TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Ragnar-Kevin Kaselt 185225IAIB

Ristsõltuvuses olevate mikroteenuste ühiskasutatavate failide eraldamine ning migratsioon uude sõltumatusse komponenti

Bakalaureusetöö

Juhendaja: Jekaterina Tšukrejeva
MSc

Tallinn 2023

Autorideklaratsioon

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

Autor: Ragnar-Kevin Kaselt

15.04.2023

Annotatsioon

Antud lõputöö eesmärgiks on Siseministeeriumi infotehnoloogia- ja arenduskeskuses (edaspidi SMIT) oleva infosüsteemi ristsõltuvuses olevate mikroteenuste ühiskasutatavate failide eraldamine ning migratsioon uude sõltumatusse komponenti. Esmalt selgitatakse lõputöös kahe infosüsteemi sõltuvust ning põhjuseid miks on vaja süsteemidest ühine komponent välja tuua.

Analüüsitakse migratsiooni ning süsteemide uuendamise riske. Lisaks tuuakse välja arendusprotsess, kuidas uus projekt püsti on pandud ning lühidalt on räägitud mõnest arendustükist. Samuti on räägitud ka migratsioonist ning kuidas see lahendatud on.

Lisarõhk on pandud komponendi tehnoloogilisele uudsusele, hooldatavusele ning ka riskide vähendamisele. Lõputöös tuuakse välja kuidas on lahendatud failide viirusekontroll.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 40 leheküljel, 9 peatükki, 7 joonist, 1 tabel.

Abstract

Cross Dependent Microservices Separation And Migration Of Shared Data Into An Independent Component.

This thesis aims to separate shared files of microservices with cross-dependencies in an information system of the Information Technology and Development Center of the Ministry of the Interior (SMIT) and migrate them to a new independent component. Firstly, the thesis explains the dependencies between the two information systems and the reasons why a common component needs to be extracted from the systems.

The risks of migration and system update are analyzed. In addition, the development process of setting up the new project and some development pieces are briefly discussed.

The focus is also on the technological novelty, maintainability, and risk reduction of the component. The thesis also outlines how virus scanning of files has been implemented.

The thesis is in Estonian and contains 40 pages of text, 9 chapters, 7 figures, 1 table.

Lühendite ja mõistete sõnastik

Loose coupling	Mikroteenust peaks saama muuta nii, et teistes mikroteenustes pole vaja muudatusi seetõttu teha
Spring Boot	Java baasil olev raamistik
LTS	Long-term support, ehk pikaajaline tugi
Cron-job	Cron-job on ajastamise tööriist, mis võimaldab Unix-põhistel süsteemidel kasutajatel automatiseerida ülesandeid, käivitades käske või skripte ajakava järgi
Pull request	Viis muudatuste ettepanekute tegemiseks versioonihaldussüsteemis nagu Git
Spring Data	Raamistik Springis, mis lihtsustab erinevate andmeallikate töötlemist. See pakub tööriistu andmete päringute, salvestamise ja taastamise tegemiseks ning integreerub populaarsete andmejuurdepääsutehnoloogiatega
Spring Security	Spring Framework'i moodul turvafunktsioonide lisamiseks Java rakendustele
Spring MVC	Spring Framework'i moodul veebirakenduste loomiseks, kasutades Model-View-Controller (MVC) arhitektuurimustrit
API	API-d (Application Programming Interfaces) on definitsioonide ja protokollide kogum, mis võimaldavad kahel tarkvarakomponendil omavahel suhelda
Getter	Getter meetodit kasutatakse klassis oleva välja väärtuse teada saamiseks
Setter	Setter meetod on mõeldud klassis olevate privaatsete väljade väärtuste muutmiseks
Konstruktor	Konstruktor on koodi plokk, mis võimaldab tekitada klassidest objekte
Builder	Builder klass eraldab objekti loomise loogika originaalsest klassist

JAR	Java archive.
CI/CD	Continuous Integration & Delivery, ehk pidev integreerimine ja pidev tarnimine
Mapper	Kiht mis liigutab andmeid objektide vahel hoides neid eraldiseisvana
Front-end	Front-end veebiarendus on andmete teisendamine graafiliseks liideseks kasutades selleks HTML, CSS ja JavaScripti, et kasutajad saaksid andmetega suhelda ja neid vaadata
Back-end	Veebirakenduse osa, mis haldab serveripoolseid toiminguid ja koordineerib andmete töötlemist ning andmebaasiühendust.
Endpoint	Veebiteenuse või rakenduse API lõpp-punkt, mille kaudu saab teha päringuid andmete saamiseks või salvestamiseks
Code smell	Mis tahes omadus programmi lähtekoodis, mis võib osutada sügavamale probleemile
Yml fail	Inimloetav andmete serialiseerimiskeel. Seda kasutatakse tavaliselt konfiguratsioonifailide jaoks ja rakendustes, kus andmeid salvestatakse või edastatakse
Rollback	Riistvaratoote või tarkvaraprogrammi varasema versiooni tagastamine pärast hilisema versiooniga seotud probleemide ilmnemist

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Lõputöös keskendutakse SMIT-i ühe infosüsteemi mikroteenuste vahelisele sõltuvusele ning selle mõjule süsteemi terviklikkusele ja tulevastele arenguplaanidele. Mikroteenuste arhitektuur on muutumas üha olulisemaks organisatsioonide infosüsteemides ning seetõttu on oluline uurida ja mõista nende omavahelisi sõltuvusi ning riske.

Selle lõputöö peamisteks eesmärkideks on:

1) Analüüsida hetkel kasutusel olevaid mikrosüsteeme, tuvastada nende omavahelised sõltuvused ja riskid ning uurida mikroteenustega seotud tulevikuplaane. Selle käigus soovin paremini mõista olemasoleva süsteemi terviklikkust ja tuvastada võimalikke arenguvõimalusi.

2) Analüüsida erinevaid andmete migratsioonilahendusi ning uurida tänapäeva uuemates rakendustes kasutatavaid tehnoloogiaid. Selle käigus soovin selgitada välja parimad praktikad andmete edukaks ja tõhusaks üleviimiseks ühest süsteemist teise ning hinnata nende lahenduste sobivust uuritavas infosüsteemis.

3) Arendada uus eraldiseisev ja kergekaaluline komponent, mis vähendab sõltuvusi mikroteenuste vahel ning on tehnoloogiliselt uuenduslik ja taaskasutatav. Selle komponendi eesmärk on luua süsteem, kus uute mikroteenuste integreerimine on lihtne ning väheneb turvariskide ja jõudluskadude oht. Lisaks soovin parandada arhitektuurilist arusaama uuritavast mikrosüsteemist.

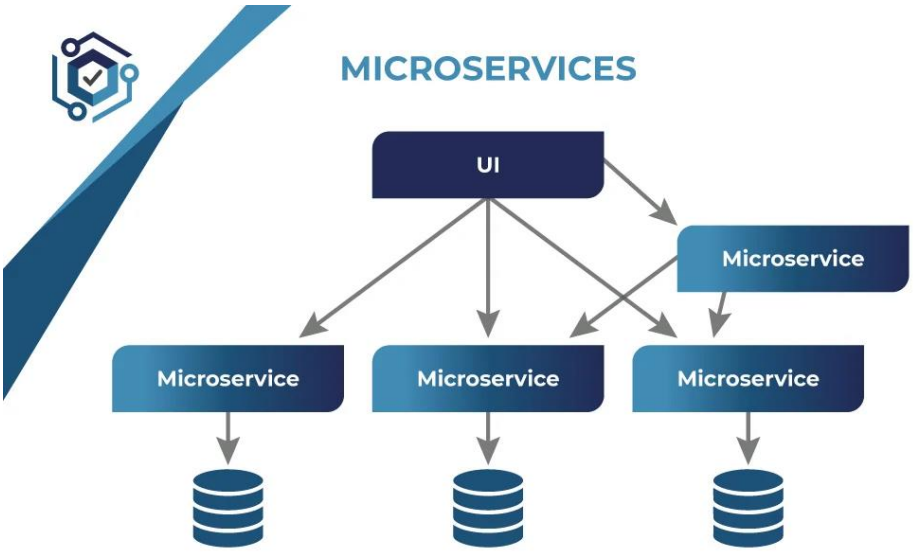
Uurimistöö keskendub olulisele teemale ning pakub väärtuslikku panust mikroteenuste arhitektuuri paremaks mõistmiseks ja arendamiseks. Loodetavasti aitab antud lõputöö kaasa infosüsteemide tõhususele ja jätkusuutlikkusele ning loob aluse edasisteks uurimusteks selles valdkonnas.

