TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Rasmus Ilmjärv 206318IADB

Politsei veebilehe süüteoavaldusele funktsionaalsuse lisamine

Bakalaureusetöö

Juhendaja: Meelis Antoi

Magistrikraad

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: Rasmus Ilmjärv

15.05.2023

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on luua lahendus, mis võimaldaks politsei veebilehel asuvasse süüteoavaldusse saata täitmise käigus faile teisest seadmest, näiteks mobiiltelefonist. Seda tehakse, luues mikrorakenduse, mis haldab avaldusega seotud failide üles laadimist.

Arenduse käigus luuakse mikrorakendus, mis võimaldab võtta vastu faile ja saata neid seotud rakendusele, andes kasutajale võimaluse laadida faile teisest seadmest samaaegselt ülejäänud, konfidentsiaalset ja olulist infot sisaldavale avaldusele kõrvalist juurdepääsu andmata. Arendus on jagatud nelja ossa, mis katavad uue mikroteenuse ja klientrakenduse loomist ning täiendusi olemasolevatele teenus- ja klientrakendusele.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 6 peatükki, 7 joonist.

Abstract

Adding Functionality to the Statement of Offence on the Police Website

The aim of the current thesis is to find a solution, that will enable adding files from a secondary device, like a smartphone, to the statement of offence on the police website while filling the form. That is done by creating a microapplication, that manages uploading files connected to the form.

During the development a microapplication is created which enables recieving files and sending them to the connected application, allowing users to upload files from a different device without compromising the confidetial data in the form. The development has been split into four parts, that cover creating the new microsevice and the client, and adding functionality to the existing web service and client.

The thesis is in Estonian and contains 30 pages of text, 6 chapters, 7 figures.

Lühendite ja mõistete sõnastik

API	*application programming interface*, rakenduse programmiliides ehk rakendusliides
CLI	*commandline interface*, käsurea liides
GUI	*graphical user interface*, graafiline kasutajaliides
HTTP	*Hyper Text Transfer Protocol*, hüperteksti edastamise protokoll
JSON	*JavaScript Object Notation*, JavaScript-i objektil põhinev andmevahetusvorming
multipart	*multipart/form-data*, HTTP päringu tüüp, millega saab saata faile või muid binaarandmeid
MVC	*Model-view-controller*, mudel-vaade-kontroller – kasutajaliidese disainimuster, kus seotud loogika on jagatud kolme elementi.
REST	*representational state transfer*, representatiivne oleku ülekandmine
RPC	*remote procedure call*, hajutatud protseduuri kutse
TARA	Riigi autentimisteenus
TCP	*Transfer Control Protocol*, edastusohje protokoll
QR-kood	Kahemõõtmeline masinloetav maatrikskood, kus info on kodeeritud kahe värvilistesse ruutudesse

Sisukord

Jooniste loetelu

1 Sissejuhatus

Andmete esitamiseks veebi kaudu kasutatakse üldiselt mingil kujul vormi. See vorm võib olla kasutusel selleks, et teha postitust sotsiaalmeediasse või esitada olulisi andmeid riigiasutustele.

Sõltuvalt vormi täitmise eesmärgist võib aegajalt olla oluline saata vormiga koos kaasa ka faile. See võib tähendada nii oluliste dokumentide edastamist kui ka piltide või videote üleslaadimist teistega jagamiseks. Praegusel ajal aga, kus paljud kasutavad nende jäädvustuste tegemiseks mõnda nutiseadet, on tihti vormile failide lisamiseks vaja liigutada neid faile ühest seadmest teise, mis kulutab aega ja võib teha olulise info edastamise raskemaks.

Käesolev lõputöö analüüsib probleemi Politsei veebilehel oleva süüteo avalduse kontekstis. Probleemi pakutud lahenduseks on mikroteenus, mis haldaks vormiga seotud failide vastuvõtmist, ja võimaldaks kasutajale pakkuda avaldusest eraldi seisva sessioonina võimalust saata avaldusele faile, andmata ligipääsu või kontrolli täidetava avalduse üle kasutatavast mobiilsest seadmest. Seda kindlustamaks, et avalduse originaalsel täitjal on endiselt täielik kontroll ja vastutus avalduses esitavate andmete üle.

2 Ülevaade probleemist

Politsei veebilehe süüteoavaldus on vorm, mille kaudu saab kannatanu või pealtnägija esitada politseile infot aset leidnud süüteo kohta, et politsei saaks kirjeldatud süütegu uurima asuda. Vormi esitamiseks peab kasutaja olema autenditud, antud juhul läbi TARA ehk riigi autentimisteenuse läbi. Vormiga esitatakse andmeid süüteo toimumise ja asukoha kohta, ning on võimalik ka lisada süüteoga seotud faile, millest uurimise alustamisel võib kasu olla või mis selle vaoks vajalikud võivad olla.

Probleem tuleneb sellest, et inimesed, kellel on vaja sellist vormi täita ja sellele faile juurde lisada, võivad neid hoida näiteks mobiilses nutiseadmes, nagu nutitelefon. Seega peavad kasutajad kulutama aega failide ühest seadmest teise liigutamisele, tihti kasutades selleks vahendeid nagu e-mail. See lisatöö ja ajakulu esitajale võivad viia selleni, et ei esitata kogu olulist infot või jäetakse avaldus üldse esitamata.

2.1 Eksisteerivad lahendused

Failide esitamise mõnevõrra lihtsamini tegemiseks pakutakse juba võimalust anda avalduses olevas süüteo kirjelduses kaasa link pilveteenuses hoitavatele failidele. See aga ei ole ideaalne, sest kui läbi avalduse otse faile esitada, on võimalik faile palju lihtsamalt automaatselt kontrollida ja töödelda, kuid kirjelduses lingiga lisatud faile peab vastu võtma avaldusega tegelev ametnik käsitsi. Lisaks võivad nii lingiga failide lisamisel tekkida näiteks probleemid korrektselt jagamata juurdepääsuõigustega nendele failidele või, sõltuvalt kasutatavast pilveteenusest, ka failide aegumise või kustumise oht. Kuigi see ei takista avaldust käsitleval ametnikul avalduse esitajaga näiteks ühendust võtta lisainfo saamiseks ja ütluste kogumiseks, raskendab see nii avaldust esitava kasutaja kui ka avaldust vastu võtva ametniku tööd. Lisaks sellele on kasutaja sunnitud kasutama kolmandate osapoolte lahendusi, mille üle neil kontroll puudub. Viimase olulise probleemina jääb ikkagi ka failide töötlus: vormi kaudu otse esitatud faile saab mitmel viisil automaatselt kontrollida ja töödelda, kasutades muuhulgas failhoidmise või vahendamise programme, mis faile viiruste olemasolu kohta kontrollivad. Lingiga saadetud faile peab aga jällegi käsitsi töötlema.

2.2 Uue lahenduse skoop

Uueks lahenduseks sellele probleemile pakub autor välja uue, kasutajapoolset failide üleslaadimist haldava mikrorakenduse loomist, mis keskenduks kasutajalt failide vastuvõtmisele ning failide info põhirakendusele edastamisele, et sellega sidumise kaudu antud vormide rakendusi täiendada. Selleks, et mitte luua kõrvalist juurdepääsu konfidentsiaalset infot sisaldavale vormile, haldabki kogu failide vastuvõtmist ja kontrolli saatmist loodav rakendus. Loodav rakendus integreeritakse kahe esirakenduse ja ühe teenusrakendusega. Muuhulgas on loodava rakenduse osaks ka mobiilile orienteeritud graafiline kasutajaliides, millele juurdepääsu haldab loodav rakendus.

2.3 Metoodika

Lahenduse analüüsis esitatakse rakenduse nõuded jaotatuna funktsionaalseteks ja mittefunktsionaalseteks nõueteks. Seejärel kirjeldatakse erinevaid võimalikke lahendusi, analüüsides nende sobivust ülesande täitmiseks ning selgitatakse lahenduses kasutatavate tehnoloogiate valikut.

Välja pakutud lahenduse arendust on kirjeldatud neljas osas, mis käsitlevad nii uue teenuse arendust, olemasoleva teenuse uuendamist, olemasolevale klientrakendusele tehtud muudatusi kui ka uue, telefoni jaoks mõeldud klientrakenduse arendust. Viimase osana on kirjeldatud loodud lahendust tervikuna, hinnates selle sobivust ülesande täitmisel.

3 Loodava veebirakenduse analüüs

Loodava veebirakenduse analüüs kirjeldab üldiselt olemasolevat rakendust, uue lahenduse nõuete määramist, uue lahenduse arhitektuuri, lahenduse tehnoloogia, sealhulgas programmeerimiskeelte ja raamistike valikut ning lahenduses kasutavate teiste tehnoloogiate, andmebaasi ja sõnumivahendaja, valikut.

3.1 Olemasolev rakendus

Hetkel kasutatakse süüteoavalduse realiseerimiseks klientrakendusest, teenusrakendusest ja failihoidlast koosnevat lahendust. Klientrakenduses saab kasutaja sisestada vajalikud andmed ja failid, mida võib süüteo menetlemiseks vaja minna. Klientrakendus saadab need edasi teenusrakendusele, mis talletab ja töötleb saadetud andmeid ning faile. Et vältida kahju süsteemile läbi ohtlike failide, olgu selleks siis kas programmid või viirustega nakatatud failid, talletatakse avaldusest saadud failid failihoidlas Metadefender Vault. Metadefender Vault on turvatud failide hoiustamise ja edastamise rakendus. Hoidlasse paigaldatud failid kontrollitakse rakenduse poolt, tuvastamaks kuritahtlike või nakatunud faile. Kahtlust äratavad failid puhastatakse ning faili kontrolli staatuse kohta saadetakse info tagasi rakendusele [1].

3.2 Nõuete määramine

Nõuded loodavale rakendusele selgitati välja läbi vestluse rakenduse haldajaga ning olemasoleva rakenduse analüüsi põhjal. Nõuded on kirjeldatud, jaotatuna funktsionaalseteks ja mittefunktsionaalseteks nõueteks.

3.2.1 Funktsionaalsed nõuded

Järgnevalt on toodud rakenduse funktsionaalsed nõuded.

- Loodav rakendus peab võtma kasutajalt vastu faile, saatma need failihoidlasse ja kontrolli tulemuse edastama rakendusele, mille jaoks neid faile saadeti.
- Loodav rakendus peab faili kontrolli tulemused edastama faili saatnud kliendile.

