fakto* majutuskeskkonnaks erinevas suuruses ettevõtete ja organisatsioonide poolt. Selle platvormiga on võimalik majutada mikroteenustel baseeruvat arhitektuuri ja tagada kõrgkäideldavus.

5.2.2. Pideva paigaldamise tarkvara

Pideva paigaldamise tööriistaks pakume eelkõige Argo CD[38] tarkvara. Antud tarkvara paigaldatakse Kubernetese klastrisse ja sellega hallatakse Kubernetes klastri komponente. Põhiline erisus vanamoodsa lahendusega on see, et kui varasemalt on olnud valdav, et Gitlab juhtkanalid vajavad pääsu Kubernetes klastrisse ja algatavad muudatused, siis uue lähenemise puhul CD tegevusi otseselt ei algata Gitlab, vaid Kubernetes klastris asuv Argo CD käiv ise kontrollimas Gitlabi konfiguratsiooni repositooriumis, kas on muudatusi. Vajadusel saab teha Veebikonkse (*web-hook*), et teavitada CD süsteemi toimunud muudatustest. Küll aga nimetatud lähenemine ei vaja üldse CD juhtkanalite loomist. Pideva paigaldamise tarkvaraga kasutamisega on tagatud ka GitOps põhimõtted, millest on täpsemalt kirjutatud peatükis 9.2. Samuti järgib see lähenemine Kubernetes operaatori mustrit (*Operation Pattern*)[39]

5.2.3. Monitooring

Süsteemi hetkeolukorra, alusplatvormi jõudluse ja rakenduste vaatlusparameetrite reaalajast hetkeolukorra jälgimiseks ning vajalike häiresignaalide väljastamiseks huvitatud osapooltele tuleb kasutusele võtta monitooringu moodulid-mikroteenused. Käesoleval ajal on valdkonnas üldpopulaarsed ja levinud tööriistad Micrometer[40] teek, Prometheus[41] monitooringu tsentraalne agregaator ja Grafana[42] monitooringuinfo visualiseerimise tööriist. Joonisel 14 on kirjeldatud mikroteenuste monitooringulahendus Kubernetes majutusplatvormil kasutades nimetatud tööriistu.

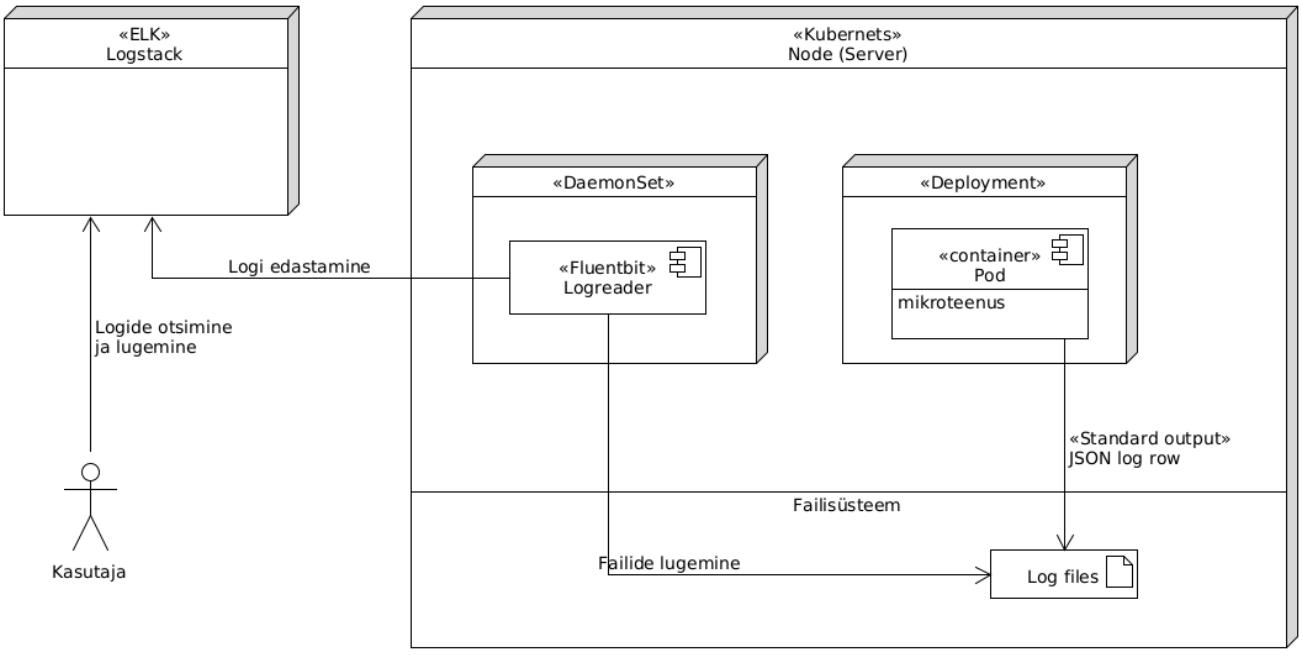


Joonis 14 Monitooringu tööriistade põhimõtteskeem Kubernetes majutuses

Suures süsteemis on oluline ka skaleerimine, selle eest peab hoolt Kubernetes horisontaalne automaatne skaleerimise funktsionaalsus (Horizontal Pod Autoscaler) edaspidi HPA. HPA seadistatakse töötama vastu Prometherus adapterit, kust saab töötava rakenduse konteineri reaalajalise info. Vastavalt seadistustele ja etteantud limiitidele skaleerib PHA vajadusel rakendust – lisab või vähendab õlgasid. Juhul kui on kasutusel klastri GitOps operaator, siis on võimalik HPA funktsionaalsust rakendada ka selle kaudu. Eelnimetatud lahendus oleks eelistatud siis, kui klastri GitOps operaatorit kasutusel poleks.

5.2.4. Tehnilised logid

Süsteemi tehniline logi on äärmiselt vajalik saamaks aru süsteemi toimimisest ja eriti, siis kui süsteemis on veaolukorrad või tõrked. Tarkvaravigade silumiseks on tarkvaraarendajal vajalik pääseda ligi logidele. Keerulistest süsteemides tuleb logisid käsitleda ühtse tervikuna, kuna päringud ja andmed läbivad mikroteenuseid ja protsesse. Kogu tervikust ilma korraliku logisüsteemita pole võimalik aru saada. Joonisel 15 on kujutatud meie poolt välja pakutud tehnilise logi kogumise ja logide kesksüsteemi edastamise lahendus.