2 Probleemi kirjeldus, võimalike lahenduste analüüs ja eesmärk

SMIT, mis tegutseb Siseministeeriumi haldusalas, on asutus, mis vastutab peamiselt Politsei- ja Piirivalveameti, Päästeameti, Häirekeskuse, Sisekaitseakadeemia ja Siseministeeriumi infotehnoloogia teenuste arendamise ja haldamise eest. Lisaks oma igapäevastele arendus- ja haldustegevustele teeb SMIT koostööd nii Sisekaitseakadeemia kui ka Tallinna Tehnikaülikooliga, et toetada IKT valdkonna ülikooliharidust. SMITi vastutusaladesse kuuluvad mitmed ülesanded vastavalt põhimäärusele, sealhulgas: IKT lahenduste ja süsteemide hankimine, arendamine ja haldamine ning nende tegevuse korraldamine, riikliku elektroonilise isikutuvastusvahendi arendamise ja turvalisuse tagamine, IKT valdkonna riigihangete korraldamine ja küberturbe standardite rakendamine, sealhulgas küberturbealaste soovituste väljatöötamine ja ettepanekute tegemine nende rakendamiseks [1].

SMIT-is kasutatakse enamjaolt mikroteenuste arhitektuuri. Riiklikud it-infosüsteemid eelistavad mikroteenuste arhitektuuri kasutamist monoliitarhitektuuri kasutamise asemel. Mikroteenuste arhitektuuri järgivad infosüsteemid peaksid eraldiseisvalt toimima ja olema omavahel vähe seotud või sõltumatud (Joonis 1). Mikroteenus peaks järgima Loose couplingu ehk nõrga sidestuse kontsepti. [2] Mikroteenused ei tohiks olla omavahel sellises sõltuvuses, et kui ühes teenuses tehakse muudatus, tuleb teises teenuses samamoodi muudatus teha.

Mikroteenused on väiksema mõõtmelised, kergekaalulised ning üksteisest piisavalt eraldatud süsteemid. [3] Mikroteenuste arhitektuuris suhtleb kasutajaliides mikroteenustega, kuid võibolla ka nii et mikroteenused suhtlevad omavahel. Kui ühte mikroteenust uuendada või juhtub intsident, kus teenus pole kättesaadav, ei tohiks kogu süsteemi ulatuses teised mikroteenused suuresti kannatada. Seda muidugi täiesti välistada ei saa, eriti siis kui süsteemid omavahel suhtlevad. Lisaks sellele on ka uuendamine lihtsam, kuna teised mikroteenused saavad jätkata oma tööd, kui üks mikroteenustest taaskäivitatakse uuenduse jaoks.



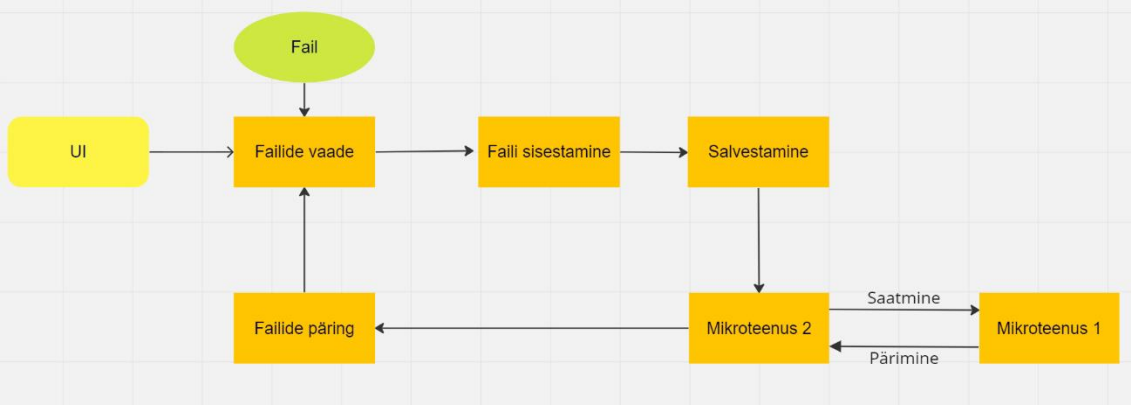
Joonis 1. Näide mikroteenuse arhitektuurist

2.1 Hetkeolukord

Infosüsteemis on kahe mikroteenuse vahel tugev sõltuvus. Edaspidi nimetan neid mikroteenuseid kui teenus 1 ja teenus 2. Teenuses 1 hoitakse mõlema süsteemi faile. Kui teenusel 2 on vaja teha päring, et faili kätte saada või salvestada on vaja teenus 1-te see saata. Hetkel toimub faili salvestus nii, et saadetakse UI-st fail enne teenusesse 2 ning pärast seda saadetakse see edasi teenusesse 1, kus toimub ka salvestus (Joonis2). Juhul kui mikroteenus, milles faile hoitakse läheb hooldusesse või on intsident, kus teenus pole kättesaadav, kannataksid mõlemad mikroteenused, kuna üks teenus ei saaks enam faile kätte ja teine mikroteenus pole kättesaadav.

Lisaks eelnevalt mainitud põhjustele on uue komponendi tegemise vajadus veel laialdasem. Tulevikus tuleb kahele mikroteenusele juurde veel üks mikroteenus, mis oleks pidanud hakkama salvestama oma faile samuti teenusesse 1, kuid äriloogika järgi oleks pidanud see toimuma läbi teenuse 2. Selline päringute jada ei ole hea tava mida järgida.

Teenus 1 on tänaseks päevaks juba küllaltki kaugel mikroteenuse mõistest. See teenus on üsna mahukaks kasvanud. Väga suur osa sellest on failide spetsiifiline loogika, toimingud kui ka salvestus. Samuti on keeruline liita uut teenust juurde, mis kasutaks faile.



Joonis 2. AS-IS Hetkeolukorra päringu näide

2.2 Probleemile võimalike lahenduste analüüsimine

On mitmeid erinevaid viise lähenemaks tugevale sõltuvusele kahe mikroteenuse vahel infosüsteemis. Üks võimalik lahendus on luua kergekomponent, mis saaks hakkama

failihaldustöödega mõlema mikroteenuse jaoks, samal ajal kui see oleks ka tulevikus teiste mikroteenuste jaoks taaskasutatav. See aitaks vähendada mikroteenuste vahelist sõltuvust ja parandada süsteemi skaleeritavust. Uus komponent võiks olla disainitud ka juhtumite või hooldusprobleemide talumiseks, et mikroteenuste üldine funktsionaalsus ei oleks mõjutatud.

Teine lähenemine on analüüsida põhjalikumalt iga mikroteenuse poolt hetkel käsitletavat failiga seotud protsessi ja otsustada, kas need protsessid võiks konsolideerida uude komponenti. See vähendaks mitte ainult mikroteenuste vahelist sõltuvust, vaid loob ka tõhusama ja taaskasutatava lahenduse, mida saaks tulevikus kasutada ka teiste mikroteenuste puhul. [4]

Sõnumijärjekorra süsteemi kasutuselevõtt aitab ka mikroteenuste sõltuvust vähendada ja muudab need tõrkekindlamaks. Sõnumijärjekorra abil saavad mikroteenused saata sõnumeid üksteisele asünkroonselt, mis tähendab, et nad ei pea ootama vastust enne teiste päringute töötlemist. See aitab parandada süsteemi skaleeritavust ja usaldusväärsust. [5]

Veel üks võimalus on kasutada sündmustepõhist arhitektuuri, kus mikroteenused suhtlevad üksteisega sündmuste avaldamise ja tellimise kaudu. See vähendab mikroteenuste vahelist sõltuvust ja muudab süsteemi skaleeritavamaks ja paindlikumaks. [6]

Analüüsi käigus otsustas autor, et kõige õigem lahendus lõputöö probleemile on esimesed kaks välja toodud varianti. Uue komponendi loomine lahendaks faili salvestuse, hoolduse ning taaskasutatavuse probleemi. Teise analüüsitud tehnika kasutamine parandaks üldpildis mikroteenuse arhitektuuri. Kui erinevad protsessid konsolideerida uude komponenti väheneks teiste teenuste keerukus ja suurus. Samuti oleks üldpildis kergem aru saada infosüsteemi ehitusest.