- Loodav rakendus peab pakkuma eraldi keskkonda, et kasutaja saaks saata faile teisest seadmest.
- Teise seadme vaatele peab olema autentimata ligipääs.
- Teise seadme vaade peab olema ühekordselt ligipääsetav.
- Algses klientrakenduses peab olema näha, et failid on lisatud.
- Iga liidestatud rakendus peab saama kehtestada, kui palju faile peab olema võimalik saata.
- Iga liidestatud rakendus peab saama kehtestada, mis tüüpi faile on võimalik saata.
- Loodav rakendus peab kontrollima, et ei saadetaks rohkem faile, kui mahupiirang ette näeb.
- Loodav rakendus peab kontrollima, et saadetaks ainult lubatud failitüüpe.

3.2.2 Mittefunktsionaalsed nõuded

Järgnevalt on välja toodud rakenduse mittefunktsionaalsed nõuded.

- Loodavat rakendust peab olema võimalik käitada mitme instantsina.
- Uute teenust kasutavate rakenduste liidestamine peab toimuma dünaamiliselt, see tähendab, ilma teenust osutavat rakendust peatamata ja selle koodi muutmata.
- Loodava rakenduse ja teenust kasutava rakenduse vaheline suhtlus peab olema sünkroonne.
- Teisest seadmest laetud faili info peaks jõudma originaalse kliendini operatiivselt.
- Faile võetakse vastu ainult määratud aja jooksul.
- Loodav rakendus ei tohi edastada telliva rakendusega seotud andmeid mobiili kliendile.

3.3 Arhitektuur

Välja pakutud uue lahenduse näol on tegu uue mikroteenusega, mille peamiseks ülesandeks saab mingi muu rakenduse, antud juhul süüteoavaldust vastu võtva rakenduse jaoks failide üles laadimise haldamine. Uus mikroteenus vastutab selle eest, et saadetavad failid vastu võtta ja edastada faile hoiustavale ja kontrollivale

MetaDefender Vault-ile ning saades sellelt failide kontrolli tulemuse tagasi, edastab selle info algsele rakendusele. See võimaldab pakkuda kasutajale kasutamiseks ka iseseisvat veebirakendust muust seadmest failide üles laadimiseks. Kuna teise seadme vaade peab olema ühekordselt ligipääsetav, on sellele ligipääsu kontrolli rakendamiseks kõige lihtsam piirata seda samuti selle mikroteenuse kaudu. Klientrakendused suhtlevad uue teenusega peamiselt REST API kaudu. REST ehk *representational state transfer* on tihti rakendusliidestes ehk API-des kasutatav arhitektuur, mida iseloomustab selgelt määratletud päringute tegemine ja ressursikesksus, mis võimaldab päringuid kasutada teise osapoole olekut teadmata [2].

Välja pakutud lahenduse rakendamiseks kasutatakse mikroteenuse arhitektuuri. Mikroteenuse arhitektuur on arhitektuuriline stiil, milles rakendus ehitatakse iseseisvateks osadeks ehk teenusteks jagatuna. Iga selline osa on iseseisvalt kasutusele võetav, nõrgalt seotud ja organiseeritud mingi kindla võimekuse ümber [3]. See võimaldab antud lahendust kergemini teiste, sarnase vajadusega rakendusega siduda. Teine aspekt, mida selline arhitektuur aitab saavutada, on see, et loodav rakendus haldab ise autentimist põhirakendusega, mistõttu ei ole vaja klientidele saata autentimisinfot, mis eriti puudutab just lahenduse mobiilset klientrakendust.

Kasutajale antakse ligipääs mobiilivaatest failide üles laadimiseks ühekordse lingi kaudu, mida kasutajale kuvatakse ruutkoodi ehk QR-koodi kujul. QR-kood kasutajale näitamiseks genereeritakse jooksvalt klientrakenduses. Link viitab samuti mikroteenusele, mis serveerib kasutajale mobiilivaate ja kontrollib vaate avamisel, et tegu oleks vaate esmakordse avamisega. Kui vaadet on juba korra avatud antud lingilt, ei saada server vaatele API võtit, mille kaudu faile üles laadida, piirates sellega välist juurdepääsu failide üles laadimisele.

Planeeritav rakendus peab hoiustama peamiselt ainult andmeid erinevate vormide kohta, millega seotud faile ta vastu võtab ja edastab. Kuna neid andmeid peab hoidma ainult nii kaua, kui failide saatmine on lubatud, ja need on lihtsal kujul, piisaks üldjuhul nende andmete mälus hoidmisest. Samas peab antud rakendus olema skaleeritav mitme instantsi peale, mis tähendab, et kehtivad andmed peavad olema kekselt kättesaadavad. Seega peab kasutama andmete hoiustamiseks andmebaasi.

3.3.1 Klientrakenduse oleku uuendamine

Et ühe avaldusega seotud faile võib jõuda serverrakendusse kahest klientrakendusest, peab protsessi juhtivas klientrakenduses olema võimalik saada ülevaade serverisse jõudnud failidest. Selle saavutamiseks on kaks võimalikku meetodit: pärida rakenduselt aeg-ajalt failide seisukorda, või anda rakendusele võimekus saata infot uute failide kohta ise klientrakendusele. Esimese meetodi rakendamiseks on kaks võimalust: seisu pärimine või pikkade päringute kasutamine mudatuste pärimisel. Seisu pärimine on kõige lihtsam: serverrakendusest päritakse kas „värskenda“ nupule vajutades või perioodiliselt avalduse failide seisu. See oleks aga ressursimahukas, kui seisu näiteks tihedamini pärida, eriti automaatse pärimise puhul, kuid isegi manuaalse pärimise puhul tekib probleem, et kasutaja ise peab päringut alustama, ning kui kõikide failide töötlemisega läheb aega, võib enne vastuse saamist jõuda väga palju kordi serverist seisu pärida.

Pikki päringuid kasutades teeks klient serverile päringu, mida hoitakse avatuna seni, kuni serverrakendusel on saata kliendile uut infot. Seejärel sooritatakse uus selline pikk päring. See on ressursside suhtes tõhusam, kui suure hulga üksikute päringute tegemine, sest päringuid tehakse vähem ja vastused võivad olla samuti väiksemad, sisaldades ainult uute failide infot. Samas on sellel probleem: ka pikad päringud aeguvad ja serverrakendus peab kuidagi arvet pidama muutuste üle, mida on kliendile juba saadetud ning mida mitte. Esiteks peab klient sooritama uue päringu aegajalt, sest ka pikkadel päringutel on ajapiirang, üldiselt 100 kuni 300 sekundit [4]. Teiseks, serveri poolt vastuse saatmisest selle kliendini jõudmine ja siis kliendil uue päringu tegemine võtab samuti aega. Seega peab serverrakendus ka kuidagi pidama arvet selle üle, mis on kliendile juba saadetud ja mis mitte. Selleks peab kas klient oma seisu saatma tagasi serverile või peab server vahepeal tulevad uuendused panema mingisse järjekorda või kuidagi muidu neid talletama, et teada, mida ei ole klientrakendusele veel saadetud päringute vahepeal.

Teise meetodi puhul saaks teenusrakendus ise klientrakenduse seisu uuendada. Selle rakendamiseks on peamine lahendus veebisokli kasutamine. Veebisokkel on tehnoloogia, mis võimaldab luua kahesuunalise suhtlussessiooni kliendi ja serveri vahel. Selle kaudu on võimalik saata serverile sõnumeid ja serveril saata sõnumeid kliendile ilma, et oleks vaja serverilt vastust pärida [5]. Veebisoklit kasutades on võimalik

klientrakenduse panna kuulama või tellima enda instantsiga seotud sõnumeid, mida serverrakendus saab ise siis sündmusepõhiselt klientrakendusele saata. Veebisokkel võimaldab kasutaja jaoks ühtlasemalt infot saada, ning serverrakendusel lihtsamalt kontrollida, millal infot saata. Kuna veebisoklit toetavad kõik modernsed veebilehitsejad, ei ole selle kasutamisel erilisi puudujääke võrreldes pikkade päringutega, ning kuna see on operatiivsem, kui pikad päringud, on see kõige sobivam lahendus loodava rakenduse jaoks.

3.3.2 Suhtlus serverrakenduste vahel

Loodava lahenduse serverrakenduse sidumiseks telliva serverrakendusega kasutatakse RPC ehk *Remote Procedure Call* arhitektuuri. Tegu on arhitektuuriga, mis on variatsioon üldisemast klient-server mudelist. Teenust kasutav ehk klientrakendus teeb sünkroonse päringu teenust osutavale ehk serverrakendusele, mis teostab mingi operatsiooni või protseduuri ning saadab tulemuse tagasi klientrakendusele. Kuna tegu on sünkroonse päringuga, ootab teenust kasutava rakenduse töö enne jätkamist päringu vastuseni [6]. RPC on erinev REST arhitektuurist peamiselt selle poolest, et REST on disainilt ressursikeskne, samas kui RPC on suunatud rohkem operatsioonide ja protseduuride peale. Lisaks sellele on RPC peamiselt sünkroonne protokoll, kus kontroll vaheldub rakenduste vahel [7]. RPC sünkroonne olemus teeb selle hästi sobivaks olukordades, kus on mõistlik kasutada klient-server mudelit, kuid päringuid on vaja töödelda sünkroonselt ja ilma, et ükski päring kaduma läheks, näiteks suure koormuse tõttu.

RPC arhitektuuri kasutamiseks ja veendumaks, et saadetavad sõnumid ülekoormuse tõttu kaduma ei läheks, võetakse loodava serverrakenduse ja seda kasutava serverrakenduse vaheliseks suhtluseks kasutusele sõnumivahendaja.

3.4 Tehnoloogia valik

Loodava rakenduse kasutatavate tehnoloogiate valik hõlmab endas rakenduse teenuse loomiseks kasutatava keele ja raamistiku valikut ning klientrakenduse loomiseks kasutatava raamistiku valikut. Olulised aspektid laheduse jaoks tehnoloogia valikul on autori kogemused tehnoloogiaga ja juba kasutusel olevad tehnoloogiad. Lisaks sellele

on oluline, mida ja kuidas konkreetne tehnoloogia mingi rakenduse aspekti loomisel lihtsamaks või tõhusamaks teeb, ehk mida see raamistik pakub.

