TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Gert Kuldma 192847IADB

Sõnumirakenduse intervallide salvestamine andmebaasi

Bakalaureusetöö

Juhendaja: Kristiina Hakk
PhD

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: Gert Kuldma

15.05.2023

Annotatsioon

Rakenduse planeerimisel ja arendamisel ei ole alati võimalik ette näha kõiki võimalikke probleeme, mis saavad tekkida rakenduse kasutamise käigus. Vahel aga on võimalik ennetada teatud probleeme rakenduse juures algselt arendusele rohkem aega panustades.

Bakalaureusetöö eesmärk on luua uus funktsionaalsus, mis võimaldab kasutajal muuta rakenduses teatud süsteemseid parameetreid läbi kasutajaliidese kiirelt, lihtsalt ja mugavalt. Süsteemsete parameetrite viimine konfiguratsioonifailist andmebaasi elimineerib vajaduse rakendus taaskäivitada parameetri väärtuse muutmiseks.

Töö käigus uuritakse olemasolevat rakendust, selles kasutatavaid tehnoloogiaid ja leitud informatsiooni põhjal vormistatakse vajalikud tehtavad arendused uue funktsionaalsuse realiseerimiseks.

Praktilises osas viiakse läbi arendus iteratiivselt, kus igas arendustsüklis täiendatakse erinevat süsteemi. Alustatakse andmebaasist ja liigutakse edasi läbi tagasüsteemi kasutajaliidesesse. Bakalaureusetöö lõpuks valmib kliendi rakendusele uus kasutatav lahendus.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 27 leheküljel, 6 peatükki, 18 joonist, 5 tabelit.

Abstract

Storing Message Application Processing Intervals to a Database

It is not always possible to anticipate all the possible problems that may arise during the use of an application when planning and developing it. However, sometimes it is possible to prevent certain issues with the application by initially dedicating more time to development.

The aim of this bachelor´s thesis is to create a new functionality that allows users to quickly, easily and conveniently change certain systemic parameters in the application through the user interface. Bringing systemic parameters from the configuration file to the database eliminates the need to restart the application when changing the parameter value.

During the work, the existing application and technologies used in it will be studied and based on the information found, necessary developments will be formulated to realize the new functionality.

The practical part will involve iterative development, where different parts of the system will be enhanced in each development cycle. Starting from the database layer and moving on the back-end and user interface. By the end of the bachelor´s thesis, a new usable solution will be completed for the client´s apllication.

The thesis is in Estonian and contains 27 pages of text, 6 chapters, 18 figures, 5 tables.

Lühendite ja mõistete sõnastik

Angular	JavaScript raamistik
API	*Application Programming Interface*, rakendusliides
CSS	*Cascading Style Sheets*, kaskaadlaadistik
GitLab	Platvorm versioonihalduseks
Gradle	Tarkvara arendamisel kasutatav automatiseerimistööriist
HTML	*HyperText markup language*, hüperteksti märgistuskeel
JavaScript	JavaScript
Jest	JavaScripti testimise raamistik
JUnit	Ühiktestide raamistik Java keelele
Liquibase	Andmebaasihaldustarkvara
Mockito	Ühiktestide raamistik Java keelele
NgRx	Oleku haldamise raamistik Angular rakendusele
REST	*Representational state transfer*, arhitektuuri stiil
Sprint Boot	Raamistik Java arenduseks
TypeScrypt	Tüübikirjeldusega JavaScript
UX/UI	*User Experience/User Interface*, kasutajakogemus/kasutajaliides

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Rakendust planeerides ja nõudeid analüüsides pööratakse tihti vähe tähelepanu rakenduse süsteemsetele parameetritele. Sageli võib juhtuda, et on vajalik mõne parameetri muutmine rakenduse töö mõjutamiseks: stabiliseerimiseks, koormuse hajutamiseks, pordi muutmiseks, andmebaasi vahetamiseks vms. Konfiguratsioonifaili muutmine toob kaasa aga vajaduse rakendus taaskäivitada ja see ei ole alati igal hetkel võimalik.