Sõnumijärjekorra süsteemi kasutuselevõtt ei sobiks probleemi lahenduseks, kuna selle süsteemiga jääksid ikka suuremahulised failid ühte teenusesse. Samuti tekiks ka probleem, kui on vaja faili süsteemi lisada veel ühte teenust, mis neid salvestama ja kasutama hakkaks. Sarnaselt on sündmustepõhise arhitektuuri kasutusele võtmisega. See küll vähendab sõltuvust, kuid süsteemide keerukus ja mahukus jääb samaks. (Tabel 1)

Tabel 1. Probleemile lähenemise võimaluste võrdlus

Lähenemine	**Taaskasutatav**	**Skaleeritav**	**Teiste teenuste mahu vähendamine**	**Intsidentide ja hoolduste taluvus**
Uus kergekaaluline komponent	jah	jah	mitte täielikult	jah
Protsesside konsolideerimine	jah	jah	jah	mitte täielikult
Sõnumijärjekorra süsteem	ei	jah	ei	ei
Sündmustepõhine arhitektuur	ei	jah	ei	ei

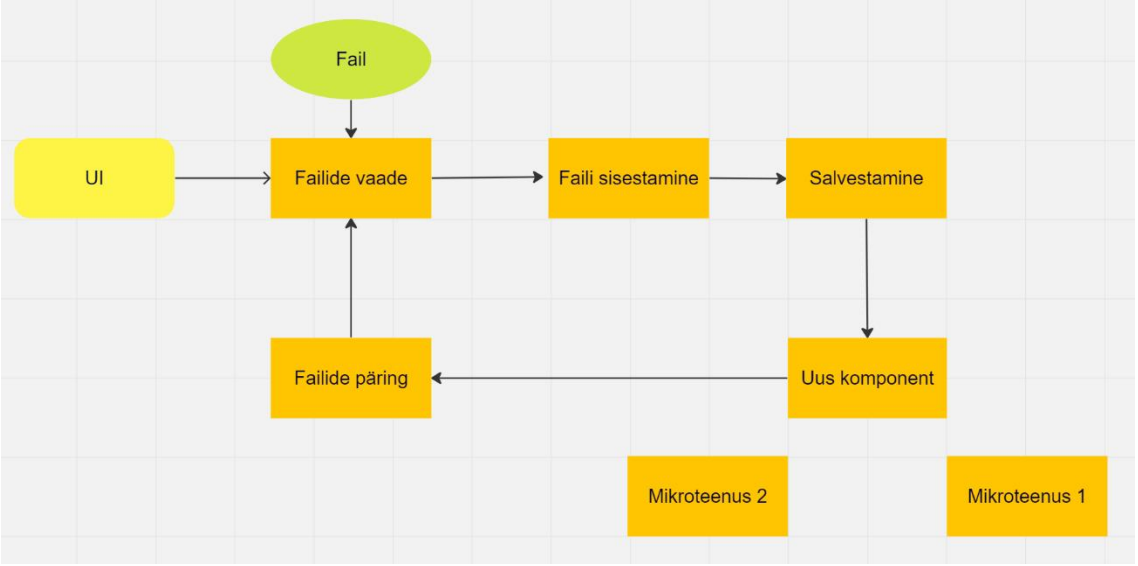
Oluline on probleem hoolikalt analüüsida ja kaaluda mitut lahendust, enne kui otsustada parima käitumisviisi üle. Igale lahendusele pikemaajaliste tagajärgede kaalumine võimaldab luua stabiilsema ja efektiivsema infosüsteemi.

2.3 Soovitud eesmärk

Eesmärgiks on luua kergekaaluline väiksem komponent, mis suudab mõlema mikroteenusega seotud faile vastu võtta, kontrollida ja mikroteenusele vastavaid vajalikke protsesse läbi viia. Lisaks sellele on uuel komponendil taaskasutuse aspekt juures, ehk seda komponenti on võimalik ka teistel mikroteenustel kasutada. Analüüsimise käigus selgus, et uue komponendi loomisel tekib siiski sõltuvus uuest komponendist. Kui uues komponendis on intsident või hooldus, ei ole maas terve

mikroteenus. Häiritud on vaid teiste teenuste osaline funktsionaalsus. Tulemuseks on ainult mikroteenustes ühe väiksema osa mitte toimimine intsidendi või hoolduse ajal.

Uue komponendi loomisel on võimalik ka teisi mikroteenuseid juurde liita. Nagu eelnevalt mainitud on tulevikus plaanis kolmandal teenusel liituda, mis kasutaks uut komponenti. Äriloogika nõuete vastavalt hakkab UI kasutama uut komponenti eraldiseisvana. Uues komponendis on kõik vajalik info, mis teenusest iga dokument pärineb. Failide osa infosüsteemist saab olema eraldatud, et väheneks sõltuvus teenuste vahel. UI suhtleb otse uue komponendiga failide salvestuse ja kuvamise osas (Joonis 3).



Joonis 3. TO-BE Näide uue komponendiga lihtsustatud päringu näide

Autor otsustas peale süsteemide analüüsi, et tuleks ka peatükis 2.1 mainitud failidega seotud protsessid tõsta uude komponenti. Lisaks sellele on näiteks failide loogikat juba teistesse mikroteenustesse arendatud, ehk ka nendest tuleks failide loogika funktsionaalsus ühte kohta tõsta. Sellisel juhul ei pea iga teenus enda sisse failidega seotud protsesse kirjutama hakkama. Samuti oleks hulga taaskasutatavat funktsionaalsust, mida saaks ära kasutada tulevikus liituvate teenuste jaoks.

3 Põhilised kasutatud tehnoloogiad

Antud peatükis kirjeldan täpsemalt lõputöö käigus kasutatud tehnoloogiatest. Valiku tegemisel lähtus autor kogemusest, erinevate tehnoloogiate analüüsist kui ka juba ettevõttes kasutusel olevatest tehnoloogiatest.

3.1 Java, Spring Boot

Spring Boot on avatud lähtekoodiga Java baasil olev raamistik millega tehakse mikroteenuseid. [7] Seda saab lihtsasti üles panna ning ei vaja väga põhjalikku konfigureerimist, et saaks rakenduse esimest korda käima panna. Spring Boot on loodud selleks et lihtsustada Springi arendusega alustamist. See sisaldab laia valikut sisseehitatud komponentidest ning sisse integreeritud teekidest nagu näiteks Spring Data [8], Spring Security [9], Spring MVC [10].

Hetkel on infosüsteemi mikroteenustel enamasti kasutatud vanemat Java. Kuna üks eesmärkidest on tehnoloogilise võla mitte tekkimist ning kõige uuemate versioonide kasutamine, otsustas autor võtta kasutusele Java versioon 17, kuna töö kirjutamise hetkel on see kõige uuem Java versioon millel on LTS tugi. [11] Java 17 sisaldab mitmeid jõudluse parandusi ja turvaparandusi, mis võivad aidata Java rakendustel töötada kiiremini, tõhusamalt ja kaitsta rakendusi turvariskide eest. Kokkuvõttes pakub Java 17 mitmeid eeliseid vanema Java versiooni ees, sealhulgas uued keele funktsioonid, parem turvalisus, täiustatud prügikasti kogumine, täiustatud platvormitugi ja parem tööriistatugi. Java ning Spring Booti valik oli suuresti mõjutatud sellest, et riigi it-s on see enamlevinud keel ning raamistik.

3.2 IDEA Intellij

IntelliJ IDEA on populaarne integreeritud arenduskeskkond (IDE) Java arenduseks. Mõned IntelliJ IDEA eelised on järgmised [12]:

1. Nutikas koodi lõpetamine: IntelliJ IDEA pakub nutikat koodi lõpetamist, mis võib aidata arendajatel kiiremini ja vähem vigu tehes koodi kirjutada.

2. Täpne koodianalüüs: IDE pakub täpseid koodianalüüsi tööriistu, mis võivad avastada potentsiaalseid vigu ja vigu koodis ning soovitada neid parandada.

3. Refaktoreerimise tugi: IntelliJ IDEA pakub refaktoreerimise toe, mis võimaldab arendajatel koodi lihtsalt muuta, mõjutamata rakenduse funktsionaalsust.

4. Sisseehitatud tööriistad: IntelliJ IDEA sisaldab palju sisseehitatud tööriistu Java rakenduste arendamiseks, sealhulgas versioonihaldussüsteeme, ehitustööriistu ja silumistööriistu.

5. Pistikprogrammi tugi: IntelliJ IDEA-l on suur hulk saadaolevaid pistikprogramme, mis võivad laiendada IDE funktsionaalsust ja lisada tuge teistele programmeerimiskeeltele ja raamistikele.

IntelliJ IDEA on mugav kasutada ning sellel on mitmeid eeliseid. Lisaks on autoril selle IDE-ga rohkelt kogemust nii töölt kui ka ülikoolist.

3.3 OpenAPI Swagger ja Lombok

Swagger OpenAPI on REST teenuste ehitamiseks ning dokumenteerimiseks loodud spetsifikatsioon. See pakub standardformaati REST teenuste kirjeldamiseks. Swagger OpenAPI pakub masinloetavad formaati, mida saab kasutada klienditeekide, dokumentatsiooni ja muude tööriistade loomiseks, muutes teiste arendajatel API-ga [13] töötamise lihtsamaks. [14] Swagger OpenAPI-t kasutatakse mitmetes ettevõte rakendustes ning autoril on selle kasutamises kogemust.

Samuti on kasutusel ka OpenAPI generaator, mis on koodigenereerimise tööriist, mis vähendab manuaalse koodikirjutamise aega kui on tarvis suhelda teiste API-dega. [15] OpenAPI generaatorit valis autor, sest see kiirendab koodi kirjutamist ning vähendab võimalike inimeksimuste eest.