3.4.1 Teenusepoolse programmeerimiskeele valik

Uue teenusepoolse rakenduse arendamiseks kasutatava keele valik sõltub peamiselt kahest aspektist: autori kogemus programmeerimiskeelega ning olemasolevate komponentide keel ning nendes kasutatava keele tase.

Uuendatav vormide haldamise serverrakendus on kirjutatud Java programmeerimiskeeles, ning kasutab failihoidla Metadefender Vault-iga suhtlemiseks samuti Java keeles kirjutatud teeki. Mainitud teek on kirjutatud Java versioon 11-le.

Uue rakenduse kirjutamiseks valitakse seega Java, sest autoril on selle keele kasutamisel kogemus olemas, ning olemasolevad rakendused kasutavad samuti Java keelt. Kuna varasemalt rakendatud koodis kasutatakse failihoidlaga suhtlemisel ettevõtte sisest teeki, mis on kirjutatud Java 11-s, saab projektis kasutatavaks keeleks Java versioon 11. Kuigi hooldatavuse kaalutlustel ja pikas perspektiivis oleks õigem kasutada keele hilisemat versiooni, eriti, kuna mainitud Java versiooni aktiivne tugi lõpetatakse 30. septembril 2023, ning turvauuendused lõppevad aastal 2026, ei ole selle värskendamine diplomitöö piiratud ajaraamis õigustatud [8].

3.4.2 Teenusepoolsed raamistikud

Arvestades, et loodava teenusepoolse rakenduse keeleks on valitud Java, saab rakenduse arendamiseks kasutatava raamistiku valida Java põhiste veebiraamistike hulgast. Põhilised valikud Java veebirakenduste ehitamiseks on Spring ja Jakarta EE.

Järgnevalt on toodud nende raamistike võrdlus.

Jakarta EE – Jakarta EE on hulk spetsifikatsioone ja nende põhiseid koodikomponente, nagu Servlet ja Server Pages, mis võimaldavad ehitada veebirakendusi ja mikroteenuseid. Tegu ei ole otseselt raamistikuga. Jakarta EE on muuhulgas aluseks ka Spring raamistikule, kuid seda kasutades on võimalik ka otse selle komponentidega ehitada komplekssеid mitme kihilisi veebirakendusi [9].

Spring – Spring on populaarseim Java põhine veebirakenduste ehitamiseks kasutatav raamistik [10]. See on laialt kasutusel ja on toetatud paljude poolt. Suureks tugevuseks

Springi puhul on see, et raamistik ise ja selle tütarprojektid võimaldavad koos hallata nii veebirakenduse turvet, suhtlust andmebaasiga kui ka vajadusel MVC arhitektuuriga luua kasutajaliideseid. See võimaldab lihtsamalt ja ühetaolisemalt ehitada üles uut rakendust.

Loodava rakenduse arendamiseks kasutatakse Java Springi, kuna see raamistik võimaldab lihtsamini ehitada uut veebirakendust. Lisaks sellele on rakendust haldavas asutuses mitmetes rakendustes juba kasutusel Java Spring, mille tõttu lihtsustab uute rakendsute selle peale üles ehitamine lihtsustada rakenduse hooldamist ja täiendamist hiljem.

3.4.3 Kliendipoolne tehnoloogia

Töö raames fookuses olev avalduse vorm, mille täiendamisega tegeletakse, on ehitatud kasutades Vue raamistiku versioon kahte. Seega peab kasutama ka vormile juurde arendatava funktsionaalsuse jaoks Vuejs versioon 2.

Loodaval rakendusel on vaja ka eraldi klientrakendust, et võimaldada kasutajal oma teise seadme, näiteks mobiiliga, faile saata. Seda klientrakendust hakatakse serveerima otse serverrakendusest, et oleks lihtsam kontrollida kasutajate ligipääsu rakendusele.

Brauseripõhise klientrakenduse arendamiseks on võimalik valida mitme klientrakenduste raamistike vahel. Populaarseimad klientrakenduste raamistikud aastal 2022 olid ReactJs, JQuery, Angular ja VueJs [11]. Järgnevalt on toodud nende raamistike võrdlus.

ReactJs – ReactJs on avatud lähtekoodiga teek kasutajaliideste ehitamiseks. ReactJS komponendid on funktsioonid, mille sisendiks on andmed ja väljundi põhjal joonistatakse kasutajaliides. Reacti põhjal on loodud raamistikud nagu Next.Js ja Remix, mis võimaldavad ehitada kogu rakenduse kasutades React arhitektuuri [12].

JQuery – Jquery on väike ja kiire avatud lähtekoodiga teek, mis võimaldab hallata lihtsamini sündmusi ning lihtsustab dokumendi manipulatsiooni [13]. Lisaks üldisele teegile pakutakse ka kasutajaliidese ja ühiktestimise lihtsustamiseks mõeldud teeke. JQuery teeke on võimalik kasutada nii eraldi kui ka koos teiste teekide või raamistikega veebirakenduste loomiseks.

Angular – Angular on avatud lähtekoodiga veebirakenduste ehitamise raamistik. Angular kasutab komponendipõhist arhitektuuri. Selle komponendid koosnevad mallist ja komponendi loogikat ja väärtusi kirjeldavast JavaScripti objektist. Angular toetab andmete sidumist malli ja Javascripti loogika vahel, ehk peab arvestust andmete muutuste üle ja uuendab selle põhjal väärtusi mallis [14].

Vue.Js – Vue avatud lähtekoodiga on kasutajaliideste ehitamise raamistik [15] [16]. VueJs üheks eripäraks on niinimetatud ühe faili komponendid, ehk üks Vue komponent on terviklikult ühes failis, nii HTML, CSS kui ka JavaScripti osa. VueJs võimaldab deklaratiivselt kirjeldada lehe struktuuri süntaksis, mis laiendab baas HTML-i süntaksi. Lisaks sellele peab VueJs arvet oleku muutuste üle ja uuendab vastavalt vajadusele dokumenti [16]. VueJs-i võimekust aitavad tõsta ka suur hulk ametlikke lisateeke, näiteks olekuhaldus teek Pinia ja Vue Router, mis võimaldab lihtsalt luua üheleherakendusi.

Kuigi on tegu eraldi klientrakendusega, siis et pikemas perspektiivis lihtsustada rakenduse hooldamist, valitakse loodava lahenduse klientrakenduse loomiseks Vue.js raamistiku versioon 3. Lisaks on tänu VueJs-i lisateekidele selles mõõdus väikese üheleherakenduse loomine väga ühtlane. Muuhulgas on tegu raamistikuga, millega diplomitöö autoril on võimalikest variantidest kõige rohkem kogemust. Samas, kuna mobiili klientrakendus on eraldiseisev olemasolevast veebirakendusest, on selle arendamisel võimalik kasutada raamistiku uuemat versiooni, tekitamata konflikte olemasoleva koodiga.

3.5 Andmebaasi valik

Andmebaasid jagunevad suures pildis kaheks: relatsioonilased andmebaasid ning mitterelatsioonilised andmebaasid, tihti ka NoSQL andmebaaside nime all [17]. Populaarseimad relatsioonilased andmebaasid on Oracle, MySQL, MS SQL Server ja Postgre. Populaarseimad mitterelatsioonilised andmebaasid on MongoDb, Redis ja Cassandra [18].

Järgnevalt on toodud 3 relatsioonilise ja kahe NoSQL andmebaasi võrlus.

Oracle – Oracle on Oracle Corporation poolt pakutav andmebaas ja andmebaasihaldussüsteem. Tegu on relatsioonilise andmebaasiga. Andmebaasi

kasutamine on kuni kahe instantsi ja piiratud jõudlusega Oracle pilves tasuta, ning on võimalik osta suurema jõudlusega instantse ja rohkem andmebaase. Oracle on kõige populaarsem andmebaas [18].

Postgres – Postgres ehk PostgreSQL on avatud lähtekoodiga objekt-relatsiooniline andmebaas. Postgres toetab 170 featuuri SQL:2016 spetsifikatsioonis nõutud 179st. Postgres on üleüldiselt tuntud oma featuurirohkuse tõttu, toetades muuhulgas protseduurilisi päringuid [19].

MySQL – MySQL on populaarseim avatud lähtekoodiga relatsiooniline andmebaas ja Oracle järel populaarsuselt teine andmebaasi haldussüstem [18]. Võrreldes Postgresiga on MySQL featuurivaesem, kuid on selle võrra üldiselt natuke kiirem.

MongoDb – MongoDb on avatud lähtekoodiga mitterelatsiooniline andmebaas, täpsemalt dokumendiandmebaas. See andmebaas salvestab infot JSON-i laadsetes dokumentides. Kuna andmeid salvestatakse JSON objektidena, ei ole andmete struktuur nii kindlalt ettekirjutatud, kui relatsiooniliste andmebaaside tabelites [20]. Samas on lihtsa kujuga andmete puhul dokumendiandmebaas üldiselt relatsioonilistest andmebaasidest kiirem [17]. Samuti on andmed inimesele üldiselt lihtsamini loetavamad, kuna on tuttaval JSON kujul [20].

Redis – Redis on avatud lähtekoodiga andmestruktuuride hoidla, mida kasutatakse muuhulgas andmebaasi, sõnumivahendaja või vahemäluna. Redis võimaldab hoiustada suur variatsiooni andmestruktuure ja saab töötada nii täielikult mälupõhisena kui ka mitmel erineval viisil andmeid kettale salvestades [21]. Redis toetab vähemalt 53 erinevat keelt või tehnoloogiat ametlike klientteekidega [22].

Kuigi antud rakenduse puhul on andmekuju lihtne ja ühetaoline, siis rakenduse andmete hoiustamiseks valiti relatsiooniline andmebaas, sest töö autor on sellega tuttavam. Relatsiooniliste andmebaaside hulgast sai valitud Postgres andmebaas, sest tegu on avatud lähtekoodiga andmebaasiga, selle kasutamine on tasuta ja see on juba täiendatava rakenduse juures kasutusel.

3.6 Sõnumivahendaja valik

Loodav rakendus peab suhtlema autenditud rakendusega ning saatma sellele jooksvalt infot kasutaja lisatud failide kohta. See aga tekitab olukorra, kus teise rakenduse kaudu on võimalik tellivale rakendusele saata rohkem päringuid, kui see vastu võtta suudab. Lisaks on oluline, et suure kasutuse koormuse või ajutiste seisangute tõttu antud info kaduma ei läheks. Seepärast nende kahe serverrakenduse vahelise suhtluse haldamiseks ja puhverdamiseks võetakse kasutusele sõnumivahendaja.