Süsteem, mida lõputöö raames täiendatakse, kuulub kliendile. Nõuded ja tingimused esitab klient omalt poolt teostajale. Teostaja on antud töös meeskond, kes tegeleb terve arenduse protsessiga. Meeskonnas on nii analüütikud, arendajad kui ka testijad.

Lõputöö eesmärk on olemasolevat süsteemi täiendada, et kasutajal oleks võimalik läbi kasutajaliidese vaadata, lisada, kustutada ja muuta sõnumitöötlusintervalle ilma rakenduse tööd häirimata. Loodav lahendus peab olema kasutajale mugav ja kindlustama, et kasutaja ei saaks intervallide sisestamisel teha vigu, mis võivad rakenduse tööd häirida. Autori soov on luua lahendus, mille ligipääs paikneb rakenduse kasutajaliideses.

Autor töötab meeskonnas ja seetõttu ei realiseerinud autor kogu lahendust üksinda. Autori ülesanded hõlmasid koodi kirjutamises uue andmebaasi tabeli loomist, sõnumi intervallide lisamise ja kustutamise teenuseid rakenduse tagasüsteemis, kasutajaliidesesse uute komponentide kirjutamist intervallide haldamiseks ja osade testide kirjutamist, mis täpsemalt on välja toodud tehtud tööde kirjeldustes.

Bakalaureusetöö sisaldab kuut peatükki. Töö teises peatükis vaadeldakse lähemalt lahendatava probleemi tausta ja kirjeldatakse töö eesmärk. Kolmandas peatükis analüüsitakse lahenduse nõudeid ja kasutatavaid tehnoloogiaid. Neljandas peatükis kirjeldatakse lahenduse loomist. Viiendas peatükis tutvustatakse töö tulemusi ja autor toob välja võimalused, kuidas andmebaasist rakendatavaid sõnumitöötluse intervalle oleks võimalik veel edasi arendada.

2 Probleemi taust ja töö eesmärk

Käesolevas peatükis kirjeldatakse probleemi tausta, seatakse eesmärgid ja antakse ülevaade rakenduse süsteemsete parameetrite hetkeolukorrast, mida kavatsetakse parandada.

2.1 Probleemi taust

Uue rakenduse ehitamine algab analüüsist. Analüüsi käigus on vajalik läbi mõelda loodava rakenduse mitmed küljed. Näiteks on analüüsi käigus vajalik tuvastada, kellele ja miks uut rakendust luuakse. Kasutajate tundmine aitab kaasa nende vajaduste ja nõuete paremale analüüsile. Analüüs hõlmab ka rakenduse ärireeglistiku koostamist. Analüütiku töö on kirja panna funktsionaalsed nõuded, mida uus loodav rakendus peab olema suuteline tegema. [1]