Lombok on Java teek. See sisaldab annotatsioone, mis aitab automatiseerida Java klassides konstruktorite, getterite, setterite, builderite [16] ja standardkoodi loomist. See aitab vähendada korduva koodi kirjutamist, mis aitab kiirendada arendamist ning vähendada vigade tekkimist. Samuti on Lombokiga võimalus kasutada logidesse logi kirjete loomist, mis võib aidata hiljem vigu parandada. [17] Lombok on samuti

kasutatud mitmetes ettevõte rakendustes. Autori valik sõltus sellest ning ei näinud vajadust alternatiivsete lahenduste otsimiseks.

3.4 PostgreSQL

PostgreSQL on relatsioonilise andmebaasi haldamise süsteem. See on tuntud enda robustsuse, usaldusväärsuse ja lisafunktsioonide näol. PostgreSQL on hästi skaleeritav, see suudab käsitleda suuri andmemahtusid ja kõrget liiklust. See toetab erinevaid indekseerimismeetodeid, lisaks pakub PostgreSQL paralleelpäringute tegemist, mis tõstavad andmetöötluse kiirust. PostgreSQL-il on sisse ehitatud turvalisuse funktsioone nagu näiteks SSL krüpteerimine ja autoriseerimismehhanismid. Üldiselt on PostgreSQL võimas ja usaldusväärne andmebaasisüsteem, mis saab hakkama paljude andmehaldusülesannetega. [18] PostgreSQL on kasutusel tööd käsitlevates mikroteenustes ning autor ei pidanud oluliseks ka uue komponendi loomisel midagi muud kasutada.

3.5 Angular

Angular on TypeScriptis kirjutatud veebirakenduste raamistik. Angularis on komponendipõhine arhitektuur, mis muudab keerukamate veebirakenduste arendamise ja hoolduse lihtsamaks. Raamistik võimaldab erinevaid kasutajaliidese elemente koondada korduvkasutatavetesse komponentidesse, mis muudab koodi haldamise lihtsamaks. Tänu sellele, et Angular on TypeScriptis kirjutatud, on mitmeid eeliseid nagu parem vigade kontroll, hooldatavus ja skaleeritavus. Lisaks on Angularil suur ja aktiivne kogukond arendajatest, see tähendab et õppimiseks ja toetuseks on saadaval palju ressursse. Saadaval on ka palju kolmandate osapoolte teeke ja tööriistu, mis aitavad rakenduse funktsioone laiendada. [19] Angular on hetkel kasutusel olev raamistik infosüsteemi kasutajaliidese pool.

4 Teenuse püsti panemine ja konfigureerimine

Käesolevas peatükis on seletatud kokkuvõtlikud sammud kuidas projekt loodi ning kuidas keskkonnad seadistati. Projekt on loodud Spring Bootis ning püsti pandud Bamboo keskkonnas.

4.1 Spring Boot rakenduse loomine

Intellij IDE rakenduses on võimalik väga lihtsasti uus Spring Booti rakendus luua. IDE genereerib ise vajalikud failid ning pärast seda on võimalik ise kõik valmis seadistada. Seejärel tuleks kirjutada konfiguratsiooni fail, kus on määratud erinevad parameetrid. Spring Boot-is pole otseselt vajadus kohe konfiguratsiooni faili kirjutada kui on kasutatud @EnableAutoConfiguration annotatsiooni. Selle järgi oskab Spring Boot genereerida Spring rakendust vastavalt mis jar [20] sõltuvusi on lisatud [21].

Uus loodud komponent on selliselt üles seatud, et oleksid profiilid mille vahel saab valida. Ehk lokaalselt on local profiil, DEV keskkonnas on dev, TEST keskkonnas on test ja LIVE keskkonnas on live profiil. Seega on vastavalt nimetustele ka .yml failid paika pandud. Näiteks lokaalne konfiguratsiooni fail on kirjutatud järgmise vormindusega: application-local.yml. Lisaks võib olla mitme keskkonna vahel ühisosa mis kirjutatakse lihtsalt application.yml faili. Nendes failides määratakse ära erinevad springi, andmebaasi, serveri parameetreid ja palju muud.

Mõned näited konfigureerimisest. Server parameetriga on määratud serveri porti ning ka baas URL, kuhu poole saab hiljem kasutaja pöörduda, et päringuid teha. Lisaks on kirjeldatud andmebaasi andmeid, turvalisuse parameetreid kui ka logimise sätteid. Logimise sätteid, swaggeri asukoht ning erinevaid muutujaid mida on võimalik hiljem konfiguratsioonides kasutada (Joonis 4).

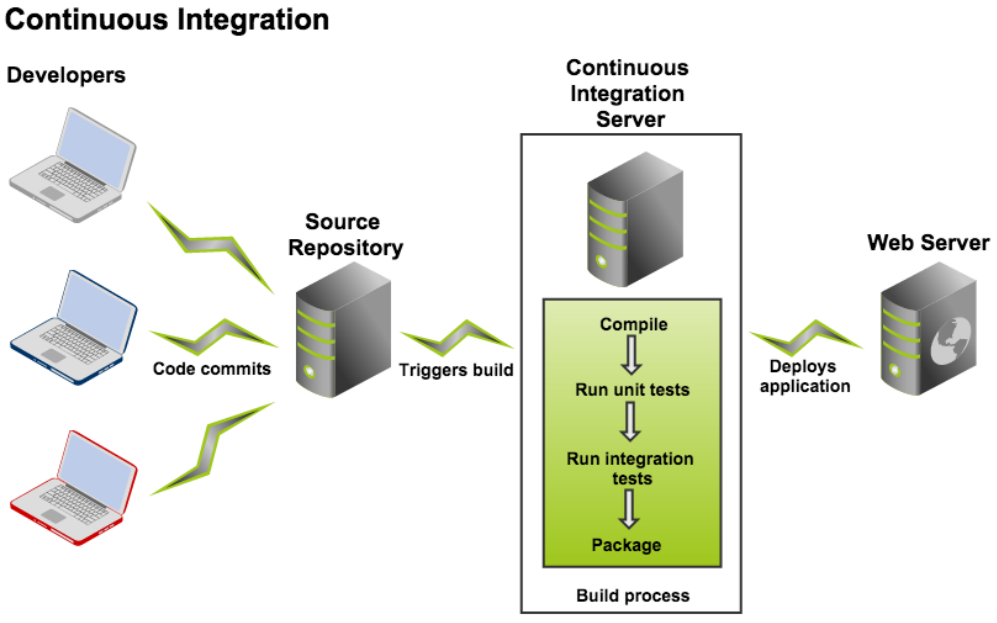
```yaml
server:
  port: 8080
  servlet:
    context-path: /example/api/v1

spring.mvc.format:
  time: iso
  date: iso
  date-time: iso
spring:
  jpa:
    hibernate.ddlAuto: none
    properties:
      hibernate:
        dialect: org.hibernate.dialect.PostgreSQLDialect
  security:
    oauth2:
      client:
        registration:
          clientId: example
          clientSecret: parool
      resourceserver:
        jwt:
          jwk-set-uri: https://example.example.dev/token_example

security:
  oauth2:
    resource:
      token: https://example.example.dev/oauth/token

management.endpoints.web.cors:
  allowed-origins: 'http://localhost:4200'
  allowed-methods: '*'
  allowed-headers: '*'


logging:
  config: classpath:logging/logback-json.xml
  level:
    org:
      springframework: info
      hibernate:
        type: info

springdoc:
  swagger-ui:
    path: /swagger/index.html
  api-docs:
    path: /api/v3/api-docs
    enabled: true
```

Joonis 4. Näide application.yml

4.2 Bamboo plaani konfigureerimine

Bamboo on populaarne CI/CD [22] server, mis on välja töötatud Atlassiani poolt (Joonis 5). [23] See on loodud tarkvararakenduste buildimise ehk ehitamise, testimise ja deployment-i ehk juurutuse automatiseerimiseks ning pakub laias valikus funktsioone ja tööriistu, mis muudavad tarkvaraarendusprotsessi haldamise ja automatiseerimise lihtsaks.

Bamboo plaan on ülesannete kogum, mis määrab konkreetse tarkvaraarenduse töövoo. Bamboo plaani konfiguratsiooni süsteemi peamised aspektid on plaani loomine, ülesande, triggeri ja muutujate konfigureerimine. Plaani loomisel peab määratlema konkreetsed ülesanded, mis moodustavad arendustöövoo. Uue komponendi loomisel oli esialgu vaja kirjutada rakenduse ehituse plaan kui ka juurutusplaan. Käivitaja konfigureerimise käigus saab paika panna, mis hetkel plaan käivitada. Plaani saab käima panna sündmustel näiteks koodi sisse kandmisel, uue haru loomisel või käsitsi käivitamisel. Neid saab kohandada vastavalt konkreetsetele nõuetele. Bamboo võimaldab määratleda ja konfigureerida muutujaid, mida saab kasutada Bamboo plaani parameetrite muutmiseks. Muutujaid saab kasutada näiteks tundliku teabe (näiteks API-võtmete ja paroolide) lisamiseks. Samuti on võimalik seadistada ka teavitusi. Need teavitused saab näiteks saata e-mailile, kui plaani käigus on tõrkeid tekkinud.