Sõnumivahendaja on tarkvara, mis valideerib, hoiustab ja edastab sõnumeid rakenduse osade või erinevate rakenduste vahel. Selleks kasutab see sõnumi järjekordasid, mis hoiustavad ja järjestavad sõnumeid, ning edastab neid alles siis, kui neid kasutav rakendus neid vastu võtta saab. Selle tulemusena ei lähe sõnumid kaotsi, kui tarbiv rakendus neid kõiki korraga vastu võtta ei suuda [23].

Populaarsed sõnumivahendajad on näiteks Memphis, RabbitMQ, Apache Kafka, Apache ActiveMQ, ZeroMQ [24]. Järgnevalt on välja toodud nende sõnumivahendajate võrdlus.

Memphis – Memphis on avatud lähtekoodiga sõnumivahendaja, mis kasutab TCP-põhiseid protokolle ja HTTP protokolli. Seda on võimalik kasutada klastrites, et suudaks käidelda suuremat nõudlust. Sõnumivahendajat on võimalik administreerida nii GUI kui CLI käidu. See sõnumiahendaja võimaldab kasutada peamiselt avaldaja-tellija topoloogiat. Lisaks sellele pakub see järjekordade dubleerimise ja põhjalikku logimise funktsionaalsusi [25].

RabbitMQ – RabbitMQ on avatud lähtekoodiga sõnumivahendaja, mis võimaldab kasutada laia valikut protokolle. Rabbit toetab kõiki enamlevinud programmeerimiskeeli ja tehnoloogiaid, nagu Java, PHP, .Net ja Javascript [26]. Sõnumivahendaja administreerimiseks on nii CLI [27], GUI kui ka HTTP REST liidestus [28]. See vahendaja võimaldab kasutada peamiselt vahetuse ja järjekorra põhist topoloogiat, kuid ka avaldaja-tellija topoloogiat [29]. Sõnumivahendajat on võimalik kasutada klastrites ja see võimaldab ka järjekordade dubleerimist. Rakenduse kasutamine on tasuta, kuid on võimalik maksta täiendava toe eest.

Apache Kafka – Kafka on avatud lähtekoodiga sõnumivahendaja, mis on mõeldud peamiselt voohalduseks. Kafkaga on peamiselt võimalik kasutada avaldaja-tellija topoloogiat, et edastada või hoiustada sündmuste infot. Sellest tulenevalt on Kafkasse sõnumite kirjutamine sündmuste kujul. Kafka on suuremahuline ja kiire, kuid sellest tulenenvalt võib see lihtsamate kasutuste puhul liiga keeruline olla [30].

Apache ActiveMQ – ActiveMQ on avatud lähtekoodiga sõnumivahendaja, mis toetab 8 keelt otse ning võimaldab läbi Stomp protokolli implementeerida uusi kliente mitmetele teistele keeltele ja platvormidele. Kuna see on originaalselt kirjutatud Java keeles, on sellel väga hea liidestus ja tugi Java ja Spring-iga [31]. See on peamiselt disainitud klient-server sõnumivahetuseks.

ZeroMQ- ZeroMQ ei ole otseselt sõnumivahendaja, ehk selle puhul ei eksisteeri vaherakendust, mis sõnumeid haldaks ja vahendaks. Selle asemel kasutatakse ZeroMQ puhul sokleid ja madala taseme teek. See lahendus on kasutatav vähemalt 15 erinevas programmeerimiskeeles, kuid on teiste sõnumivahendajatega võrreldes ebastandardne lahendus [32].

Loodavas rakenduses võetakse kasutusele RabbitMQ, sest sellel on laialdane toetus erinevatele keeltele ja rakendustele ning selle kasutuse eest ei pea maksma. Lisaks eeltoodud punktidele on RabbitMQ võimeline töötama klastrites ehk võimaldab lihtsamini suure kasutuse puhul sõnumivahendaja kihi töövõimekust tõsta, mis on kasulik, kui antud lahendust peaks kasutusele võetama rohkemates rakendustes ja on vaja suuremat jõudlust. Selle vahetuse ja järjekorra topoloogia võimaldab ka kergemini hallata ja juhtida eri rakenduste ja nende instantsidega seotud sõnumeid, lihtsustades nende sõnumite konkreetsele rakendusele suunamist.

3.7 Analüüsi kokkuvõte

Analüüsis käsitleti loodava lahenduse funktsionaalseid ja mittefunktsionaalseid nõudeid. Olemasoleva teenuse ja uue lahenduse arhitektuuri analüüsi põhjal ilmnes, et uus lahendus saab olema mikroteenus, mis kasutab vormi haldava rakendusega suhtlemiseks sõnumivahendajat ja põhikliendi oleku uuendamiseks veebisoklit (Lisa 4).

Loodava lahenduse teenusrakenduse kirjutamiseks valiti Java programmeerimiskeele versioon 11 ja Spring raamistik, nii seetõttu, et see on juba kasutusel avaldust haldavas

rakenduses, kui ka seepärast, mida Spring võimaldab. Klientrakenduse loomiseks valiti VueJs raamistiku versioon 3. Rakenduse andmete hoiustamiseks valiti PostgreSQL andmebaas. Loodava mikroteenuse ja seda kasutava teenuse vahelise suhtluse haldamiseks ja puhverdamiseks valiti sõnumivahendaja RabbitMQ.

4 Rakenduse arendus

Rakenduse arendus on jagatud nelja ossa: uue teenuse arendus, olemasoleva teenuse täiendamine, uue klientrakenduse arendus ning olemasoleva klientrakenduse täiendamine. Mõlemad teenusrakendused on ehitatud Spring raamistiku peale. Uus klientrakendus on ehitatud, kasutades VueJs raamistiku versiooni 3 ja olemasolev klientrakendus kasutab VueJs versiooni 2. Esimene peatükk kirjeldab uue veebiteenuse arendust ja sõnumivahendaja konfiguratsiooni. Teine peatükk kirjeldab muudatusi ja täiendusi olemasolevale vormi haldavale rakendusele. Kolmas peatükk kirjeldab uue klientrakenduse arendust ning neljas muudatusi olemasolevale klientrakendusele, kaasa arvatud veebisokli kasutust ja failide saatmist.

4.1 Uue veebiteenuse arendus

Uue veebiteenus arenduseks luuakse uus Spring Boot projekt, kasutades selleks Spring Initializr-it aadressilt *start.spring.io*. Loodud projekti kasutatakse seejärel põhjana faile vastu võtva rakenduse loomiseks.

Rakenduse klassid on jaotatud parema halduse mõttes funktsionaalsuse põhjal järgnevatesse kaustadesse:

- *Config* – rakenduse konfiguratsioonifailid. Andmebaasiga seotud konfiguratsioonifailid on selle kausta alamkaustas *db*.
- *Controllers* – REST API kontrollerid.
- *Domain* – Andmebaasi objekti klass ja andmebaasiga suhtlemise loogikat sisaldav klass.
- *Services* – erinevad äriloogikat sisaldavad klassid. Need klassid sisaldavad meetodeid, mida kontrollerid kasutada saavad.
- *Util* – üldist korduvkasutatavat loogikat sisaldavad klassid.

4.1.1 Konfiguratsioon

Rakenduse seadistust määratakse läbi konfiguratsiooniklasside ja *application.properties* faili läbi. Konfiguratsiooniklassid asuvad kaustas *config* ja on rakendusele äratuntavaks tehtud *@Confifuration* annotatsiooni abil. Konfiguratsiooni kaustas asuvad veebirakenduse turbe konfiguratsioon, sõnumivahendaja konfiguratsioon, veebisokli konfiguratsioon ja alamkaust *db*, milles sisalduvad andmebaasi konfiguratsioonifailid.

Rakenduse arenduse käigus kiiremaks ja lihtsamaks käitamiseks on andmebaasi konfiguratsioonid defineeritud eraldi arenduse profiilile ja tootmise profiilile kasutades Spring-i *@Profile* annotatsiooni. Arenduse jaoks mõeldud profiiliga kasutatakse mälupõhist HyperSQL andmebaasi ja põhjalikumat päringute logimist, mis lihtsustab silumist. Tootmiseks mõeldud profiiliga kasutatakse eelnevas peatükis valitud PostgreSQL andmebaasi.

Veebisokli konfiguratsioonis seadistatakse veebisoklile vahendaja, mis kasutab baasaadressi */queue*, kuhu edastada id põhjal klientrakendusele infot saadud failide kohta. Lisaks registreeritakse */websocket* otspunkt, et seda oleks võimalik kasutada algselt veebisokliga ühenduse loomisel nii standardse veebisokli ühendusega kui ka kasutades SockJs teeki. Lubades mõlemad ühenduse meetodid ei ole hiljem klientrakenduste lisamisel piiratud kasutama ühte ühendamise meetodit. Veebisokli kasutust on lähemalt selgitatud klientrakenduse arendust käsitlevas peatükis.

Sõnumivahendaja konfiguratsioonis kirjeldatakse ära ja konfigureeritakse ära ühendus sõnumivahendajaga, rakenduse operatiivsete sõnumite edastamiseks kasutatav sõnumijärjekord ning seda haldav sõnumivahetus (Lisa 2). Lisaks deklareeritakse siin sõnumite saatmiseks kasutatav mall, millele määratakse *application.properties* failist loetud vastuste ootamise aeg. Viimaks on siin defineeritud eelmainitud malli hiljem uuendamiseks kasutatav administraatorobjekt. Sõnumivahendaja täpsemat kasutust selgitatakse hilisemas sõnumivahendaja peatükis.

4.1.2 REST API ja kontrollerid

Loodava rakenduse REST API-l on kaks peamist funktsionaalsust: võtta vastu klientrakendustest saadetud faile ja verifitseerida erinevate klientide õigusi rakendusele faile saata, sealhulgas kontrollida klientide ligipääsu mobiilivaate klientrakendusele.