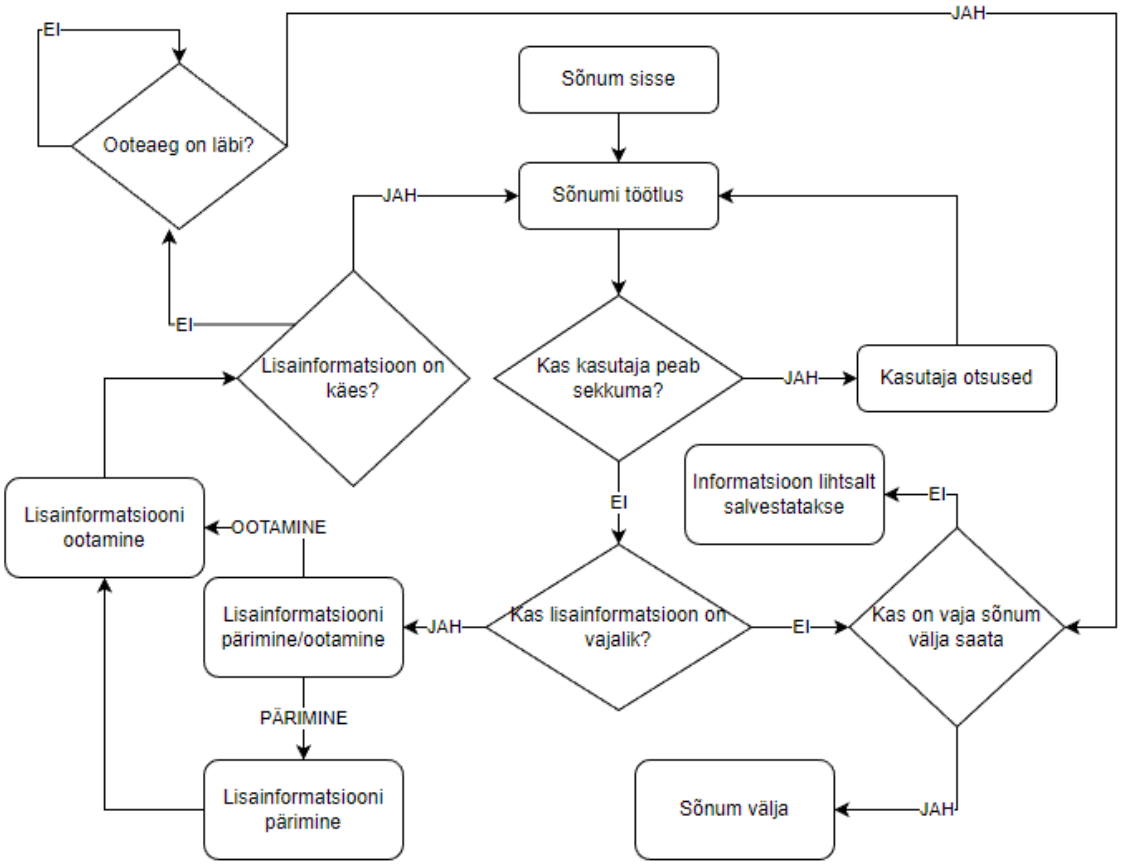
Analüüsi protsessile järgneb rakenduse arenduse planeerimine. Tänapäeval on populaarseks muutunud agiilsed arendusprotsessid [2]. Planeerimise käigus hakatakse looma kasutajalugusid. Iga kasutajalugu tagab ühe või mitme rakenduse tööks vajaliku funktsionaalsuse valmimise. Kasutajalugu sisaldab omakorda väikesemaid arendusülesandeid, mis kokku lahendavad loo ja mida on võimalik jagada laiali arendajatele. Nii saavad erinevad arendajad paralleelselt tööd teha ja rakendusega kiiremini edasi liikuda. Iga loodav ülesanne peab lahendama ühe funktsionaalse tüki rakenduses. [3]

Rakenduse funktsionaalsete osade ehitus põhineb rakenduse arhitektuuril, mille loomine on arhitekti ülesanne.

Töö tellijaks oleva kliendi rakendus töötleb erinevaid sõnumeid. Sõnumid on seotud erinevate toodetega ja on mõeldud nende kohta käiva informatsiooni töötlemiseks. Sõnumid tulevad sisse, neid töödeldakse, salvestatakse vajalik info andmebaasi ja vajadusel saadetakse sõnum edasi välja. Rakenduses on palju erinevaid sõnumitüüpe ja igal sõnumil on erinev äriline funktsionaalsus.

Igal sõnumitüübil on rakenduses erinev protsess, kuidas informatsiooni töödeldakse. Rakenduses ei eksisteeri kindlat tasakaalu ja ajas on erinevate sõnumitüüpide poolt tekitatav koormus rakendusele erinev. Mõne sõnumi puhul on töötlemine lihtne ja kiire,

teiste puhul saadetakse lisainfo päring välja ja andmetöötlus läheb pausile seniks, kuni tuleb sisse päritud täiendav informatsioon. Sõnumiprotsessi lihtsustatud voodiagramm on toodu joonisel 1.