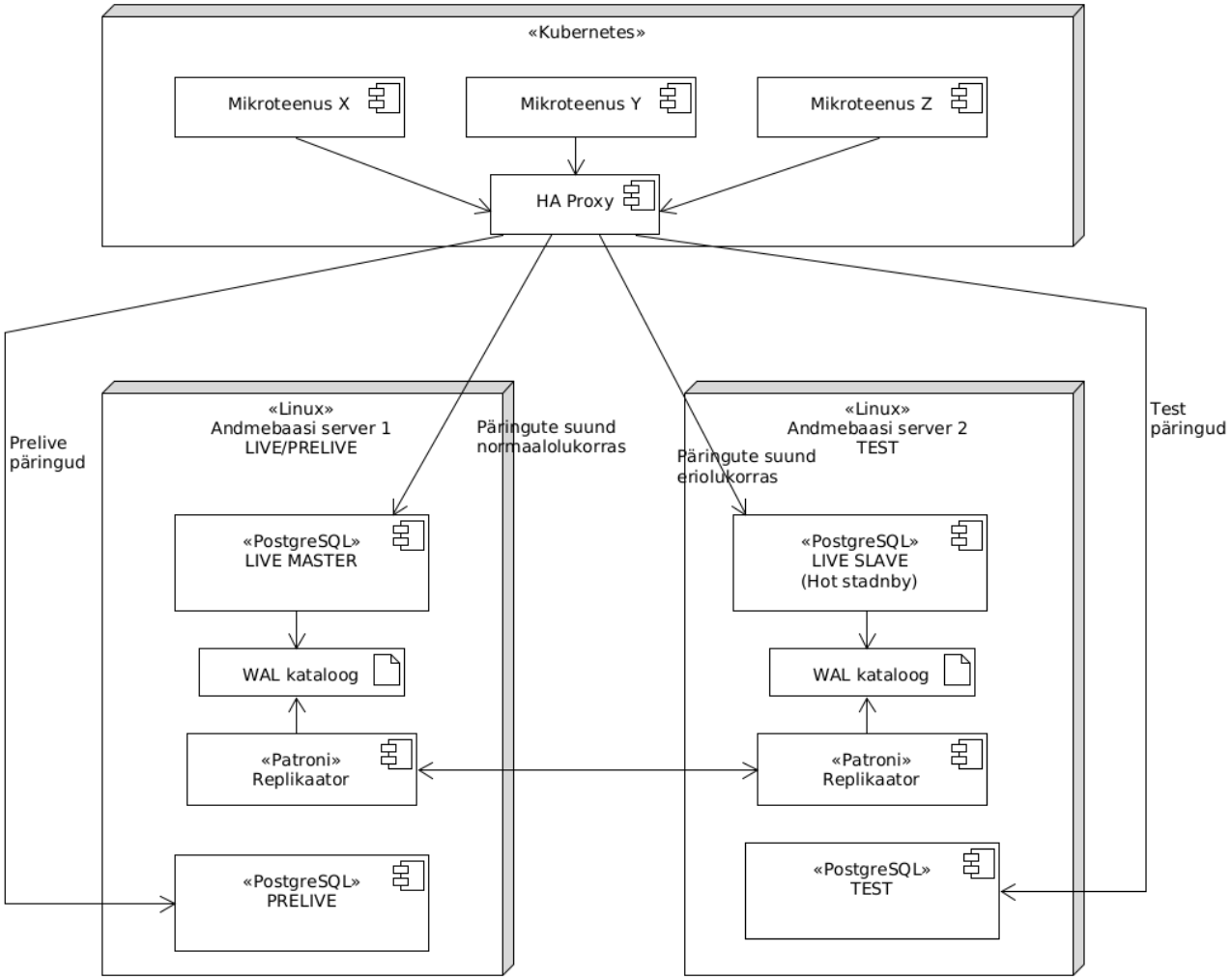


Joonis 15 Tsentraalne logimine

Kubernetes klastris on igal rakenduse õlal sõltuvalt klastri konfiguratsioonist logifail, mis asub Kubernetes serveri failisüsteemis. Logi eksisteerib seni, kuni rakenduse õlg eksisteerib. Et vältida potentsiaalset logide kaotamist (selle risk on äärmiselt kõrge), tuleb logid deponeerida automaatselt kesksüsteemi, kus on neid võimalik keskselt kasutada ja analüüsida. Logide transportimiseks tuleks paigaldada Kubernetes klastrisse Fluentbit[43] süsteemne protsess, mis kogub logid kokku serveri failisüsteemist ja edastab logid kesksesse logiserverisse.

5.3. PostgreSQL andmebaasi kõrgkäideldatav evitus

Tänapäeval on normaalsus, et süsteemid on kõrgkäideldavad. Seda nii rakenduste tasemel, kus kasutame Kubernetes klastrit, kui ka andmebaasides. Kuigivõrd me ei soovita andmebaasi mootorit majutada Kubernetes klastris soovitame andmebaasimotoorit majutada eraldiseisvas virtuaalmasinas ja kahe sellisega koostöös on võimalik saavutada kõrgkäideldavus. Joonisel 16 on kujutatud Kõrgkäideldava PostgreSQL andmebaasiklastri evitusjoonis.



Joonis 16. Kõrgkäideldava PostgreSQL klastri kontseptuaalne evitusjoonis

Kuigivõrd Kõrgkäideldava andmebaasiklastri käivitamniseks on mitmeid viise soovitame selleks kasutada voogedastuse replikatsiooni (*Streaming Replication*), kus kesksek replikaatoriks on vabavatraline tarkvara Patroni[63].

Jooniselt lähtub, et tuleks kasutusele võtta kahe individuaalse virtuaalmasinaga süsteem, mis võivad asuda erinevates võrkudes ja või füüsilistes asukohtades. Üks n.ö. toodangu server ja teine test server. Sellisel viisil ei pea kulutama ressurssi toodangu süsteemile kahe virtuaalmasina paigaldamiseks. Küll aga tuleks selline lähenemine läbi arutada infoturbe vaatest. Lubada toodangu päringud test võrku on infoturbeline otsus. Tavaolukorras suunab Kubernetes klastrisse paigaldatud HA Proxy (ainult andmebaasimootorile) päringud toodangu suunas, kui toodangu pole enam kätte saadav lülitab proksi päringud automaatselt ümber kuumas valimiduses skeundaarse andmebaasiserveri suunas.