4.1.3 Failide vastuvõtmine ja töötlemine

Failide vastuvõtmise eest vastutab klass *FileController*. Antud klassil on põhieesmärgiks võtta vastu faile sisaldavaid *multipart* ehk mitmeosalised POST päringuid. Peale päringu vastuvõtmist on esimene ülesanne verifitseerida päring ja viia kokku see konkreetse telliva rakenduse instantsiga. Päringu identifitseerimiseks edastatakse päises klienti identifitseeriv id. Eduka verifikatsiooni korral suunab kontroller saadetud faili koos faili omava kliendi andmetega faili töötlemiseks ja edastamiseks teenusklassile *FileService*.

Failide töötlemise loogikat haldab klass *FileService*. Sellel klassil on mitu üleseannet: eraldada verifitseeritud päringust fail ja see valideerida, saata fail kontrollimiseks failihoidlasse, saata failihoidla kontrolli tulemus algsele rakendusele ning pidada arvet vastuvõetud failide kogumahu üle. Eelnevalt pidas failide mahu ja lubatud failitüüpide üle arvestust faile saatev klientrakendus, kuid uue rakenduse kasutusele võtu järel on klientrakendusi, mis faile saata võivad mitu, ning seega lubatud kogumahu ja failitüüpide üle peab pidama keskset arvestust. Kui faili üldised näitajad vastavad kehtestatud piirangutele, saadetakse fail seejärel failihoidlasse, kasutades selleks failihoidlaga suhtlemist haldavat eraldi teeki. Peale failihoidlast vastuse saamist saadetakse faili info tellivale rakendusele läbi sõnumivahendaja. Lõpuks, kui faili info on edastatud tellivale rakendusele, saadetakse faili saatnud klientrakendusele töödeldud faili kohta vastus. Lisaks algsele POST päringule vastamisele saadetakse faili töötlemise tulemused veebisokli kaudu ka põhilisele klientrakendusele.

4.1.4 Klientrakenduse ligipääsu haldamine

Mobiili klientrakenduse serveerib kasutajale samuti teenusrakendus. Selle saavutamiseks kasutatakse Springi sisseehitatud *ErrorController*-it. Rakenduse tavaline käitumine navigatsiooni korral on saata kasutajale üldine lehe mitte leidmise sõnum. Kuna serveeritav klientrakendus on aga üheleherakendus, siis et kasutajale õigesti seda kuvada, kirjutatakse üle */error* navigatsioon. Kui kasutaja teeb GET päringu, mida ei ole mõne funktsiooniga seotud, saab ta vastu üheleherakenduse. See ühelerakendus teeb juhul, kui päringu parameetrid on õiged ja kujul *hostinimi/keel/ühekordne id*, päringu rakenduse *ClientController*'ile. Õigete päringu parameetrite korral saab klientrakendus

ClientController'ilt endale failide saatmiseks API võtme. Selle sama päringu tulemusena märgitakse ka ära, et samade andmete põhjal enam API võtit välja ei anta. Juhul kui aga algsed parameetrid on valed, valel kujul või enam selle id kohta võtit ei saa, näidatakse kasutajale üldist teadet, et lehte ei leitud. Selle lahenduse tulemusel on ainult esimesel väljastatud lingi kasutajal võimalik avada konkreetse avaldusega seotud mobiilivaade ja selle kaudu faile saata.

4.1.5 Sõnumivahendaja

Sõnumivahendaja on loodavas rakenduses kasutusel uue faile vastuvõtva rakenduse ja täiendatava rakenduse vahelise infovahetuse vahendamiseks, kontrollimiseks ja puhverdamiseks. Eelnevas analüüsi peatükis valiti kasutavaks sõnumivahendajaks RabbitMQ. Sõnumivahendajaga ühendusega seotud konfiguratsioon on kirjeldatud eelnevalt „Konfiguratsioon" peatükis.

Sõnumivahendaja vahendab rakenduste vahel kaht tüüpi sõnumeid: operatiivsed ehk loodava rakenduse tööd juhtivad sõnumid ning failide info ja failide andmeid sisaldavad sõnumid. Operatiivsed sõnumeid on 6 tüüpi: registreeringute tellimused, registreeringute kinnitused, kustutussõnumid, lõpetussõnumid, üldised kinnitussõnumid ja veasõnumid. Kõik operatiivsete sõnumite tüübid sisaldavad välja, mis sisaldab infot sõnumi liigi kohta. Lisaks sellele on igal sõnumitüübil oma spetsiifilised väljad, mis sisaldavad seda tüüpi sõnumeid edastatavat infot. Järgnevalt on toodud iga sõnumitüübi spetsiifilisem kirjeldus:

- Registreeringute tellimused – sõnumid, milles edastatakse andmed telliva rakenduse instantsi kohta. Sõnum on tüüpi *REGISTRATION_REQUEST* ja sisaldab tunnust, millega tellivas rakenduses failid õigele kasutajale määrata, telliva rakenduse failide vastuvõtmiseks kasutatava sõnumivahetuse nime, lubatud failide kogusuurust, praeguseks juba kasutatud suurust, lubatud failitüüpe ja aega, kui kaua registreering peaks kehtima (Lisa 3).
- Registreeringute kinnitused – sõnumid, milles saadetakse tellivale rakendusele info, kuidas saab hakata vastuvõtvale rakendusele saatma faile. Sõnum on tüüpi *REGISTRATION_REQUEST* ja sisaldab mobiilikliendi aadressi, failide API aadressi, API võtit, telliva rakenduse tunnust, veebisokli aadressi ning aega,

milleni registreering on kehtiv (Lisa 3). Sõnum saadetakse vastusena eelnevatele registreeringute tellimustele.

- Kustutussõnumid – sõnumid, milles saadetakse infot failide kustutamise kohta, et oleks võimalik pidada arvet kasutatud failimahu kohta. Sõnum on tüüpi *DELETE_REQUEST* ja sisaldab telliva rakenduse tunnust ning kustutatud faili suurust.
- Lõpetussõnumid - sõnumid, millega antakse teada telliva rakenduse töö lõpust, kui täidetav vorm kinnitatakse ja ära saadetakse. Sõnum on tüüpi *TERMINATION* ja sisaldab telliva rakenduse tunnust.
- Üldised kinnitussõnumid – sõnumid, mida saadetakse vastusena näiteks eduka faili kustutamise või lõpetuse puhul. Sõnum on tüüpi *SUCCESS* ja sisaldab välju olekukoodide ja sõnumi lisamiseks.
- Veasõnumid - sõnumid, mida saadetakse vastusena vea ilmnemise puhul. Sõnum on tüüpi *ERROR* ja sisaldab välja selgitava sõnumi lisamiseks.

Kõik sõnumitüüpe kirjeldavad klassid põhinevad ühel abstraktsel klassil, mis määrab, et iga klass sisaldaks välja *type*, kuhu märgitakse sõnumi tüüp ja mille järgi sõnumi saamisel erinevaid sõnumitüüpe tuvastatakse. Loodav rakendus kasutab aktiivset kuulajat, et saada operatiivsete sõnumite järjekorrast sõnumeid, mille sisu sõne kujule tõlgendatakse. Sealt loetakse välja sõnumi liik, mille põhjal deserialiseeritakse sõnum info töötlemiseks sobiva klassi objekti kujule.

Kõigilt eri rakendustelt pärinevad operatiivsed sõnumid saadetakse ühe vahetuse kaudu ühte sõnumijärjekorda, millest siis loodav rakendus järjest võtab töötlemiseks sõnumeid ja siis nende vastused saadab sama sõnumivahetuse kaudu automaatselt loodud vastuste järjekorda.

Failide tellivale rakendusele saatmiseks lastakse iga telliva rakenduse registreerimisega määrata, mis nimega sõnumivahetust kasutada saadud failide info saatmiseks. Ühe sõnumivahetuse ja sõnumijärjekorra külge võib olla ühendatud mitu telliva rakenduse instantsi või ainult üks. Kui vastuvõetud fail edastatakse tellivale rakendusele edasi saatmiseks, loetakse andmebaasist kliendi realt soovitud vahetuse nimi ja deklareeritakse malli administraatorobjekti kasutades vastav sõnumivahetus ja selle külge käiv sõnumijärjekord. Sellesse sõnumivahetusse saadetakse faili puudutav info.

Vastuvõetud faili info saadetakse tellivale rakendusele *StorageFileInfo* objekti formaadis (Joonis 1). Selles klassis on telliva rakenduse tunnus, faili info ja failihoidla kontrolli tulemusena saadud vastus. Faili infot edastatakse klassi *StorageFileDto* sees. See klass sisaldab faili originaalset nime, faili uut nime failihoidlas, faili kasutustüüpi, faili suurust, faili koodi failihoidlas ja välja faili sisu jaoks.

```
@AllArgsConstructor
@Data
public class StorageFileInfo {
    private MetaDefenderVaultFileStatus fileStatus;
    private StorageFileDto file;
    private String token;
}
```

Joonis 1. Vastuvõetud faili infot sisaldava klassi *StorageFileInfo* programmikood.

4.1.6 Andmebaas

Et oleks võimalik tellivale rakendusele faile edastada ja et need seal õige instantsi külge ühendada, peab telliva rakenduse kohta hoiustama andmeid andmebaasis. Eelneva peatüki analüüsis sai rakenduse juures kasutatavaks andmebaasiks valitud PostgreSQL. Andmebaasi konfiguratsiooni on kirjeldatud eelnevalt peatükis „Konfiguratsioon“. Andmebaasis hoitakse andmeid ainult ühes tabelis.

Andmebaasiga suhtluseks kasutatakse Jakarta Persistence API-t ehk JPA-d. JPA on teek, mis võimaldab lihtsamalt rakendusel andmeid lihtsamini pärida ja salvestada [33]. Lisaks sellele võimaldab JPA standardselt defineerida tabeleid ja olemite vahelisi suhteid, sõltumata kasutatavast andmebaasist. JPA kasutus on teostatud *ClientRepository* klassi kaudu. See klass on liideseks JPA kasutamisel ja kõik JPA-d kasutavad funktsioonid on selles klassis. Nii andmebaasiga seotud koodi ülejäänust eraldamine aitab hoida koodibaasi selgemana ja vähendada komplekssust [34].

Telliva rakenduse kohta käivaid andmeid sisaldavat tabelit kirjeldab *Client* klass (Joonis 2). Tabeli primaarvõtmeks on *id*, mis genereeritakse automaatselt olemi andmebaasi salvestamisel UUID põhimõttel. Teiseks väljaks on samal põhimõttel genereeritud mobiili vaate võti *frontId*, mis genereeritakse rakenduse poolt olemi loomise järel.