Joonis 5. Näide Bamboo CI/CD

5 Vajalike tööde arendus

Enamus tehtud arendusi on mikroteenuste äriloogikal põhinevad arendused, mida detailselt selle lõputöö raames ei käsitleta. Siiski üldisematest arendustest, mida saaks ka teistes projektides kasutada, on võimalik täpsemalt lõputöös rääkida. Vajalike arenduste hulka kuulub failide salvestamine, failide kuvamine, viiruste kontroll, andmete migratsioon, failide allkirjastamine ning palju muud teenuste põhilisi ülesandeid.

Üks esimestest piletitest oli arendus, kus oli vaja lisada uuele komponendile funktsionaalsus, mis võtaks vastu faile ning need failid andmebaasi salvestaks. Failide salvestusel tuleb kaaluda ka turvalisust ning viiruste tabamist, sellest tuleb lähemalt juttu kuuendas peatükis. Kuna tegemist on failide salvestamisega ning neid oleks mõistlik pilveteenusesse salvestada. SMIT-i poolt on Amazon S3-le sarnane süsteem loodud, kuhu on võimalik failid salvestada. Failide metaandmed on salvestatakse uue komponendi andmebaasi ning neid on nende andmete põhjal võimalik pilvest pärida.

5.1 Arendus

Üks arendaja esimestest arendustest oli töö, kus oli vaja lisada uuele komponendile funktsionaalsus, mis võimaldaks sellel vastu võtta faile ja salvestada need andmebaasi. Peamine eesmärk oli tagada failide salvestamise turvalisus ja kaitsta süsteemi viiruste eest.

Arendusprotsessi käigus oli autoril vaja luua võimalus, et salvestada failid pilveteenusesse, selle jaoks on küll mitmeid võimalusi, kui kõige ohutum ja lihtsam oli salvestada need SMIT-i poolt juba välja töötatud pilveteenusesse, mis sarnaneb Amazon S3-ga ja võimaldab failide salvestamist.

Alguses tuli autoril analüüsida komponendi nõudeid ja olemasolevat süsteemi, et mõista, kuidas uut funktsionaalsust sinna integreerida. Seejärel alustasin süsteemi

arendamist vastavalt nõuetele. Failide vastuvõtmise ja andmebaasi salvestamise funktsionaalsuse arendamisel pidin tagama, et süsteem oleks turvaline. Rakendati erinevaid turvameetmeid, näiteks failide põhjalik skaneerimine viiruste suhtes enne salvestamist.

Arendusprotsessi käigus oli vaja testimist läbi viia, et veenduda uue funktsionaalsuse töökindluses ja turvalisuses. Kasutati automaatseid ja manuaalseid testimismeetodeid, et kontrollida, kas failide salvestamine toimib ootuspäraselt ning et süsteem vastab nõuetele. Pärast arendus- ja testimisfaaside lõpuleviimist oli näha, et uus funktsionaalsus töötab sujuvalt. Uude komponendi sai nüüd edukalt faile süsteemi laadida, kus need salvestati turvaliselt pilveteenusesse ning nende failide metaandmed andmebaasi. Tulemusena loodi arendus, mis vastas kliendi nõuetele ja täitis ootused.

5.2 MapStruct

Kui API ots võtab andmeid vastu, siis selle kohta saab teha andmetüübi, nagu näiteks FileSaveRequest, aga selleks, et salvestada andmebaasi oleks vaja domeeni objekti. Vastu võetud andmete domeeni objektiks tegemiseks on vaja teha mapper. Mapper on kiht mis liigutab andmeid objektide vahel hoides neid eraldiseisvana. [24]

Selles infosüsteemis, kus selle projekti raames tööd tehakse on mikroteenustes kasutusel Orika ConfigurableMapper, millega saab enda vajaduste järgi konfigureerida, kuidas mapper mingeid välju väärtustab ning ka muid toiminguid.

Otsustasin, et oleks mõistlik kasutada MapStruct-i mapperit. Mõned näited MapStruct-i mapperi eelistest.

1. MapStruct genereerib kaardistuskoodi lähtestaatuse ja sihtstaatuse objektide deklareeritud tüüpide põhjal, mis tagab tüübiturvalisuse kogu kompileerimise ajal. See võib aidata vältida tööajalisi vigu ja muuta kaardistusprotsess usaldusväärsemaks.

2. MapStruct genereerib kaardistuskoodi kompileerimise ajal, mis võib võrreldes Orika ConfigurationMapperiga, mis genereerib kaardistuskoodi tööajal, tagada kiirema tööajalise jõudluse.

3. MapStructil on autori arvates Orika ConfigurationMapperiga võrreldes lihtsam ja intuitiivsem API, mis võib muuta selle kasutamise ja mõistmise uutele kaardistusraamistikega tegelevatele arendajatele lihtsamaks.

4. MapStruct kasutab kaardistuskoodi genereerimiseks annotatsioonitöötlust, mis võib aidata vähendada vajalikku tühiprogrammi koodi, mida arendajad peavad kirjutama. See võib muuta kaardistusprotsessi efektiivsemaks ja vähem vigadega.

5. MapStruct pakub erinevaid kohandamisvõimalusi, nagu võime lisada kohandatud kaardistusloogikat või konfigureerida kaardistuskäitumist konkreetsete väljade või tüüpide jaoks. See võib aidata kaardistusprotsessi muuta paindlikumaks ja kohandatavamaks erinevate kasutusjuhtumite jaoks.

Kokkuvõttes on MapStructil mitu eelist Orika ConfigurationMapperi ees, sealhulgas tüübiturvalisus, kompileerimise ajaline jõudlus, lihtsus, annotatsioonitöötlus, kohandatavus ja IDE tugi. [25]

6 Turvalisus, failide kontroll

Failide sisendi valideerimine peaks toimuma nii rakenduse front- kui ka back end-is, et tagada maksimaalset turvalisust. Front endi valideerimine toimub enne andmete välja saatmist, samal ajal kui back endis valideerimine tagab täiendava turvalisuse kihi, et takistada kahjulikku sisendit töötlemast. [26]

Siiski peab meeles pidama, et front endi valideerimisest saab kergesti mööda minna, seetõttu tuleks alati teha ka back endi valideerimine. Front end-is saab teha päringuid ka ilma välja valideerimiseta ning võimalik saata päringuid otse andmebaasi, kui on teada kuhu päring minema peaks.

Uue komponendi puhul tegin samuti valideerimise nii front kui ka back endis. Front endis kontrollitakse, et faili laiend vastab lubatud laienditele. Back endis on kasutatakse viirusetõrje API-t, mis kontrollib viiruste olemasolu.

6.1 Viirusetõrje lahendused

Tarkvaraarenduse oluline aspekt on tagada tarkvara turvalisus ning kaitsta seda kahjuliku koodi või viiruste eest. Üks võimalus selle probleemi lahendamiseks on viiruste tuvastamise API-de integreerimine tarkvaraarenduse protsessi. Autor analüüsis erinevaid viirusetõrje API võimalusi.

API-d võimaldavad arendajatel lisada oma rakendustele funktsioone ja omadusi ilma nende nullist loomata. Viiruste tuvastamise API-sid saab kasutada tarkvarakoodi või failide skaneerimiseks ja viiruste või pahavara tuvastamiseks.

Üks populaarne viiruste tuvastamise API on VirusTotal API. VirusTotal on tasuta veebiteenus, mis analüüsib faile ja URL-e viiruste, troojalaste ja muude pahavara tüüpide suhtes. VirusTotal API võimaldab arendajatel integreerida selle viiruste skanneerimise funktsionaalsuse oma tarkvararakendustesse, mis muudab pahavara rünnakute tuvastamise ja ennetamise lihtsamaks. [27]

Teine populaarne viiruste tuvastamise API on ClamAV API. ClamAV on avatud lähtekoodiga viirusetõrje mootor, mis pakub viiruste tuvastamise võimalusi

tarkvararakendustele. ClamAV API võimaldab arendajatel kasutada ClamAV mootorit failide ja e-posti viiruste ja muu pahavara skaneerimiseks. [28]

Lisaks neile on saadaval mitmeid muid viiruste tuvastamise API-sid, nagu Jotti API [29] ja Avira API [30]. Igaüks neist API-dest pakub sarnaseid viiruste tuvastamise võimalusi, kuid erinevate funktsioonide ja hinnastruktuuridega.