Serveritele paigaldatakse vabavaraline Patroni tarkvara, mis korraldab ja haldab replikeerimist läbi WAL kataloogi (PostgreSQL spetsiifiline funktsionaalsus).

5.4. Infrastrutkuuri esialgne indikatiivne vajadus

Käesolevas alampeatükis toome välja esialgse indikatiivse riistvara vajaduse kogu süsteemi tarbeks.

Kubernetes klaster

Vähemalt kolm õlga, e. kolm serverit, mille igaühe parameetrid on alljärgnevad:

- Kõvakettapinda: 20GB
- Mälu: 16GB
- protsessori lõime: 8

Andmebaasi klaster

Vähemalt kaks õlga, e. kaks virtuaalmasinat, mille igaühe parameetrid on alljärgnevad:

- Kõvakettapinda: 50GB
- Mälu: 8GB
- protsessori lõime: 16

Elasticsearch virtuaalmasin

- Kõvakettapinda: 100GB
- Mälu: 16GB
- protsessori lõime: 4

Pilw.io blokkandmehoidla

Prognoositav maht kohe pärast migreerimist: 1TB

- JvIS - 800GB
- MTR - 100GB

Käeolev infikatsioon ei sisalda logide käitlemist. Logide käitlemiseks tuleks teha eraldi indikatsioon, kuid selle süsteemi kõvakettamaht on prognoositav sadades gigabaitides kuni terabaitides.

6. Andmeturve põhimõtted ja ülevaade

Käeolevas peatükis käsitleme andmeturbe põhimõtteid ja kontseptsioone.

6.1. Andmeturbe põhimõtted

Süsteemi arendamisel ja haldusel tuleb järgida Eesti infoturbestandardeid Infosüsteemide turvameetmete süsteem ISKE ja Eesti Infoturbestandardit[44]. Nimetatud dokumentidest viimane on uuem ja kaasaegsem. Süsteemi arenduse ja testimise käigus tuleks süsteemi kontrollida vastu OWASP Application Security Verification Standard[45], mida on võimalik ka automatiseerida ja kaasata süsteemi jätkuva integratsiooni protsessi. MKM-is kehtivad mitmed infoturbe küsimusi kitsamalt käsistlevad poliitikad, mis on loetletud järgnevalt:

- Krüptokontseptsioon
- Logimisekontseptsioon
- Teenuste turvaline seadistamine
- Võrgukontseptsioon

Nimetatud poliitikad on täpsemalt kirjeldatud MKM-i Confluence keskkonnas.

6.2. Andmekogude tsoneerimine

ISKE võimaldab andmekogusid tsoneerida. Selleks tuleks füüsiliselt eraldada osad mikroteenused või andmebaasid teistest. Kõrgema turvaosaklassiga andmed koondatakse ühte füüsiliselt eraldatud „tsooni". Kubernetes majutuskeskkonnas on võimalik eraldada rakendusi eraldi tsoonidesse kasutades nimeruumide loogikat. Andmebaasides on võimalik tsoone luua andmebaaside või andmebaasi skeemide kaupa.

7. Tarkvara arendamise ja testimise metoodikad ja praktikad

Käesolevas peatükis kirjeldame tarkvara arendamise ja testimist mikroteenuste arhitektuuril arendatud tarkvaraprojektide erinevates faasides.

7.1. Tarkvara arendamise ja projektijuhtimise metoodika

Tarkvaraarenduse protsessis on mitmeid metoodikaid ja lähenemisi. Meie soovitame mikroteenustel baseeruva süsteemi arendamiseks ja haldamiseks kasutada SCRUM[46] metoodikat. SCRUM on kergekaaluline, kuid sellegi poolest võimas kooslus väärtustest, põhimõtetest ja praktikatest. Scrum toetub multi-funktsionaalsetele meeskondadele, et tarnida tooteid ja teenuseid lühikeste iteratsioonidena võimaldades alljärgnevaid asjaolusid:

- Kiire tagasiside;
- Kiirem innovatsioon;
- Pidev parendamine;
- Kiire kohanemine muutustega;

- Rohkem rahulolevaid kliente;
- Kiirendatud tempo ideest tarnimiseni.

Scrum pakub meeskonnale tööriistad, praktikad, rutiinid, tseremooniad, mis aitavad paremini protsessi mõtestada ja oma tööd organiseerida.

Ühtlasi võib *Scrum* metoodikat nimetada agiilseks tarkvaraarenduse metoodikaks.

7.2. Testimise metoodikad ja praktikad

Käesolevas alapeatükis käsitleme testimise temaatikat, metoodikaid ja praktikaid. Käesolevas dokumendis käsitletud mikroteenuste arhitektuuril ehitatud süsteemile tuleb rakendada kõiki vajalikke testimise metoodikaid ja testimise tüüpe, et tagada süsteemi kvaliteet ja vastavus nõuetele. Muidugi peab silmas pidama, et testimine oleks võimalikult automaatne ja kasutusel oleks tööriistad, mis hõlbustavad testide loomist ja efektiivset testimist. Süsteemi testimine on olnud palju aastaid arendajate manuaalne tegevus, millest pole jäänud jälge ja mida pole võimlaik korrata ega automatiseerida. Loodavale süsteemis tuleb kasutada testimise raamistikke ja tööriistu, mis võimaldaks teste lihtsalt kirjeldada, arendada ja korduvalt mis tahes aja hetkel käivitada.

7.2.1. Funktsionaalne testimine

Funktsionaalne[47] testimine hõlmab rakenduste testimist ärinõuete suhtes. See kaasab mitmesuguseid testi tüüpe, et tagada, et rakenduse iga osa toimib nii nagu ette nähtud. Selleks kasutatakse kasutuslugusid, millised on koostanud meeskonna analüütikud ja kvaliteedi insenerid. Funktsionaalse testimise testi tüüpideks on näiteks:

- Ühiktestimine (Unit test)
- Integratsioontestimine (Integration test)
- Süsteemi testimine (System test, End to End test)
- Vastuvõtu testimine (Acceptanse test)

Ühiktestimine on esmatasandi testimise liik, mida korraldab ja arendab tarkvara arendaja vahetult tarkvara lähtekoodi kirjutades. Testidega kaetakse väikesed koodi ühikud ja sellega veendutakse, et meetodid või funktsioonid toimivad nii nagu peavad. Ühiktestid kirjutatakse kohe samal ajal, kui põhitarkvara ja veendutakse teste käivitades, et seni arendatud tarkvara toimib vastavalt tarkvara alamosa toimimise nõuetele. Tarkvaraarenduse käigus ei tuleks üldse tarkvara ennast käivitada, vaid käivitada ainult teste, mis kontrollivad tarkvara osade tööd. Testimise andmeid loob arendaja üldjuhul ise või kasutab analüütikute või kvaliteedieksperdi poolt loodud testandmeid.