Lisaks *id* väljale on olemi salvestamiseks vajalik väli tellivas rakenduses failide sidumiseks kasutatav tunnus *token.* Väli url on andmete saatmiseks mõeldud sõnumivahetuse tunnus. Väljal *timeToLive* salvestatakse aega, milleni saab saadetud faile vastu võetakse. *AllowedSize* ja *usedSize* on vastavalt maksimaalse lubatud failide koguse ja kasutatud suuruse arvestamiseks. Väli *mobileAccessed* kasutakse kontrollimaks, kas mobiilivaatele on juba API võtit väljastatud, ning väli *extensions* sisaldab lubatud faililaiendite loendit.

```
@Data
@AllArgsConstructor
@NoArgsConstructor(force = true)
@RequiredArgsConstructor
@Entity
public class Client {

    @Id
    @GeneratedValue(generator = "UUID")
    @GenericGenerator(
            name = "UUID",
            strategy = "org.hibernate.id.UUIDGenerator"
    )
    private UUID id;
    private UUID frontId = UUID.randomUUID();
    @NonNull
    private String token;
    private String url;
    private ZonedDateTime timeToLIve;
    private long allowedSize;
    private long usedSize;
    private boolean mobileAccessed = false;
    private String extensions;
}
```

Joonis 2. Telliva rakenduse kohta andmeid säilitava klassi *Client* programmikood.

4.2 Täiendused vormi haldavale rakendusele

Vormi haldavas rakenduses on vaja uue rakenduse kasutamiseks lisada üleslaadimise soovi registreerimise funktsionaalsus ja üleslaadimist haldavalt rakenduselt failide andmete vastuvõtmise võimekus.

Vormi haldavas rakenduses on praegu failide haldamiseks üks kontroller, mis töötleb kolme päringut: GET salvestatud failide lugemiseks, POST failide lisamiseks ja DELETE lisatud failide kustutamiseks. Võttes kasutusele uue lahenduse, ei pea vormi haldav rakendus enam failide lisamise päringuid töötlema. Failide väljastamine ja kustutamine jääb aga endiselt originaalse rakenduse ülesandeks. Seda seetõttu, et reaalsete kasutajate ja täidetavate vormide haldamine on endiselt originaalse rakenduse tööks. Uus rakendus failide üle arvet ei pea.

4.2.1 REST API

Nagu eelnevalt kirjeldatud, tekitab loodava rakenduse kasutuselevõtt vajaduse faile haldav kontroller ümber ehitada. Kontrollerilt saab eemaldada eelnevalt faile vastu võtnud POST otspunkt. Eelnevalt eksisteerinud otspunktidest jäävad alles GET otspunkt, mida kasutatakse vormis salvestatud failide pärimiseks ja DELETE otspunkt, mida kasutatakse failide kustutamiseks. Kontrollerile tuleb juurde lisada veel üks GET otspunkt, mida klientrakendus saab kasutada enne failide saatmist, et pärida failide saatmiseks vajalikud otspunktid ja vormiga sidumiseks kasutatavad võtmed.

Lisatav päringu otspunkt kujutab endas kontrolleri klassile lisatavat *@GetMapping* annotatsiooniga meetodit. Uus meetod võtab parameetritena sõne kujul päisest *token* väljalt vormi esitajat tuvastava tunnuse ning *HttpServletRequest* klassi objekti, mis esindab päringu andmeid. Tunnuse põhjal kontrollitakse, kas vormiga on juba seotud mingeid faile. Kui ei ole, luuakse uus vormi faile hoidev sissekanne andmebaasi ja saadetakse faile haldavale rakendusele sõnum uue vormi registreerimiseks. Konfiguratsiooni põhjal pannakse sõnumile kaasa nimekiri lubatud failitüüpidest ning saadetavate failide kogumahu piirang. Juhul, kui vormiga on aga seotud faile, lisatakse päringule ka nende failide suuruste summa juba kasutatud kogusuuruse väljale. Vastuseks saadud sõnum edastatakse päringu vastusena klientrakendusele.

4.2.2 Failide andmete salvestamine

Uue rakenduse poolt vastuvõetud failide andmed saadetakse peale valideerimist ja failihoidlasse panekut, kasutades sõnumivahendajat, originaalsele vormi haldavale rakendusele. Failide andmete vastuvõtmiseks registreeritakse originaalses rakenduses kuulamisfunktsioon, mis registreeritakse kasutades *@RabbitListener* annotatsiooni ja märkides annotatsioonis ära kasutatava järjekorra nimi (Joonis 3). Funktsiooni sisendiks

on sõnumivahendaja sõnumit kujutav *Message* klassi objekt. Kuulamisfunktsioon eraldab sõnumist selle sisu ja teisendab selle eelnevalt kirjeldatud *StorageFileInfo* kujule. Sealt salvestatakse faili kohta käiv oluline info vormi kohta käiva sissekande juurde andmebaasis.

```
@RabbitListener(queues = "${spring.rabbitmq.transferQueueName}")
public byte[] process(Message message) throws JsonProcessingException {
	return processMessage(message);
}
public byte[] processMessage(Message msg) throws JsonProcessingException {
	var result = fileService.handleUploadInfo(new Gson().fromJson(new
	    String(msg.getBody()), StorageFileInfo.class));
	log.info("Data received from file queue: file {} added",
	    result.getFileServiceFile().getFileName());
	SuccessMessage response = new SuccessMessage(result.getError());
	return objectMapper.writeValueAsString(response).getBytes();
}
```

Joonis 3. Sõnumivahendaja kaudu saabunud sõnumite töötlemine.

4.3 Mobiilivaate lahendus

Kasutades *create-vue* käsurea tööriista, luuakse uus VueJs projekt klientrakenduse arendamiseks (Joonis 4). Tööriista loodud baasprojekt sisaldab näidisvaateid, navigeerimise haldamiseks kasutatava Vue Router-i konfiguratsiooni, arendamiseks kasutatava Vite arendusserveri konfiguratsiooni ning rakenduse baasfaile.

```
Vue.js - The Progressive JavaScript Framework

√ Add TypeScript? ... No / Yes
√ Add JSX Support? ... No / Yes
√ Add Vue Router for Single Page Application development? ... No / Yes
√ Add Pinia for state management? ... No / Yes
√ Add Vitest for Unit Testing? ... No / Yes
√ Add an End-to-End Testing Solution? » No
√ Add ESLint for code quality? ... No / Yes
```

Joonis 4. Uue VueJs projekti loomine kasutades *create-vue* käsurea tööriista.

Projekti loodavate elementide failid asuvad projekti *src* kaustas, kus nad on jaotatud järgmistesse kaustadesse:

- *Assets* – rakenduse välimuse seadistamiseks kasutatavad failid.
- *Components* – rakenduse kasutajaliidese elemente kirjeldavad failid.
- *Router* – navigatsiooni funktsionaalsust pakkuva Vue-Routeri konfiguratsioonifailid.
- *Types* – rakenduse staatilisi andmeid, sealhulgas tõlkeid hoidvad failid.

Loodav klientrakendus on ehituselt lihtne, sest rakendusel on ainult kaks vaadet: failide üles laadimise vaade ja veateate vaade, mida näidatakse vigase navigatsiooni või edutu ligipääsupäringu korral (Joonis 5).

Joonis 5. Mobiili klientrakenduse failide üles laadimise vaade

Loodav klientrakendus kasutab serverrakenduse API-ga suhtlemiseks Axios teeki. Axios on HTTP klient, mis võimaldab standardsemal viisil sooritada HTTP päringuid [35]. See klientrakendus peab tegema 2 tüüpi päringuid: GET päringuid lehele saabudes saatmisõiguse saamiseks ja POST päringuid failide saatmiseks. Faile saadetakse serverrakendusse *multipart* ehk mitmeosaliste päringute kaudu. Et hoida mobiilivaate klient lihtsamana ja klientrakenduse eraldi hoidmiseks ei suhtle see klientrakendus originaalse serverrakendusega. See tähendab, et antud rakendusest ei ole võimalik näha failide seisu või eemaldada vormiga seotud faile. See funktsionaalsus jääb ainult põhi klientrakendusele. Seetõttu ei näidata sellesse rakendusse välja ka detaile failide saatmisel tekkinud vigade kohta. Detailset infot vigade kohta saadetakse teenusrakendusest põhi klientrakendusele, kus seda siis selgelt välja kuvatakse.

4.4 Muudatused olemasolevatele klientrakendustele

Välja pakutud lahenduse kasutuselevõtuga kaasneb failide saatmise protsessi keerulisemaks muutumine kliendi poolelt, mistõttu tuleb uuendada ka olemasolevat klientrakendust. Olemasoleva klientrakenduse vormis kasutatakse failide saatmiseks mõeldud välist komponenti *nuxt-dropzone*, mistõttu ei saa kogu failide saatmisega seotud loogikat otse muuta. Välist komponenti kapseldav komponent on aga iseseisev ja spetsiaalselt välise komponendi kontrollimiseks ehitatud. Seetõttu on võimalik viia sisse kõik vajaminevad muudatused just sellesse komponenti, muutmata seejuures komponendi üldist kasutust veebilehe erinevates vormides. Failide saatmise komponendile tehtavaid muudatusi on 3: failide üleslaadimise alustamise päringu seadistamine, veebisokli seadistamine uute failide jooksvalt kättesaamiseks ja mobiilivaate lingi kasutajale kättesaadavaks tegemine.

Failide üleslaadimise funktsionaalsuse uuendamiseks tuleb rakendada kaks muudatust: üleslaadimise API võtme küsimine ja selle aegumisel selle uuendamine. Uus üleslaetavaid faile vastuvõttev rakendus kasutab nende vormiga sidumiseks unikaalset tunnust, mille järgi vastuvõetud failide andmed õigele rakendusele saadetakse. Selle lahendamiseks seadistatakse komponent nii, et esimese faili saamisel teeb komponent kõigepealt alustamise päringu vormi haldavasse serverrakendusse, saades vastuseks *RegistrationResponce* klassi objekti kujutava JSON-i. Vastuses olevat API aadressi ja võtit kasutakse siis failide üles laadimisel kuni selle aegumiseni. Juhuks, kui faile

saadetakse pärast võtme aegumist, on faile vastu võttev rakendus konfigureeritud failid tagasi lükkama ja vastuseks andma HTTP koodi 404. Sellises olukorras sujuvalt töö jätkamiseks seadistatakse klientrakendus kordama eelnevalt mainitud üleslaadimise alustamise toiminguid, ning peale uute võtmete saamist jätkama failide saatmist.