Viiruste tuvastamise API-de integreerimine tarkvaraarenduse protsessi on oluline tagamaks tarkvara turvalisuse ja kaitstes seda kahjuliku koodi või viiruste eest. Kasutades neid API-sid saavad arendajad avastada ja ennetada pahavara rünnakuid, kaitstes nii oma kasutajaid kui ka tarkvararakendusi.

6.2 Viirusetõrje kasutamine projektis

Autori poolt mainitud kolmandas peatükis on projektis kasutatud OpenAPI generaatorit. OpenAPI generaatoriga viirusetõrje püsti panek vähendab suuresti manuaalset koodikirjutamist. Viirusetõrje API-s on mitmeid endpointe kuhu poole pöördutakse. Lisaks on ka palju erinevaid objekte mis tuleks kirjutada kõik manuaalselt, kui ei kasutataks OpenAPI generaatorit.

6.2.1 OpenAPI generaatoriga viirusetõrje üles seadmine

OpenAPI generaatoriga tuleb luua spetsifikatsiooni fail, kus on kirjeldatud päringute teekonnad, päringutüübid, parameetrid ja vastused. Kui tekib vajadus muuta parameetreid või funktsioone, siis saab seda kirjutada konfiguratsiooni failis. Sellega on see eelis, et kui vaja mingeid muudatusi sisse viia, siis uus kood genereeritakse rakenduse käivitamise ajal ning ei pea manuaalselt ise koodi poolt muutma.

Selleks et OpenAPI generaator saaks eelkirjeldatud .yml konfiguratsiooni faili järgi koodi genereerida, on vaja ka build.gradle failis muudatusi teha. Üks näide kuidas initsialiseerida OpenAPI generaator viirusetõrje API jaoks build.gradle failis. Seal tuleb määrata kus asub konfiguratsiooni fail, ning kuhu genereeritud kood pannakse. Samuti tuleb erinevate pakkide nimed ära määrata ning mõningaid konfiguratsiooni parameetreid ära määrata (Joonis 6).

```
task generateExampleVirusScannerClient(type:
org.openapitools.generator.gradle.plugin.tasks.GenerateTask) {
    generatorName = "java"
    inputSpec = "$rootDir/src/main/resources/api/virus-scanner-
api.yml".toString()
    outputDir = "$buildDir/generated/api".toString()
    invokerPackage = "com.example.virus.scanner"
    apiPackage = "com.example.virus.scanner.api"
    modelPackage = "com.example.virus.scanner.api.model"
    skipValidateSpec = false
    configOptions = [
            library          : "webclient",
            dateLibrary      : "javaX",
            delegatePattern  : "true",
            title            : "Example Virus Scanner",
            useTags          : "true",
            serializableModel: "true"
    ]
}

compileJava.dependsOn generateExampleVirusScannerClient
```

Joonis 6. Näide viirusetõrje API konfiguratsiooni initsialiseerimiseks build.gradle failis

6.2.2 Viirusetõrje kasutamise tulemus

Iga kord, kui laetakse üles uus fail, kontrolliti UI poolel, kas failide laiend vastab nõuetele ning ei ületaks faili mahu määra. Kui fail on edukalt uude komponenti jõudnud, kontrolliti samad asjad uuesti üle, ehk et failide laiend ja suurus vastaks ette antud nõuetele. Seejärel saadeti fail edasi viirustõrje API-sse, kus kontrollitakse, kas üles laetud fail sisaldab viiruseid või muud pahavara. API-st tuleb vastu positiivne või negatiivne vastus. Negatiivse vastuse korral lisatakse logisse kirje, et fail sisaldas viirust ning ka UI poolel näidatakse kasutajale sama veateadet. Positiivse vastuse korral toimub faili salvestamise protsess.

7 Migratsioon, plaan ja valideerimine

Andmete migratsioon on protsess, kus andmed liigutatakse ühest süsteemist või asukohast teise. See võib hõlmata andmete teisaldamist ühest füüsilisest salvestusseadmest teise, näiteks andmete üleviimist vanalt kõvakettalt uuele kõvakettale. Samuti võib see hõlmata andmete liigutamist ühest tarkvararakendusest või platvormist teise, näiteks andmete rändamist vanast kliendihaldussüsteemist uude.

Andmete migratsioon on alati vajalik, kui organisatsioonid uuendavad oma tehnoloogia taristut, vahetavad uute tarkvarasüsteemide vastu või liituvad teiste ettevõtetega. Andmete üleviimise planeerimine ja teostamine on oluline, et tagada andmete mitte kaotamist, mitte rikkumist ega ohustamist protsessi käigus. Edukas andmete migratsioon peaks tagama, et kõik andmed kantakse uude süsteemi või asukohta täpselt ja et andmed jäävad kättesaadavaks ja kasutatavaks neile, kes seda vajavad. [31]

7.1 Migratsiooniplaan

Migratsiooniplaan on üksikasjalik strateegia, mis kirjeldab samm-sammult, millised on vajalikud tegevused andmete või süsteemide üleviimiseks ühest asukohast või süsteemist teise. Mõned olulised punktid mida plaani koostamisel järgida oleks: eesmärgid, ajakava, ressursid, riskianalüüs, suhtlus, testimine, valideerimine ja alternatiivplaan (Joonis 7). [32]

Kokkuvõtlik migratsiooniplaan uue komponendi loomisel oli järgmine. Kirjutasin skripti, mis käivatakse cron-job-ga. Cron-job on ajastamise tööriist, mis võimaldab Unix-põhistel süsteemidel kasutajatel automatiseerida ülesandeid, käivitades käske või skripte ajakava järgi. [33]

Samuti tegin REST lõpp-punkt teenusesse 1, kus on kõik failide kirjed, erinevates tabelites. Uude komponenti kirjutasin eelmainitud skripti mis käivitub cron-job-ga. Skriptis on kood mis pöördub uue REST lõpp-punkti pihta, mis küsib omakorda andmeid teenuse 1 andmebaasist. Kokku on 6 tabelit millest kaks on seotud failide ajaloo andmetega. Skriptis implementeerisin logimist nii veaolukordade, õnnestumiste kui ka protsesside logidest. Cron-job-i käima lükkamine on juhitud keskkonna muutujaga.

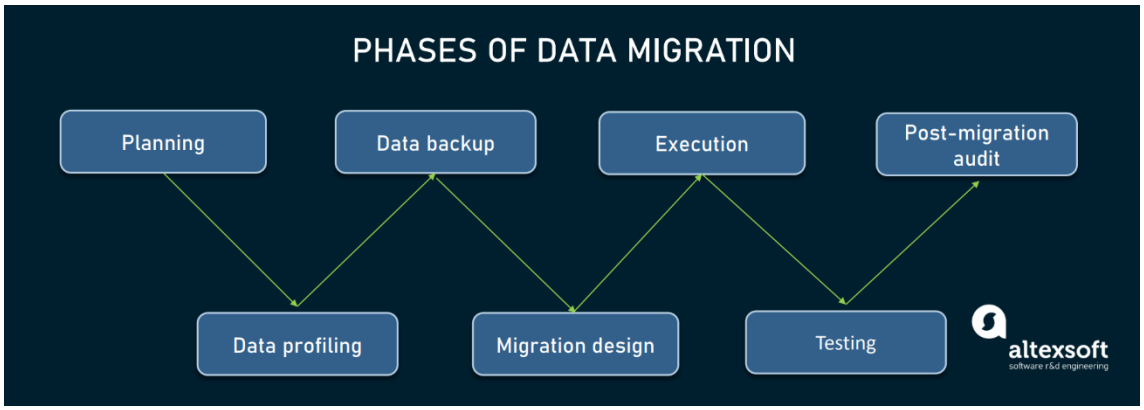
Kuna andmemaht on küllaltki suur ja andmete kogus jääb ca 30000 kirje kanti, siis tuli läbi mõelda, kuidas täpsemalt migratsiooni läbi viia. Kaalusin erinevaid migratsiooni variante ning mõtlesin migreerimise põhisammuna järgmise meetodi. Selline plaan kiideti heaks ka teiste arendajate poolt. Kõigepealt küsisin kõikide migratsioonile kuuluvate andmete identifikaatorid. Seejärel käisin kätte saadud identifikaatorite nimekirja tsüklis läbi ning iga identifikaatori järgi küsitakse failide tabelist rida. Kõik andmebaasid kasutasid andmebaasi süsteemina PostgreSQL-i.