Integratsioontestimine on testimise liik mille abil testitakse juba süsteemi moodulite ja üksikute funktsioonide koostoimimist. Integratsioontestimise arendamise käigus jälgitakse samu põhimõtteid, kui ühiktestimise käigus järgiti. Selle testimise käigus võivad osaleda ja olla hõlmatud ka teised süsteemid või osapooled, mida tuleb testides, siis simuleerida. Integratsioontestid on üldjuhul nagu ka ühiktestid koodi lahutamatu osa, seega nad asuvad koodiga samas repositooriumis. Paim viis selleks on käivitada testide käivitamise ajaks ka kõik teised integratsiooni osapooled ja kasutades testandmeid vastata päringutele või täita käskluseid. Selle testimise liigi testandmed peaks olema loodud meeskonna üleselt kvaliteedieksperdi poolt.

Süsteemi testimine on testimise liik, kus testid rakendatakse süsteemi kui „musta kasti" suhtes. Nimetaud testid hõlmavad endas nii programmliidese teste kui ka kasutajaliidese teste. Üldjuhul simuleeritakse lõppkasutaja tegevusi ja testid käivitatakse päris eeltestimise keskkonnas kasutades toodangukeskkonnale ligilähedasi andmeid. Testandmestik peaks olema meeskonna üleselt hallatud ja loodud kvaliteedieksperdi poolt. Testide koostamise ja käitamise eest peaks hea seisma samuti kvaliteediekspert.

Vastuvõtutestimine on viimase funktsionaalse testimise faas ja seda kasutatakse, et kontrollida tarkvara kui terviku vastavaust ärinõuetele ja on valmis tarnimiseks toodangukeskkonda. Selle testimise käigus veendutakse, et tarkvara vastaks lõppkasutaja vajadustele. Seda testimist korraldavad ja teste arendab üldjuhul meeskonna kvaliteediekspert (automaatsed testid) ja manuaalses lõpptestimises osaleb ka lõppkasutaja esindaja. Testimine viiakse üldjuhul läbi vastuvõtutestimise või eeltoodangu keskkonnas, kasutades selleks toodangukeskkonnaga ligilähedasi andmeid.

Soovitame testimise korraldamiseks kasutada järgnevaid tööriistu ja vahendeid:

- **JUnit 5**[48] – Java ühiktestimine ja integratsioontestimine
- **Testcontainers**[49] – Integratsioontestimine, süsteemi testimine
- **Cypress**[50] - Süsteemi testimine, E2E testimine, kasutajaliidese testimine, vastuvõtutestimine
- **WireMock**[51] – Integratsioontestimine, Süsteemi testimine
- **Postman**[52] – tagarakenduste REST pöörduspunktide testimine

7.2.2. Mittefunktsionaalne testimine

Mittefunktsionaalne testimine peaks hõlmama tarkvara operatiivseid aspekte. Sellisteks testideks on näiteks:

- Jõudlustestimine (Performance testing)
- Turvatestimine (Security testing)
- Kasutustestimine (Usability testing)
- Ühilduvuse testimine (Compatibility testing)

Jõudlustestimise eesmärgiks on kontrollida, kuidas rakendus käitub erinevates tingimustes ja selle käigus püütakse simuleerida reaalolukorda või koormatakse süsteemi üle planeeritud kasutuse, et leida süsteemi murdepunkt. See testimise liik jaguneb veel omakorda koormus, stress, tipu ja vastupidavuse testideks. Jõudlusteste tehakse eelkõige eeltoodangu keskkonnas suhtes, kuid on ka teisi stsenaariumeid.

Turvatestimine peab välja selgitama eelkõige, kas süsteemis käsitletavad andmed on kaitstud, süsteem on töövõimeline ja selle funktsionaalsusi poleks võimalik ära kasutada viisil, mis pole kirjeldatud ärinõuetega. Testitakse nii kasutajaliidese kaudu, kui ka pöörduspunktide kaudu. Turvatestimine tehakse eelkõige eeltoodangu keskkonnas suhtes.

Kasutustestimise eesmärgiks on kasutusmugavuse testimine lõppkasutaja vaates. Eesmärk on tuvastada, kas disain on esteetiline ja rakendus vastaks ettenähtud kasutusvoogudele. Need testid on hea viis tiimide üleselt testida mikrokasutajaliideseid ja kogu süsteemi kui tervikut.

Ühilduvustestimine võiks antud dokumendi raames kirjeldatud süsteemi osas rakendada kasutajaliidestele, et välja selgitada, kas sama kasutajaliides toimiks erinevates operatsioonisüsteemides ja Interneti sirvikute toodetes. Tagarakenduste osas sellist testimise liiki otseselt pole vajadust kohandada.

Soovitame testimise korraldamiseks kasutada järgnevaid tööriistu ja vahendeid:

- **Sonarqube**[53] – koodi staatiline analüüs, turvatestimine, koodi objektiivne kvaliteet
- **Apache JMeter**[54] - jõudlustestimine

8. Koodi- ja konfiguratsiooni haldus, tehiste ehitamine, tarne

Käesolevas peatükis kajastame Tarkvara lähtekoodi, rakenduste ehitamiste ja automaatsete süsteemide haldamist.

8.1. Lähtekoodi haldus, tarne ja CI/CD

Käesolevas peatükis kirjeldame detailsemalt lähtekoodi haldust, lähtekooid majutuse, tarneprotsessi, pideva integreerimise ja pideva paigaldamise temaatikat.

8.1.1. Lähtekoodi majutamine

Lähtekoodi hallatakse Git repositooriumis. Konkreetselt MKM Gitlab tarkvaraga, mis on majutatud eraldiseisvalt lõpprakendustest.

Sõltuvalt projektist tuleks kaaluda, kas monorepo, polürepo või hübriidrepo kasutamist. Monorepo on Git repositoorium, kus kõigi moodulite, teekide, mikroteenuste lähtekood on samas GIT repositooriumis. Polürepo on Git repositoorium, kus teegi, mikroteenuse või mooduli lähtekood on paigutatud eraldi repositooriumisse.