Joonis 1. Lihtsustatud sõnumiprotsessi voodiagramm.

Selleks, et muuta rakenduse tasakaalu ja prioriseerida mõne kindla sõnumi töötlemist, on rakenduse konfiguratsioonifaili lisatud sõnumitöötlemise intervallid. Intervallid annavad võimaluse rakenduse töövoo juhtimiseks ja tasakaalustamiseks.

Käesoleva rakenduse algsel projekteerimisel pole hinnatud või ette nähtud vajadust antud intervallide muutmiseks reaalajas ilma rakenduse tööd seejuures häirimata. Rakendust kasutades on aga tulnud välja, et vahel on vajalik antud intervalle rakenduse tasakaalustamiseks muuta. Sellega kaasneb hetkel vajadus rakendus taaskäivitada, mis tekitab tööseisaku ja nõuab mitme partneri teavitamist.

Lõputöö autori ülesanne on tiimiga täiendada rakenduse funktsionaalsust, et oleks võimalik intervallide muutmisega seoses tekkivad probleemid kõrvaldada. Autor on

vastutav andmebaasi tabeli loomise, tagarakenduses intervallide lisamise ja kustutamise teenuste lisamise, kasutajaliidese vaadete loomise ja oma loodud koodi testimise eest.

2.2 Töö eesmärk

Siin peatükis kirjeldatakse lõputöö raames loodud funktsionaalsuse kasulikkust kliendile. Kliendi jaoks on tähtsad kasutajamugavus, rakenduse kättesaadavus ja andmete turvalisus.

2.2.1 Kasutajamugavus

Üks töö eesmärk on muuta sõnumitöötlemise intervallide muutmine lihtsamaks ja turvalisemaks. Viies intervallide muutmise kasutajaliidesesse, on võimalik neid muuta igal kasutajal, kellele on vastavad õigused antud. Lisaks on võimalik kasutajaliideses rakendada erinevaid vaildatsioonireegleid, mis vähendavad tõenäosust, et kasutaja võiks sisestada vigaseid andmeid ja seeläbi halvata rakenduse töö. Sobiva arendusega on võimalik tõsta kasutajamugavust.

2.2.2 Kättesaadavus

Kliendi sõnumirakendus on tihedalt töös. Tagamaks töövoogude stabiilsust, peab rakendus olema kättesaadav ja seetõttu on vajalik vähendada rakenduse maasoleku aega. Üks soblik võimalus selle saavutamiseks on täiendada rakenduse funktsionaalsust uute võimalustega, mis töötaksid ilma rakendust sulgemata. Tänasel päeval peab klient teavitama teist keskkonda, kust sõnumid tulevad, igast planeeritud ja planeerimata rakenduse maasolekust. See on vajalik tagamaks info terviklikkust, mis tähendab, et maasoleku ajal saabuvad sõnumid saadetakse tagantjärgi rakenduse üles tulles.

Rakenduse puhul on palju erinevaid võimalusi, mis võivad tekitada probleeme rakenduse toimimises või põhjustada vajaduse rakendus taaskäivitada. Käesoleva lõputöö raames on plaanis eemaldada üks võimalikest olukordadest, mis nõuaks rakenduse taaskäivitamist. Lisaks annab loodav funktsionaalsus võimaluse uuteks arendusteks tulevikus.

2.2.3 Turvalisus

Hetkel peab kasutaja rakenduse konfiguratsioonifailis sõnumitöötlemise intervalli vahetama käsitsi ja seejärel rakenduse taaskäivitama. Selline lähenemine nõuab kasutajalt spetsiifilisi teadmisi ja loob võimaluse vigade tekkimiseks. Lisaks puudub antud

lahenduse puhul muudatuste jälgitavus. Arenduse juures kasutatakse versioonihaldussüsteemi, mis tagab selle jälgitavuse, kuid kliendi keskkonnas on kliendil endal võimalik neid muuta vastavalt soovile ja vajadusele. Käsitsi intervallide muutmisega on aga võimalik tekitada vigu ja seeläbi pikendada rakenduse maasoleku aega.