7.1.1 Kirjeldatud migratsiooni meetodi sobivus

Saades kõigepealt kõik migreeritavate andmete identifikaatorid, vähendab meetod vajadust pärida kogu andmestikku iga üksiku kirje jaoks eraldi. See parandab oluliselt efektiivsust, vähendades vajalike andmebaasipäringute arvu, eriti suure andmemahtu korral. Identifikaatorite kasutamine tagab igale kirjele täpse tuvastamise ja taastamise lähtekohast andmebaasist. See lähenemisviis tagab, et migratsiooniprotsessis ei jää andmeid vahele või kopeerita topelt, eeldusel et identifikaatorid on korrektselt hangitud. Kuna meetod hõlmab identifikaatorite nimekirja läbivaatamist ja vastavate ridade pärimist erinevatest tabelitest, suudab see töödelda märkimisväärset andmekogust. Selline skaleeritavus muudab meetodi sobivaks juhtumite korral, kus andmestik sisaldab umbes 30 000 kirjet või isegi rohkem.

7.1.2 Migratsiooni järgsed tegevused

Peale migratsiooni, tuleks mõneks ajaks andmeid alles hoida andmebaasis kust migreeriti. Seda seetõttu, et kui on märgatud mõni viga migratsiooni käigus, siis on võimalik veel parandusi sisse viia kasutades vana andmebaasi, kus andmed esialgu paiknesid.



Joonis 7. Näide migratsiooni sammudest

7.2 Riskid

Andmete migreerimisega kaasnevad riskid võivad hõlmata järgmist [34]:

1) Andmete migreerimise käigus võib tekkida andmete kaotuse oht. See võib juhtuda siis, kui andmed ei varundata korralikult või kui migreerimisprotsessi ei teostata õigesti.

2) Andmete riknemine võib tekkida siis, kui andmed kantakse ühest süsteemist teise. See võib olla põhjustatud riistvara riketest, tarkvaravigadest või muudest probleemidest.

3) Migreerimisprotsessi käigus on võimalus, et volitamata isikud pääsevad andmetele juurde, mis võib põhjustada andmelekkeid ja muid turvaolukordi.

4) Andmete migreerimine erinevate süsteemide vahel võib kaasa tuua ühilduvusprobleeme. See võib põhjustada probleeme, nagu andmeformaadi mittesobivus, mis võib põhjustada andmete kadu.

5) Andmete migreerimine võib põhjustada süsteemi tööseisakuid, mis võib mõjutada rakenduse toiminguid. See võib põhjustada kaotatud tootlikkust ja kliendirahulolu. Tegemist on nii koguselt kui ka mahult suurte failide migreerimisega, seetõttu pikendab see failide migratsiooni aega. Tööseisaku probleemi on võimalik lahendada, kui näiteks viia migratsioon läbi sellisel ajahetkel, kus parasjagu teenust ei kasutata.

Andmete varundamine, testimine ja kontrollimine on olulised eduka migreerimise tagamiseks. Samuti on oluline omada tagavaraplaani juhuks, kui migreerimisprotsessi käigus ilmnevad probleemid.

Kui uude komponenti migreerimisel tekib viga, siis migratsioon tühistatakse ja tehakse rollback, ehk ei viida migratsiooni lõpuni ning kõik muudatused võetakse tagasi.

7.3 Valideerimine

Andmete valideerimine pärast migreerimist on oluline samm, et tagada migreerimisprotsessi õnnestumist, andmete täpsust ja täielikust.

Tuli kontrollida, et migreeritud andmed on täielikud ja täpsed. See saavutati andmete võrdlemisega lähtesüsteemis olevate andmetega ning kontrollides kirjete arvu, väljade väärtusi ja andmesuhete tõestamist, et kõik on õigesti migreeritud. Uues komponendis tehti mõlemasse andmebaasi SQL päring, mis loeb kirjete arvu. Selle päringu tulemusena sai veenduda, et kõikide andmete hulk on õige. Lisaks kirjutati koodi jupp, mis küsib mõlemast andmebaasist andmeid ning võrdleb neid. Samuti tegi autor andmete pistelist kontrolli. Sellise tehnika puhul saab valideerida ka andmete korrektsust.

8 Valminud komponendi testimine.

Testimine ja valideerimine on tarkvara arendamisel väga olulised tegevused, kuna need tagavad, et tarkvara vastab nõutavatele kvaliteedistandarditele ning töötab ootuspäraselt. Testimine aitab avastada tarkvara defekte ja vigu ning parandada neid juba varajases arengufaasis, et vältida hilisemat kulukat töö uuesti tegemist.

Lisaks sellele tagab valideerimine, et tarkvara töötab nii nagu on nõutud, vastab kasutajate vajadustele ja täidab kõiki etteantud nõudeid. See aitab vähendada tarkvaraveaga seotud riske nagu andmelekked, finantskahjud, mainekahjustus või õiguslikud probleemid.

8.1 Junit testid

Enamus kirjutatud kood ja loogika on kaetud Junit testidega. JUnit on Java platvormi raamistik, mis on loodud tarkvaratestide automatiseerimiseks ja testjuhtumite loomiseks. Kirjutatud testide kohta kontrollisin ka koodi katvust (code coverage). See on mõõdik, mis mõõdab tarkvarasüsteemi lähtekoodi ulatust, mida on testide komplektiga kaetud. Katvust mõõdetakse tavaliselt millised read, harud ja tingimused on testide käivitamise ajal käivitatud. Selle järgi oli hea teada, mis funktsionaalsus on veel testimata ning andis autorile hea ülevaate automaattestide kogupildist.

8.2 Koodiülevaatus

Iga arenduse kohta tegin pull request-i ehk tõmbetaotlus. Tõmbetaotlus on viis muudatuste ettepanekute tegemiseks versioonihaldussüsteemis nagu Git. Seal on näha muudatusi, mis on tehtud konkreetses harus. Tõmbetaotluses tehakse koodiülevaatust teiste arendajate poolt. Selle protsessi kontrollivad teised arendajad kirjutatud koodi, et leida ja pakkuda võimalikke variante kuidas vigu parandada. Nad jälgivad et kood vastaks kehtestatud programmeerimisstandarditele ja parimatele tavadele.

Koodiülevaatuse käigus vaatab arendaja koodi läbi, otsib probleeme nagu loogikavigu, turvavigu ja koodi lõhnu. Code smell on lähtekoodi omadus, mis näitab sügavamat probleemi koodibaasis. See on subjektiivne näitaja potentsiaalsest probleemist, mis võib muuta koodi hooldamise või mõistmise raskemaks. Koodilõhnad viitavad sageli halvale disainile, arhitektuurile või implementatsiooni valikutele ning võivad põhjustada vigu, tehnilist võlga ja vähendada arendaja tootlikkust. Kui koodi ülevaatusel tehti kommentaare, siis parandasin need koheselt.

Lisaks arendajate ülevaatusele, toimub ka automaatne ülevaatus Sonari liidestuse poolt, mis tuleb igale uuele rakendusele külge panna. Sonariga saab jälgida testide koodikatvust, koodi lõhnu ja palju muud.

8.3 DEV ja TEST keskkonnas testimine

Kui arendus läbib koodiülevaatuse , siis võisin antud arenduse DEV keskkonda tõsta. Kui arendus on edukalt DEV keskkonnas, siis veendusin, et kõik toimib enne kui see testijale üle antakse. Testija tagab süsteemi toimimise vastavalt konkreetse tööülesande

nõuetele ja ootustele. Testimine hõlmab testimisplaanide koostamist, testide läbiviimist vastavalt kokkulepitud ajakavale, testitulemuste dokumenteerimist ja vigade aruandlust arenduseeskirjadele. Igal arendusel on juures aktsepteerimis kriteeriumid, mille järgi saab testija testida. Kui vigu tuvastatakse, esitab testija neist raporti ja saadab need autorile tagasi parandamiseks. Samuti veendusin igas keskkonnas, et mu arendus töötaks, testisin arenduste funktsionaalsust alati läbi.

Kui arendusega on kõik DEV keskkonnas korras, siis liigutatakse see TEST keskkonda. Seal testitakse arendus uuesti läbi, aga seekord vaatab seda testija, tooteomanik ning lõpuks ka peakasutaja.

8.4 Testimiste tulemus

Testimine tagab, et tarkvara vastab nõutavatele kvaliteedistandarditele ja töötab ootuspäraselt. Testimine mängib olulist rolli tarkvara defektide ja vigade avastamisel, võimaldades neid varases arenguetapis parandada ning vältida hilisemat kulukat töö uuesti tegemist.

JUnit testide abil avastas autor defekte ja vigu varases arengufaasis. Koodiülevaatus toetas koodi parandamist vastavalt teiste arendajate kommentaaridele, tagades parema koodikvaliteedi. DEV ja TEST keskkondades tehtud testimine aitas tuvastada ja parandada süsteemi toimimisega seotud probleeme ning tagada vastavus nõuetele ja ootustele.

Näiteks tuli JUnit testide kirjutamisel välja probleem, kus ükskõik mis fail üles laeti, alati tagastas et failis on viirus. Eelmainitud test kontrollis seda, kas korrektse faili salvestumine õnnestub. Tänu sellele testile leidsin kiirelt, et viga seisnes selles, et faili laiendi kontroll oli valesti tehtud ning samuti edastati ka vale veateade.