Polü- ja hübriidrepo eelduseks on, et eksisteerib tehiste repositoorium (*artifactory*), mis näiteks sisaldab kompileeritud tarkvarateeke (Java, JavaScript jne.) Repositooriumi puudumisel on nimetatud repositooriumi stiilide puhul vajalik kõik sõltuvused alati ehitamiseks alla laadida ja alati uuesti ehitada (kompileerida ja paketeerida), mis tähendab oluliselt pikemat protsessi aega.

Monorepo puhul on kõik tarkvara komponendid ühes repositooriumis. Mitmete arenduspartnerite vahel arenduste jagamisel on selline lähenemine äärmiselt ebaefektiivne tekitades arendusprotsessid administratiivset keerukust ja vajadust täpsemaks koordineerimiseks. Käeolevas dokumendis toodud mikroteenuste arhitektuuri puhul, on selline lähenemine pigem välistatud, kuna on planeeritud kaasata mitmeid arenduspartnereid ja toimuvad mitmed sama aegsed arendused.

Tulevaste arenduste puhul on võimlaik kasutada kas Gitlabi enda tehise artifaktooriumit või MKM-i artifaktooriumit Jfrog.

8.1.2. Lähtekoodi tarne haldus

Lähtekoodi tarnimiseks tuleks kasutada mõnda levinud ja hästi juurdunud lähtekoodi tarneahela protsessi (edaspidi tarneprotsess). Ühes selliseks on *Goitflow*[55]. Selles tarneprotsessis on kirjeldatud, kuidas kasutada Git-i funktsionaalsuseid, harusid (branch), lipikuid (tag) jne. Selle tarneprotsessi põhiliseks omaduseks on järgnevad asjaolud:

- „*Main/master*” harus ei toimu arendust;
- Jooksvad aredused asuvad harus nimetusega „*develop*”;
- Funktsionaalsuste arendamine toimub „*feature*” harudes, kus aluseks on võetud „*develop*”;
- Vastuvõtutestimine toimub haru „release” peal;
- Arendusharud mestitakse „*develop*” harusse tagasi pärast arenduse lõpetamist ja lokaalset testimist;
- „*Develop*” haru mestitakse regulaarselt „*release*” harusse, mille baasil tehakse testimist ja vastuvõtutestimist;
- Tarne paigaldamise järgselt mestitakse „*release*” haru „*master*” harusse;
- Kriitiliste ja kiireloomuliste arenduste sh. turvavigade parandamise aluseks võetakse „*master*” haru ja muudatused mestitakse „*release*” ja „*develop* ” harusse;

Arenduse ja tarneprotsessi osa on koodi mestimine, mis peaks toimuma kasutades Gitlabi mestimise juhtloogika (*merge requests*).

Koodi kvaliteedi tagamiseks vastavalt mestitavale koodi harule tuleb läbi viia koodi ülevaatus (code review). Samuti tuleks rakendada tarnitavale koodile staatilisi koodianalüüsi tööriistu. MKM-is on olemas *SonarQube*[56] staatilise koodi analüüsi tööriist, mida saab kasutada.

8.1.3. Lähtekoodi ehitamine ja sõltuvuste haldus

Tarkvaraprojektides on oluliseks pakettide ja sõltuvuste haldus, samuti koodi ehituse korraldamine. Eelkõige Java programmeerimiskeele maailmas, kuid mitte ainult on enamlevinud Ant, Maven ning kolmanda põlvkonna tööriistaks võiks lugeda Gradle. Soovitamegi kasutada süsteemi ehitamiseks tarkvaraprojektides Gradle[57] tööriista ja tarkvaraprojektide paketeerimise, sõltuvuste halduse ja ehitamise tarbeks. Gradle tööriista saab kasutada nii, koodi genereerimise, kompileerimise, pakkide ehitamise, NPM, Node vms. Docker konteinerite loomise, käivitamise jne. tarbeks. Kui kõikides tarkvaraprojektides on üks tööriist kasutusel, siis on kogu süsteemi arendus kontrolli all ja unifitseeritud. Gradle kasutamisel kõigis tarkvara ehitamisega seotud tegevustes saab ehituskeskkonnas lihtsustada töövoogude seadistamist ja ehitusagentide unifitseerimist. Sama agendi tüübiga saab ehitada kõik tarkvarad hoolimata, kas see on Java, Node.js, vms. Tagarakendus või kasutajaliides.

8.1.4. Pidev integreerimine ja pidev tarne (CI/CD)

DevOps üheks aluspõhimõtteks on CI/CD (*Continuous Integration* / *Continuous Deployment*) kasutamine praktikas. Tuleb vältida manuaalseid tegevusi, mis vähendavad meeskonna läbilaskevõimet või paneb koormuse mõnele meeskonnaliikmele.

Pideva integratsiooni (CI) läbi viimiseks tuleb kasutada vastava funktsionaalsusega tarkvara. Gitlabi, mis on MKM-i ametlik Git süsteem on selline funktsionaalsus arendatud. Selleks on võimalik arendada ja konfigureerida Git repositooriumite juurde integratsiooni juhtkanalid (*work flows*), millised teevad ette määratud tegevusi automaatselt erinevate Git sündmuste toimumisel.

Näiteks saab käivitada automaatselt koodi ehitamist (kompileerimist ja paketeerimist) iseseisvaks teegiks ja deponeerimiseks tehiste varamusse e. artifaktooriumisse (*artifactory*) või Docker registrisse. On võimalus välja kutsuda ka CD tegevusi, mis jõustavad muudatused vajalikus keskkonnas.

Pideva paigaldamise (CD) läbiviimiseks on harilikult kasutatud juhtkanaleid (*workflow*). Kus rakendatakse mingis majutuskeskkonnas muudatused automaatselt. Süsteemide ja halduse keerukuse kasvamisel on mõistlik rakendada tööriistu, mis on evolutsioonis arenenud just lahendama pideva paigaldamise probleeme. Samuti võimaldab see tõsta üldist turvalisust ja detsentraliseerida protsesse. Uue põlvkonna CD põhimõtete ja tööriistade kohta kirjutame peatükis 9.

8.2. Konfiguratsioonide haldus

Käesolevas peatükis käsitletakse nii süsteemi kui ka äriliste seadete ja parameetrite haldust.