Teiseks on vaja võimaldada kasutajal pääseda ligi mobiilsest seadmest failide üles laadimiseks kasutatavale klientrakendusele. Kõige lihtsam viis kasutajale aadressi näitamiseks on kasutaja sisendi järel kuvada see sobival kujul failide saatmise komponendi peale. Et lihtsustada kasutajal selle aadressi kasutamist teise seadmega, kuvatakse antud aadress kasutajale ruutkoodi kujul. Ruutkood ehk QR-kood on viis edastada infot masinloetaval kujul, kasutades selleks kahemõõtmelist välja, millel infot kodeeritakse kahte värvi, enamasti musta ja valget, ruutude kujul [36]. Kuna ruutkoodiga saab väga lihtsalt kodeerida teksti, on see hea viis edastada näiteks veebiaadresse. Suurem osa modernseid kaameraga nutiseadmeid omab integreeritud funktsionaalsust QR-koodide lugemiseks ning lisaks on saadaval hulk rakendusi, mis seda integreeritud funktsionaalsuse puudumisel võimaldavad [36]. Ruutkoodi koostamiseks serverrakenduselt saadud lingi põhjal kasutatakse välist teeki *vue-qrcode*. See teek võimaldab lisada rakendusse komponendi, mis rakendab QR-koodi koostamise loogikat. Komponendile antakse sisendiks sõne kujul kodeeritav tekst, antud juhul siis mobiilivaate veebiaadress, ning info loodava koodi välimuse konfigureerimiseks, näiteks suuruse ja värvide muutmiseks. Komponent lisatakse kasutaja sisendi peale kuvatava ülekatte sisse, kust seda saab siis mobiilse seadmega lugeda (Joonis 6).

Joonis 6. Mobiilivaate aadressi kuvamine ruutkoodina.

Uue laheduse kasutuselevõtu järel saadetakse faile serverile kuni kahest seadmest. Lõplik kontroll avalduse üle jääb aga endiselt originaalsele klientrakendusele ja selle failide saatmise komponendile. Juhtivas klientrakenduses failide seisu jooksvalt uuendamiseks võetakse kasutusele veebisokkel. Kuna teisest seadmest saadetud failide kohta saab infot tulla ainult siis, kui kasutaja kasutab mobiilivaadet, siis veebisokkel on vaja avada alles peale seda, kui kasutaja on näinud mobiilivaate linki sisaldavat ruutkoodi. Veebisokli haldamiseks võetakse kasutusele *StompJS*, mis võimaldab veebisokli ühendusega seotud aspekte kergemini kontrollida. Veebisoklist kindla vormiga seotud failide saatmisel kasutatakse avaldamise-tellimise mudelit. See tähendab, et saabunud failide info saadetakse ainult klientidele, mis on konfigureeritud jälgima ühte kindlat kanalit [37]. Iga vorm saab oma failide saatmise API võtme põhjal oma kanali failide kohta teavituste saamiseks. Rakendus hakkab kanalit jälgima peale seda, kui kasutajale on ruutkoodi näidatud. Veebisoklist tulevate failide info põhjal näidatakse kasutajale teisest seadmest saadetud failide kohta käivat infot. Kuna mobiilses klientrakenduses näidatakse ka vigade korral minimaalset infot, kuvatakse tagasi lükatud faile ja nende veateateid kasutajale põhikliendis, kasutades ka selle info edastamiseks veebisoklit (Joonis 7).

Joonis 7. Vigase faili korral kuvatakse kasutajale veateade koos vigase failiga.

5 Hinnang loodud rakendusele

Arenduse käigus loodi mikroteenus, mis haldab failide vastu võtmist süüteoavalduse klientrakendusest, nende kontrollimist ja failihoidlale edastamist. Lisaks sellele võimaldab rakendus kasutajal edastada avaldusse faile ka mobiilist. Tulemust saab hinnata saavutatud funktsionaalsuste ja nõuetele vastavuse põhjal.

Loodud mikrorakendus täidab kõiki analüüsis välja toodud funktsionaalseid nõudeid. Rakenduse kaudu on võimalik üles laadida faile, mis kontrollitakse ja mille andmed edastatakse avalduse rakendusele. Tänu sellele on võimalik pakkuda kasutajale eraldi failide üles laadimist mobiilsest seadmest. Faile vastu võttev rakendus kontrollib vastu võttes failide tüüpi ja suurust, ning nende sobivusel edastab need failihoidlale viirusekontrolliks. Mobiilist üles laetud faile näidatakse reaalajas ka originaalses klientrakenduses. Lisaks on täidetud ka kõik mittefunktsionaalsed nõuded.

Lahenduse üks olulisemaid osasid, mida täiendada on vaja, on failihoidla MetaDefender Vault-iga suhtlemiseks kasutatava teegi keeletaseme uuendamine. See võimaldaks siis ka loodud rakenduse keeletaset uuendada. See on oluline, sest hetkel kasutuses oleva keeletaseme tugi lõpetatakse septembris 2023 ja turvauuendustega toetamine 2026, mis võib muuta rakenduse pikas perspektiivis haavatavaks. Kuna rakendusega on edaspidi võimalik liidestada ka teisi sarnast funktsionaalsust vajavaid rakendusi, mis võivad olla ka teiste asutuste, kui Politsei ja Piirivalveameti halduses, tuleks rakendusele luua võimalus muuta mobiilivaates kasutatavat logo ja ka värve vastavalt telliva rakenduse identiteedile. Peamine viis selle tegemiseks oleks lisada rakendusele asutuste profiilid ja igale profiilile määrata oma kujundus.

6 Kokkuvõte

Diplomitöö eesmärgiks oli leida lahendus, mis võimaldaks kasutajatel Politsei lehel olevale süüteoavaldusele lisada faile teisest seadmest, nagu mobiiltelefon. Töö skoopi kuulus uue mikrorakenduse arendamine ja olemasoleva süsteemi täiendamine, et luua kasutajatele eraldi keskkond, kust faile avaldusse üles laadida.

Töö analüüsi peatükk andis põhjaliku ülevaate muudetavast rakendusest, planeeritavatest funktsionaalsustest ning lahenduses kasutatavast arhitektuurist ja tehnoloogiatest. Arenduskäiku kirjeldav peatükk andis ülevaate uue teenusrakenduse ehitusest, kasutatud lahendustest ning uue mobiilivaate loomisest. Lisaks sellele oli seal kirjeldatud täienduste tegemine olemasolevatele teenusrakendusele ja klientrakendusele.

Töö võib lugeda edukaks, sest on töö käigus õnnestus valmis teha planeeritud rakenduse prototüüp, mis täidab kõiki sellele seatud nõudeid. Peale rakenduse manuaalset vastuvõtutestimist on võimalik rakendus ka reaalsesse keskkonda kasutusse võtta.

Kasutatud kirjandus

[1] OPSWAT, Inc., „MetaDefender Vault - Storage You Can Trust,“ [Võrgumaterjal]. Available: https://www.opswat.com/products/metadefender/vault. [Kasutatud 4. mai 2023].

[2] „What is REST?,“ Codecademy, [Võrgumaterjal]. Available: https://www.codecademy.com/article/what-is-rest. [Kasutatud 4. mai 2023].

[3] C. Richardson, „What are microservices,“ 2023. [Võrgumaterjal]. Available: https://microservices.io/. [Kasutatud 10. aprill 2023].

[4] „What is HTTP Long Polling?,“ PubNub Inc., 17. jaanuar 2023. [Võrgumaterjal]. Available: https://www.pubnub.com/blog/http-long-polling/. [Kasutatud 15. aprill 2023].

[5] „The WebSocket API (WebSockets),“ MDN, [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. [Kasutatud 10. aprill 2023].

[6] L. Rosencrance ja B. Matturro, „Remote Procedure Call (RPC),“ TechTarget, oktoober 2021. [Võrgumaterjal]. Available: https://www.techtarget.com/searchapparchitecture/definition/Remote-Procedure-Call-RPC. [Kasutatud 28. märts 2023].

[7] T. Charboneau, „What’s the Difference Between RPC and REST?,“ 22. märts 2022. [Võrgumaterjal]. Available: https://nordicapis.com/whats-the-difference-between-rpc-and-rest/. [Kasutatud 28. märts 2023].

[8] „Java/OpenJDK End of Support,“ 19. aprill 2023. [Võrgumaterjal]. Available:

https://endoflife.date/java. [Kasutatud 23. aprill 2023].

[9] J. Brown, „Java EE vs. Spring: Which is more popular?,“ Xperti, 9. november 2021. [Võrgumaterjal]. Available: https://xperti.io/blogs/java-ee-vs-spring/. [Kasutatud 16. aprill 2023].

[10] A. Binstock ja S. Maple, „JVM Ecosystem report 2018 - About your Platform and Application,“ Snyk Limited, 17. oktoober 2018. [Võrgumaterjal]. Available: https://snyk.io/blog/jvm-ecosystem-report-2018-platform-application/. [Kasutatud 31. märts 2023].

[11] Stack Exchange Network, „2022 Developer Survey,“ 2022. [Võrgumaterjal]. Available: https://survey.stackoverflow.co/2022/. [Kasutatud 30. märts 2023].

[12] „React,“ Meta Open Source, [Võrgumaterjal]. Available: https://react.dev/. [Kasutatud 16. aprill 2023].

[13] OpenJS Foundation, „JQuery,“ OpenJS Foundation, [Võrgumaterjal]. Available: https://jquery.com/. [Kasutatud 16. aprill 2023].

[14] „What is Angular?,“ Google LLC, [Võrgumaterjal]. Available: https://angular.io/guide/what-is-angular. [Kasutatud 10. aprill 2023].

[15] E. You, „First Week of Launching Vue.js,“ 11. veebruar 2014. [Võrgumaterjal]. Available: https://blog.evanyou.me/2014/02/11/first-week-of-launching-an-oss-project/. [Kasutatud 10. aprill 2023].

[16] E. You, „Introduction,“ [Võrgumaterjal]. Available: https://vuejs.org/guide/introduction.html. [Kasutatud 10. aprill 2023].