Kokkuvõttes saavutati töö arendamisel häid tulemusi testimisprotsessi ja testide kirjutamise ning koodiülevaatuse abil, mis aitasid tagada tarkvara kvaliteeti ja toimimist vastavalt vajadustele.

9 Kokkuvõte

Lõputöö eesmärkideks olid infosüsteemis kahe mikroteenuse ristsõltuvuse vähendamine ning sellest tulenevalt ka uue komponendi loomine, andmete ning ka funktsionaalsuse üle viimine. Lisaks eelmainitule oli tegevuste eesmärgiks ka vähendada tehnoloogilist võla tekkimise ohtu. Uue komponendi loomise käigus said kõik eelnevalt mainitud eesmärgid edukalt täidetud.

Teoreetilise poole pealt analüüsiti lõputöös probleeme ja võimalike riske, kui kahe mikroteenuse vahel on ühine komponent, mis tekitab nende vahel ristsõltuvust. Selgitati mikroteenuste arhitektuuri olulisi punkte mida tuleks järgida.

Töös on kirjeldatud erinevaid tehnoloogiaid ning abistavaid tööriistu, mis aitasid uue komponendi loomisele kaasa. Nende valikud on põhjendatud ning selgitatud, et ka tulevikus sarnase probleemi lahendajad saaksid ideid, et enda probleemile efektiivne lahendus leida.

Samuti on välja toodud mõned etapid nagu näiteks kuidas viidi läbi andmete migratsioon, mis moodi projekt püsti pandi ning kuidas lahendati failidega seotud turvariske. Lõputöös on ka räägitud kuidas toimus arendus ning kuidas toimus uue komponendi testimine.

Kasutatud kirjandus

[1] SMIT, „Siseministeeriumi infotehnoloogia- ja arenduskeskus,“ https://www.smit.ee/et/siseministeeriumi-infotehnoloogia-ja-arenduskeskus. [Võrgumaterjal]. [Kasutatud 22 05 2023].

[2] S. Peyrott, „Intro to Microservices, Part 4: Dependencies and Data Sharing,“ 09 11 2015. [Võrgumaterjal]. [Kasutatud 02 04 2023].

[3] S. G. A. L. L. M. M. F. M. R. M. L. S. Nicola Dragoni, „Microservices: Yesterday, Today, and Tomorrow,“ %1 *Present and Ulterior Software Engineering*, Springer, Cham, 2017, p. 195–216.

[4] F. S. Chris Richardson, Microservices: From Design to Deployment, NGINX, 2016.

[5] S. Tung, „Scaling Microservices with Message Queues to Handle Data Bursts,“ 14 12 2018. [Võrgumaterjal]. Available: https://songthamtung.medium.com/scaling-microservices-with-message-queue-2d389be5b139. [Kasutatud 12 04 2023].

[6] E. Novoseltseva, „Event-Driven Architecture Benefits,“ 14 01 2020. [Võrgumaterjal]. Available: https://apiumhub.com/tech-blog-barcelona/event-driven-architecture-benefits. [Kasutatud 14 04 2023].

[7] Tutorials Point, „Spring Boot - Introduction,“ [Võrgumaterjal]. Available: https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm. [Kasutatud 02 04 2023].

[8] VMware, „Spring Data Integration,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring-security/reference/servlet/integrations/data.html. [Kasutatud 02 04 2023].

[9] VMware, „WebSocket Security,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring-security/reference/servlet/integrations/websocket.html. [Kasutatud 02 04 2023].

[10] VMware, „Spring MVC Integration,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring-security/reference/servlet/integrations/mvc.html. [Kasutatud 02 04 2023].

[11] T. Anderson, „Oracle Releases Java 17,“ [Võrgumaterjal]. Available: https://www.oracle.com/news/announcement/oracle-releases-java-17-2021-09-14/. [Kasutatud 02 04 2023].

[12] JetBrains, „What is IntelliJ IDEA?,“ [Võrgumaterjal]. Available: https://www.jetbrains.com/idea/features/. [Kasutatud 03 04 2023].

[13] FCD OÜ, „Mis on API?,“ [Võrgumaterjal]. Available: https://www.disainveeb.ee/blogi/mis-on-api/. [Kasutatud 03 04 2023].

[14] SmartBear, „OpenAPI Specification,“ [Võrgumaterjal]. Available: https://swagger.io/specification/. [Kasutatud 03 04 2023].

[15] Baeldung, „Open API Server Implementation Using OpenAPI Generator,“

[Võrgumaterjal]. Available: https://www.baeldung.com/java-openapi-generator-server. [Kasutatud 04 04 2023].

[16] Taltech, „OOP,“ [Võrgumaterjal]. Available: https://javadoc.pages.taltech.ee/oop/index.html. [Kasutatud 04 04 2023].

[17] A. Zanini, „A Complete Guide to Lombok,“ 29 07 2021. [Võrgumaterjal]. Available: https://auth0.com/blog/a-complete-guide-to-lombok/. [Kasutatud 04 04 2023].

[18] Amazon, „What is PostgreSQL?,“ [Võrgumaterjal]. Available: https://aws.amazon.com/rds/postgresql/what-is-postgresql/. [Kasutatud 04 04 2023].

[19] Google, „Introduction to the Angular docs,“ [Võrgumaterjal]. Available: https://angular.io/docs. [Kasutatud 04 04 2023].

[20] Oracle, „JAR File Overview,“ [Võrgumaterjal]. Available: https://docs.oracle.com/javase/8/docs/technotes/guides/jar/jarGuide.html. [Kasutatud 05 04 2023].

[21] VMware, „Auto-configuration,“ [Võrgumaterjal]. Available: https://docs.spring.io/spring-boot/docs/1.3.8.RELEASE/reference/html/using-boot-auto-configuration.html. [Kasutatud 04 04 2023].

[22] M. Anastasov, „CI/CD: Continuous Integration & Delivery Explained,“ [Võrgumaterjal]. Available: https://semaphoreci.com/cicd. [Kasutatud 06 04 2023].

[23] Atlassian, „Understanding the Bamboo CI Server,“ 18 7 2021. [Võrgumaterjal]. Available: https://confluence.atlassian.com/bamboo/understanding-the-bamboo-ci-server-289277285.html. [Kasutatud 04 04 2023].

[24] „Data Mapper,“ [Võrgumaterjal]. Available: https://java-design-patterns.com/patterns/data-mapper/. [Kasutatud 06 04 2023].

[25] A. Zanini, „Mapping JPA Entities into DTOs in Spring Boot Using MapStruct,“ 07 07 2021. [Võrgumaterjal]. Available: https://auth0.com/blog/how-to-automatically-map-jpa-entities-into-dtos-in-spring-boot-using-mapstruct/. [Kasutatud 06 04 2023].

[26] OWASP, „File Upload Cheat Sheet,“ [Võrgumaterjal]. Available: https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html. [Kasutatud 10 04 2023].

[27] Hispasec Sistemas, „VirusTotal API v3 Overview,“ [Võrgumaterjal]. Available: https://developers.virustotal.com/reference/overview. [Kasutatud 20 04 2023].

[28] Canimaan Software Ltd, „ClamAV,“ [Võrgumaterjal]. Available: https://docs.clamav.net/. [Kasutatud 20 04 2023].

[29] Jotti, „Jotti,“ [Võrgumaterjal]. Available: https://virusscan.jotti.org/en-US/doc/apiinfo. [Kasutatud 20 04 2023].

[30] Avira Operations, „Avira,“ [Võrgumaterjal]. Available: https://www.avira.com/en/api. [Kasutatud 20 04 2023].

[31] R. Garg, „Data Migration – Explained,“ [Võrgumaterjal]. Available: https://nsrinfosystems.com/data-migration-explained/. [Kasutatud 10 04 2023].

[32] Talend, „Understanding Data Migration: Strategy and Best Practices,“ [Võrgumaterjal]. Available: https://www.talend.com/resources/understanding-data-migration-strategies-best-practices/. [Kasutatud 11 04 2023].

[33] L. L., „Cron Job: A Comprehensive Guide for Beginners 2023,“ 03 03 2023.

[Võrgumaterjal]. Available: https://www.hostinger.com/tutorials/cron-job. [Kasutatud 11 04 2023].

[34] M. K. Molden, „System migration: 7 risks to evaluate,“ 22 11 2022. [Võrgumaterjal]. Available: https://blog.hyland.com/digital-transformation/system-migration-risks-to-evaluate/. [Kasutatud 12 04 2023].

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

Mina, Ragnar-Kevin Kaselt

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose“ Ristsõltuvuses olevate mikroteenuste ühiskasutatavate failide eraldamine
2. ning migratsioon uude sõltumatusse komponenti“, mille juhendaja on Jekaterina Tšukrejeva
   2.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   2.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
3. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
4. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

15.04.2023

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.