8.2.1. Rakenduste ja keskkondade seaded

Rakenduste süsteemsed ja majutuskeskkondasid kirjeldavad seadeid ja parameetreid hoitakse GIT-i repositooriumis. See toimimise viis on kooskõlas DevOps ja GitOps põhimõtetega, millest on täpsemalt kirjutatud peatükis . Seadete jagamiseks on käesolevas dokumendis kirjeldatud tehnilise konfiguratsiooni moodul peatükis 4.2.7. Äärmiselt oluline on silmas pidida, et nimetatud seadete haldus toimuks väljaspool Kubernetes majutuskeskkonda, kuna on teoreetiline oht, et neid võib seal muuta, need võivad seal kaduda ja Kubernetesis seadete hoidmise korral pole tagatud seadete versioneerimine.

8.2.2. Äriprotsessi seaded

Äriprotsessi seadeid on mõistlik hallata samuti nagu rakenduste seadeid. GIT-i repositooriumis ja jagada läbi REST teenuse. Mikroteenuste arhitektuuris on kirjeldatud tehnilise konfiguratsiooni mikroteenus peatükis 4.2.7. Samal viisil oleks võimalik äriprotsessi seadeid hallata ja kättesaadavaks teha teistele mikroteenustele. Sellisel viisil pole vaja luua andmebaasi, vastavaid protsessi mootoreid, et mingeid protsesside ja rakenduste käigus vajalikke ärilisi parameetreid hallata. Selle meetodi rakendamiseks on vaja aga muudatuste sisseviimine korraldada kahel võimalikul viisil:

a) GIT repositooriumis seadete failide muutmiseks luuakse mikroteenus ja triviaalne kasutajaliides. Seadefaili sisu kuvatakse kasutajaliideses ja muutujate muutmisel luuakse uus tekstifail. Tagarakendus kirjutab faili sisu GIT-is üle. Selline lähenemine vajab täiendavat tarkvaraarendust, kuid pole niivõrd prioriteetne. Eeliseks on asjaolu, et seadete muutja ei pea omaga otseselt GIT-i ligipääsu, ega teadmisi GIT-i repositooriumisse muudatuste tegemiseks.

b) Tavapärane GitOps protsess, kus tehakse lokaalses (lõppkasutaja tööjaamas) muudatus, viiakse muudetud failid GIT repositooriumisse ja läbitakse mestimise protsess. Selle meetodi eeliseks on, et on tagatud GitOps põhimõtte rakendamine ka äriliste muutujate puhul ja see ei vaja eraldi tarkvara arendust. Puuduseks on, et organisatsiooni lõppkasutaja peab omaga algteadmisi GitOps tööriistadest ja protseduuridest. Teisalt, kui ärilisi muudatusi tehakse DevOps raames, siis antud asjaolu ei ole miinuseks.

9. Süsteemi majutus- ja administreerimise põhimõtted

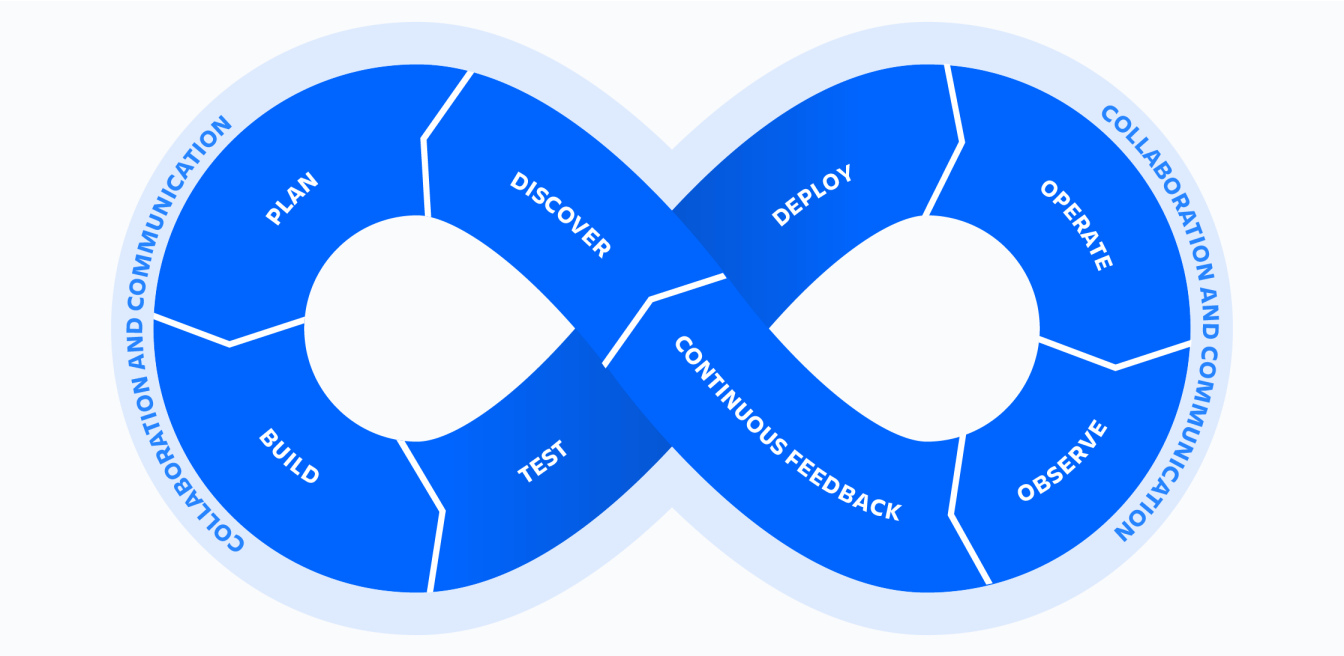
Käesolevas peatükis kirjeldame dokumendis kirjeldatud mikroteenustel arendatava arhitektuuri soovituslikku maajutuse ning administreerimise põhimõtteid. Kajastame DevOps, GitOps praktikaid ja potentsiaalset majutust Riigipilves.

9.1. DevOps

DevOps lähenemine on kogum praktikatest, tööriistadest ja IT kultuurilistest filosoofiatest, mis automatiseerivad ja integreerivad tarkvaraarenduse protsesse ja IT meeskondi. See lähenemine väärtustab tiimide jõustamist, tiimide vahelist suhtlust ja kaastööd ja tehnoloogiate automatiseerimist.[58]

Kui vanasti olid IT meeskonnad niiöelda „silod”, kus analüütikud, arendajad ja administraatorid töötasid eraldiseisvalt ja andsid tulemeid lihtsalt üle, siis DevOps lähenemisest on need rollid põhimõtteliselt integreeritud ja nii tarkvara arendusprotsessi ettevalmistus, tarkvara arendamine, kui ka toodangusse paigaldatud tarkvara haldamine on tarkvaraarendusmeeskonna igapäevaste rutiinide osa. Kuna töömaht on kõigis nendes etappides suur, siis tuleks kõikki samme võimalikult palju automatiseerida ja protsesse standardiseerida.