[17] „Relational vs. Non-Relational Databases,“ MongoDB Inc., [Võrgumaterjal]. Available: https://www.mongodb.com/compare/relational-vs-non-relational-databases. [Kasutatud 23. märts 2023].

[18] „DB-Engines Ranking,“ Solid IT gmbh, 2023. [Võrgumaterjal]. Available:

https://db-engines.com/en/ranking. [Kasutatud 23. märts 2023].

[19] „PostgreSQL: About,“ The PostgreSQL Global Development Group, [Võrgumaterjal]. Available: https://www.postgresql.org/about/. [Kasutatud 2. mai 2023].

[20] „What Is MongoDB?,“ MongoDB Inc., [Võrgumaterjal]. Available: https://www.mongodb.com/what-is-mongodb. [Kasutatud 17. aprill 2023].

[21] Redis Ltd, „Introduction to Redis,“ Redis Ltd, [Võrgumaterjal]. Available: https://redis.io/docs/about/. [Kasutatud 17. aprill 2023].

[22] Redis Ltd, „Clients,“ Redis Ltd, [Võrgumaterjal]. Available: https://redis.io/resources/clients/. [Kasutatud 17. aprill 2023].

[23] „What is a message broker?,“ IBM Corporation, [Võrgumaterjal]. Available: https://www.ibm.com/topics/message-brokers. [Kasutatud 16. märts 2023].

[24] T. Hasan, „6 Top Message Brokers for Modern Applications,“ 31. august 2022. [Võrgumaterjal]. Available: https://geekflare.com/top-message-brokers/. [Kasutatud 3. mai 2023].

[25] „Overview - Memphis,“ Memphis, 31. märts 2023. [Võrgumaterjal]. Available: https://docs.memphis.dev/memphis/memphis/overview. [Kasutatud 6. aprill 2023].

[26] „Clients Libraries and Developer Tools,“ VMware Inc., [Võrgumaterjal]. Available: https://www.rabbitmq.com/devtools.html. [Kasutatud 6. aprill 2023].

[27] „Command Line Tools,“ VMware Inc, [Võrgumaterjal]. Available: https://www.rabbitmq.com/cli.html. [Kasutatud 6. aprill 2023].

[28] „Management Plugin,“ VMware Inc., [Võrgumaterjal]. Available: https://www.rabbitmq.com/management.html. [Kasutatud 6. aprill 2023].

[29] „AMQP 0-9-1 Model Explained,“ VMware Inc., [Võrgumaterjal]. Available: https://www.rabbitmq.com/tutorials/amqp-concepts.html. [Kasutatud 6. aprill

2023].

[30] „Kafka 3.4 Documentation,“ Apache Software Foundation, [Võrgumaterjal]. Available: https://kafka.apache.org/documentation. [Kasutatud 6. aprill 2023].

[31] „ActiveMQ Classic,“ Apache Software Foundation, [Võrgumaterjal]. Available: https://activemq.apache.org/components/classic/. [Kasutatud 6. aprill 2023].

[32] „Get started,“ iMatix, 2022. [Võrgumaterjal]. Available: https://zeromq.org/get-started/. [Kasutatud 6. aprill 2023].

[33] „Java Persistence API (JPA),“ IBM Corporation, [Võrgumaterjal]. Available: https://www.ibm.com/docs/en/was-liberty/nd?topic=overview-java-persistence-api-jpa. [Kasutatud 18. aprill 2023].

[34] F. Coros, „Separating Data Access Concern,“ 31. märts 2015. [Võrgumaterjal]. Available: https://oncodedesign.com/separating-data-access-concern/. [Kasutatud 18. aprill 2023].

[35] J. J. Sarjeant, „Getting Started,“ [Võrgumaterjal]. Available: https://axios-http.com/docs/intro. [Kasutatud 19. april 2023].

[36] A. Freda, „What Are QR Codes and How Do You Scan Them?,“ Avast Software s.r.o, 4. august 2022. [Võrgumaterjal]. Available: https://www.avast.com/c-what-is-qr-code-how-to-scan. [Kasutatud 20. aprill 2023].

[37] „Pub/Sub Messaging,“ Amazon Web Services, Inc., [Võrgumaterjal]. Available: https://aws.amazon.com/pub-sub-messaging/. [Kasutatud 20. aprill 2023].

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

Mina, Rasmus Ilmjärv

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Politsei veebilehe süüteoavaldusele funktsionaalsuse lisamine“, mille juhendaja on Meelis Antoi
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

15.05.2023

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

Lisa 2 – Klassi RabbitConfig programmikood

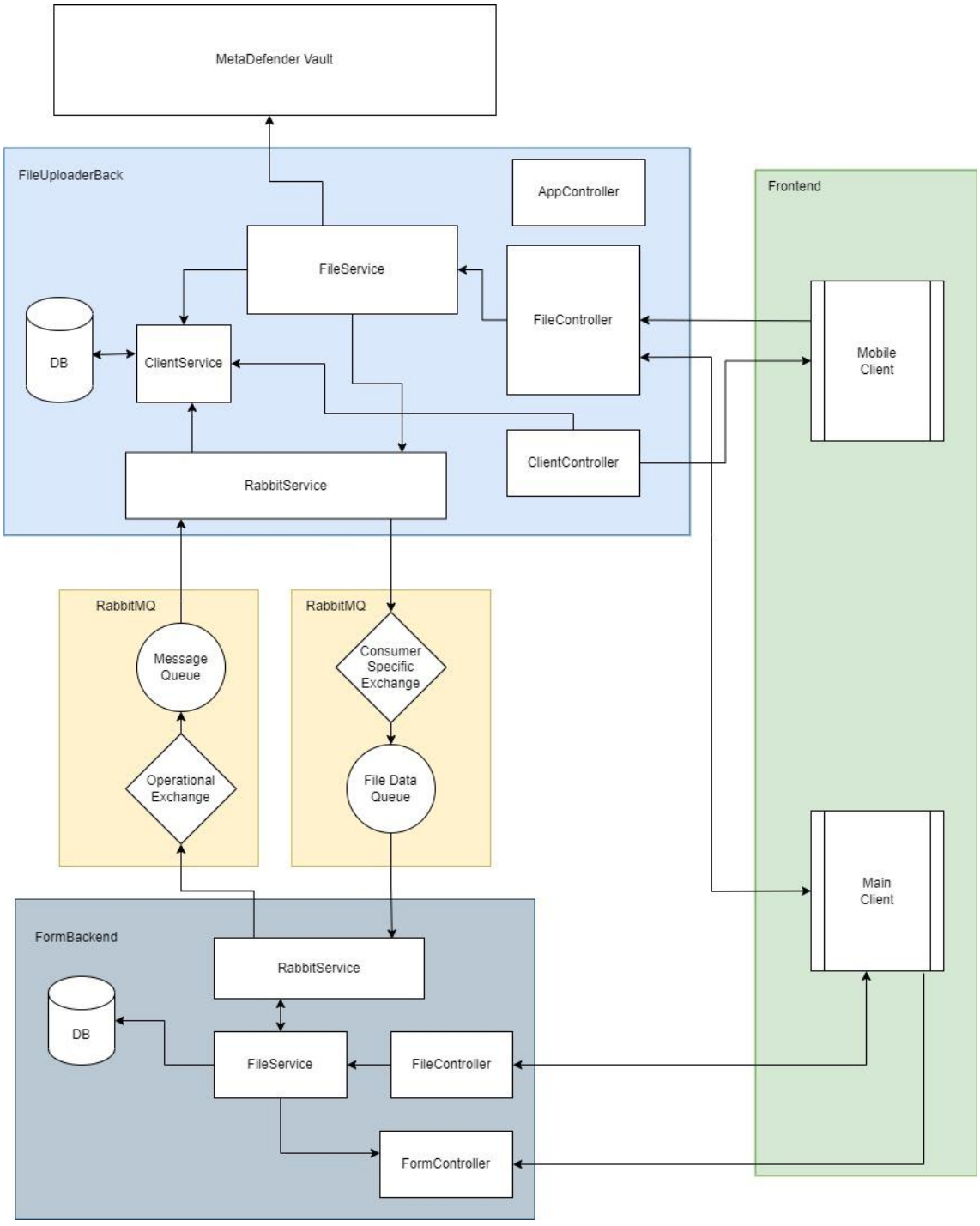
```
@Configuration
@Slf4j
@EnableRabbit
public class RabbitConfig {
    @Value("${spring.rabbitmq.msgQueueName}")
    private String messageQueue;
    @Value("${spring.rabbitmq.rpcName}")
    private String rpcName;
    @Value("${spring.rabbitmq.replyTimeout}")
    private int replyTimeout;

    @Bean
    public DirectExchange exchange() {
        return new DirectExchange(rpcName);
    }
    @Bean
    public Queue msgQueue() {
        return new Queue(messageQueue);
    }
    @Bean
    public Binding binding(DirectExchange exchange, Queue msgQueue) {
        return BindingBuilder.bind(msgQueue)
                .to(exchange)
                .with("rpc");
    }
    @Bean
    public RabbitTemplate rabbitTemplate(ConnectionFactory
connectionFactory){
        RabbitTemplate template = new RabbitTemplate(connectionFactory);
        template.setReplyTimeout(replyTimeout);
        return template;
    }
    @Bean
    public RabbitAdmin rabbitAdmin(RabbitTemplate rabbitTemplate) {
        return new RabbitAdmin(rabbitTemplate.getConnectionFactory());
    }
}
```

Lisa 3 – Registreeringu alustamiseks kasutatavat sõnumit kujutav klass RegistrationRequest ja selle vastus RegistrationResponce

```
@EqualsAndHashCode(callSuper = true)
@Data
@AllArgsConstructor
@NoArgsConstructor
public class RegistrationRequest extends BaseMessage {
    private String token;
    private String host;
    private long allowedSize;
    private long usedSize;
    private int minsToLive;
    private String[] extensions;
}
```

```
@AllArgsConstructor
public class RegistrationResponse extends BaseMessage {
    public String clientUrl;
    public String apiUrl;
    public String id;
    public String token;
    public String wsUrl;
    public LocalDateTime validUntil;
}
```

Lisa 4 – Lahenduse ehituse skeem



Lisa 5 – Loodud mikroteenuse kood versioonihaldussüsteemis

Aadress: https://bitbucket.org/rasmus_ilmjarv/fileuploader/src/master/