DevOps elutsükkel on kujutaud joonisel 17.



Joonis 17 DevOps elutsükkel [59]

DevOps-i olulisemad põhimõtted[60]:

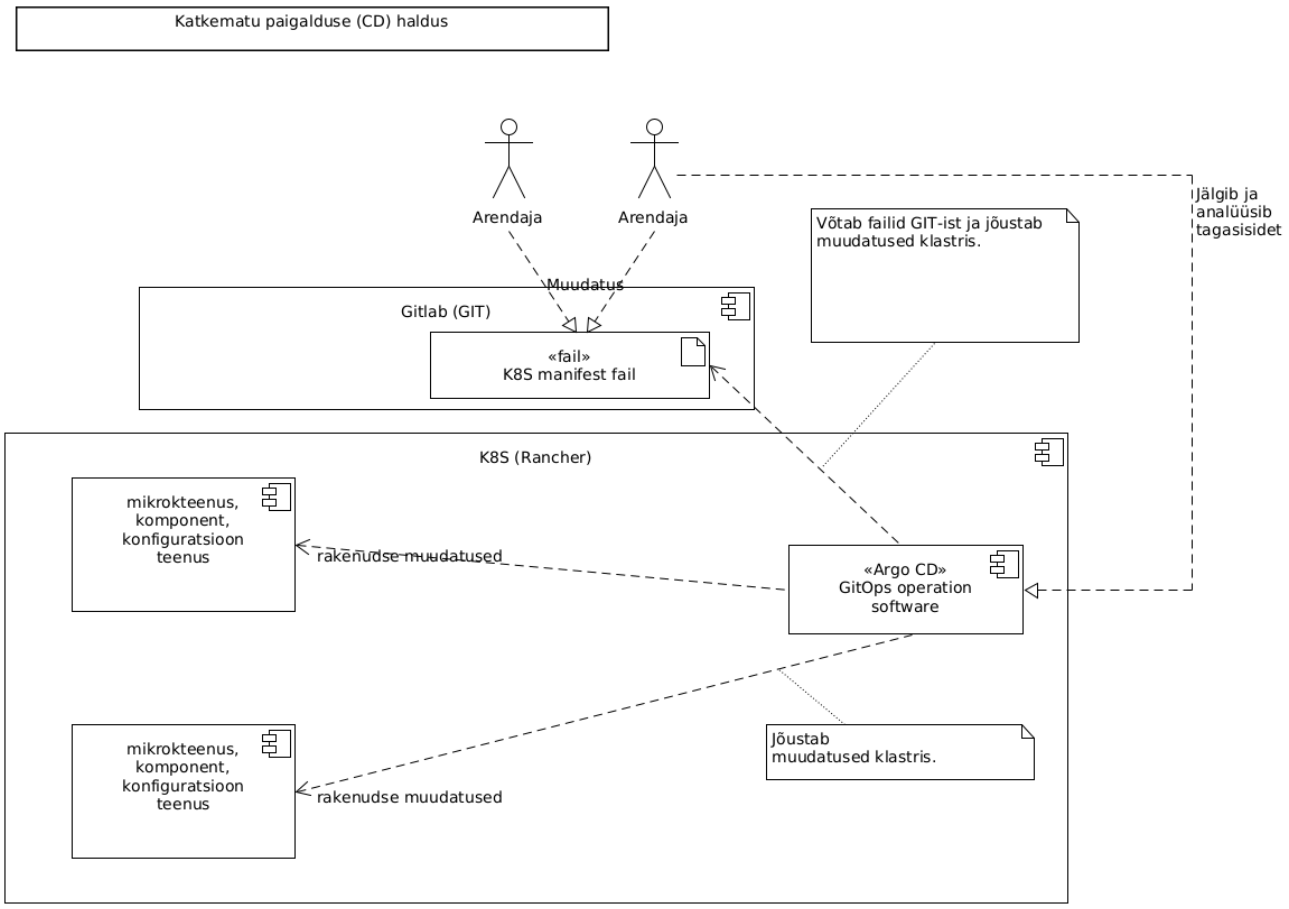
- Kõik meeskonna arendustegevuste tulemuste resultaat peaks olema kliendi põhine tegevus;
- Tarkvara on loodud lõppkasutajat silmas pidades;
- Meeskonnad peavad nõustuma, et neil on otsast-lõpuni vastutus kogu tarkvaratoote projekti jooksul.
- Edukal meeskonnal on esindatud balansseeritud oskuste komplekt, mille tulemusena on meeskond ristfunktsionaalne ja autonoomne;
- Eksperimenteerimine on äärmiselt vajalik, mille resultaat on väljakutsete võtmine ja pidev parendamine;
- IT operatsioone tuleb optimeerida ja automatiseerida kus vähegi võimalik.

9.2. GitOps

GitOps on uue põlvkonna administratiivne lähenemine. See kasutab DevOps-is tuntud tarkvara arenduse parimaid praktikaid, näiteks versioonihaldus, koostöötamine, vastavus, CI/CD ja rakendab neid infrastruktuuri, süsteemide, rakenduste administreerimisel.[61]

GitOps rakendamiseks tuleb jõustada lisaks DevOps lähenemisele asjaolu, et süsteemi kirjeldused ja konfiguratsioonid on hoiustatud GIT repositooriumites ja muudatuste haldus toimub läbi mestimise korralduse (m*erge requests*).

GitOps tööriistaks pakume Kuberentes klastris kasutada Argo CD[62], mis võimaldab rakendada GitOps lähenemist. Põhimõtteskeem on kujutatud joonisel 18.



Joonis 18. GitOps rakendamine ja Argo CD kasutamine jätkuva paigaldamise halduses

Kubernetes klastrisse on paigaldatud Argo CD operaator tarkvara, mis käib regulaarselt kontrollimas GIT seadistuste repositooriumit. Argo CD seejärel rakendab muudatused Kubernetes klastris. Argo CD kasutajaliides on ligipääsetav DevOps inseneridele ja süsteemi administraatoritele, kes saavad sellest vajalikku tagasisidet, mille abil on võimalik juhtimisotsuseid teha ja operatsioone juhtida. Käesolev lahendus tõstab ka süsteemi turvalisust, kuna pole vaja enam avada klastri administratiivset programmliidest avalikku võrku. Initsiatiiv muudatuste rakendamiseks tuleneb süsteemist seest.

9.3. Majutus Riigipilves

Süsteemi majutus peaks lähtuma põhimõttest, et kõik majutuskomponendid on välja vahetatavad ja mõistliku aja jooksul kolitavad ilma täiendavaid tarkvaraarendusi tegemata.

Erinevad teenuspakkujad võimaldavad erinevaid ärilisi probleeme lahendada omal viisil, mis tähendab, et organisatsiooni eest on ära lahendatud mingi funktsionaalsus ja tarbija saab funktsiooni kasutada sellega adapteerudes. Lühidalt on see Tarkvara nagu teenus (*SaaS – Software as a Service*). Tuleks kasutada teenuseid, mille asendamine samaväärsega on lihtne ilma tarkvaras suuri muudatusi tegemata. Teisalt pakuvad majutusplatvormid platvormi teenust (*PaaS – Platform as a Service*), kus kliendile pakutakse mingit infrastruktuuri taset nt virtuaalmasin, ja klient saab sinna oma tarkvara paigaldada ise. Käesolevas dokumendis käsitletav pilveteenus Kubernetes on Riigipilve mõttes laiendatud *PaaS*. Kuhu saab kliendi poolt tekitada nimeruume ja majutada erinevaid tarkvarasid üksteisest sõltumatult töötama.

Käesolev dokument on kajastanud arhitektuurses osas, milliseid teenuseid oleks mõistlik hallata tarkvara teenustena ja milliseid platvormi teenustena.

10. Kasutatud kirjandus

1 https://www.ttja.ee/

2 https://www.riigiteataja.ee/akt/125062021017?leiaKehtiv

3 https://www.ttja.ee/ariklient/ametist/ametist/tutvustus-ja-struktuur

4 Riigi infosüsteemide Amet INFOSÜSTEEMIDE KOLMEASTMELISE ETALONTURBE SÜSTEEMI ISKE, 2017

5 MSÜS, https://www.riigiteataja.ee/akt/106042021005?leiaKehtiv

6 https://www.europarl.europa.eu/news/et/headlines/priorities/digipoore/20210211STO97614/suurandmed-maaratlus-eelised-ja-voimalikud-probleemid-infograafikud

7 https://www.oracle.com/big-data/what-is-big-data/

8 https://www.oracle.com/big-data/what-is-big-data/

9 https://www.ria.ee/sites/default/files/content-editors/publikatsioonid/avaandmete_loomise_juhend.pdf

10 https://avaandmed.eesti.ee/

11 J. L. Harrington, Relational Database Design and Implementation, 4th Edition. Morgan Kaufmann, 2016, jt. 1, pt. 1, jt. 2, pt 3-5.

12 C. Chasseur, Y. Li, and J. M. Patel. Enabling json document stores in relational systems. 2013

13 https://json-schema.org/

14 https://www.riigipilv.ee/teenused/andmesalvestuse-teenus/pilw-io-storagevault-pilwio

15 https://datatracker.ietf.org/doc/html/rfc7519

16 https://www.keycloak.org/

17 https://jwt.io/

18 https://koodivaramu.eesti.ee/tehik/teis/persons-service

19 https://koodivaramu.eesti.ee/tehik/teis/payments-service

20 https://koodivaramu.eesti.ee/tehik/teis/signing-service

21 https://geoserver.org/about/

22 https://spring.io/projects/spring-cloud-config

23 https://cloud.spring.io/spring-cloud-config/multi/multi__spring_cloud_config_client.html

24 https://koodivaramu.eesti.ee/tehik/teis/signing-service

25 https://koodivaramu.eesti.ee/tehik/teis/messages-service

26 https://www.ria.ee/sites/default/files/content-editors/publikatsioonid/avaandmete_loomise_juhend.pdf

27 https://www.krakend.io/

28 https://koodivaramu.eesti.ee/tehik/teis/common-api-gateway

29 https://koodivaramu.eesti.ee/tehik/teis/xroad-gateway

30 https://redis.io/

31 https://www.elastic.co/

32 https://camunda.com/

33 https://martinfowler.com/articles/micro-frontends.html

34 https://livebook.manning.com/book/micro-frontends-in-action/chapter-1/16

35 https://redux.js.org/

36 https://zeroheight.com/3d136290e/p/188910-veera-disainissteem/b/94293b

37 https://kubernetes.io/

38 https://argo-cd.readthedocs.io/en/stable/

39 https://kubernetes.io/docs/concepts/extend-kubernetes/operator/

40 https://micrometer.io/

41 https://prometheus.io/

42 https://grafana.com/

43 https://fluentbit.io/

44 https://eits.ria.ee/et/versioon/2021/etalonturbe-kataloog/app-rakendused/app3-voorguteenused/app31-veebirakendused/1-kirjeldus/

45 https://owasp.org/www-project-application-security-verification-standard/

46 https://www.scrumalliance.org/about-scrum

47 https://smartbear.com/learn/automated-testing/software-testing-methodologies/

48 https://junit.org/junit5/docs/current/user-guide/

49 https://www.testcontainers.org/

50 https://www.cypress.io/

51 https://wiremock.org/

52 https://www.postman.com/

53 https://www.sonarqube.org/

54 https://jmeter.apache.org/

55 https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

56 https://www.sonarqube.org/

57 https://gradle.org/

58 https://www.atlassian.com/devops

59 https://www.atlassian.com/devops

60 https://www.getxray.app/blog/get-started-with-devops-principles-best-practices-and-tips?utm_term=&utm_campaign=GG_SEARCH_TOFU+-+DSA+Blog+Posts+-+EU2&utm_source=adwords&utm_medium=ppc&hsa_acc=9970092548&hsa_cam=17994277636&hsa_grp=136842240141&hsa_ad=615722725589&hsa_src=g&hsa_tgt=dsa-1391637226778&hsa_kw=&hsa_mt=&hsa_net=adwords&hsa_ver=3&gclid=Cj0KCQjwkOqZBhDNARIsAACsbfKWamhE2aBX0hS0Y5icg5FRLGxK-9XmngWe5O6YurKsiGWoKEGcM3caAmodEALw_wcB

61 https://about.gitlab.com/topics/gitops/

62 https://argo-cd.readthedocs.io/en/stable/

63 https://patroni.readthedocs.io/en/latest/