Lõputöö raames loodava lahendusega on võimalik tagada muudatuste jälgitavus ja läbi validatsioonireeglite loomise ka sisestatavate andmete korrektsus.

2.3 Ülevaade lähteolukorrast

Enne arenduse alustamist kaardistatakse kliendi rakenduse hetkeseis. Uuritakse kuidas kasutatakse erinevaid süsteemseid parameetreid sõnumitöötlusintervallide töötlemisel. Andmebaasi kontrollimisel ilmneb, et mõningad süsteemsed parameetrid on sinna juba paigutatud. Nende jaoks on loodud andmebaasi eraldi tabel. Edasi uuritakse, kuidas on antud parameetrid kasutusel taustsüsteemi API (*Application Programming Interface*) rakenduses ja kuidas toimub nende muutmine kasutajaliideses.

Taustsüsteemi uurimisel leitakse API päringud, mis võimaldavad tuvastada süsteemsete parameetrite olemasolu andmebaasis ja nende väärtusi. Kuna teisi vajalikke API päringuid, mis on vajalikud parameetri lisamiseks, muutmiseks ja kustutamiseks, ei leita, siis kontrollitakse tekkinud hüpoteesi jaoks kliendi kasutajaliidese rakendust. Kasutajaliidese uurimisel selgub, et andmebaasist leitud süsteemsed parameetrid on fikseeritud ja neid ei ole võimalik kasutajaliidesest lisada, kustutada ega muuta.

Edasisel uurimisel selgub, et leitud süsteemsed parameetrid on muudetavad ainult kasutades Liquibases stsenaariumit. Selline lahendus aga eeldab, et süsteemsete parameetrite muutjaks on IT taustaga isik, kellel on vajalikud teadmised andmebaasi halduseks.

Selline lahendus on aeglasem ja vajab spetsiifilisemate teadmistega kasutajat. Kui oleks võimalus, et neid parameetreid saaks vajadusel muuta ka kasutajaliidesest, saaks paremini kontrollida, kes ja kuidas neid muudab.

3 Lahenduse analüüs

Lahenduse analüüsifaasi eesmärk on äriliste vajaduste põhjal selgitada välja, millised tehnilised tegevused on vajalikud lahenduse realiseerimiseks ja milline on taustsüsteem, kus arendus toimub.

3.1 Nõuded lahendusele

Kliendi soovide tõlgendamise ülesanne on analüütikul, kes vormistab detailanalüüsi. Peale detailanalüüsi valmimist viiakse detailanalüüsi täpsustamiseks läbi planeerimiskoosolek [4], kus osalevad ka teemaga seotud arendajad. Autor osaleb lahenduse loomisel alates planeerimiskoosolekust, kus arutatakse kuidas kliendi soovitud funktsionaalsust võimaldada. Planeerimiskoosoleku käigus luuakse arendusülesanded ja hinnatakse nende ajakulu kasutades planeerimispokkerit [5]. Planeerimispokkeri käigus arutatakse iga eelnevalt loodud ülesanne läbi, proovitakse ette kujutada toiminguid, mis selle arendusega kaasnevad ja seejärel on igal osalejal üks hääl, millega ennustada, kui pikk aeg kulub antud ülesande lahendamiseks. Planeerimispokkeril hääletavad arendajad. Detailanalüüs annab arendajatele ka suunised, millised on kliendi soovid ja nõuded loodavale funktsionaalsusele.

Käesoleva töö raames loodava funktsionaalsuse nõuded on seotud sõnumitöötluse intervallidega, mis on algselt defineeritud rakenduse seadistusfailis. Muudatused on vajalikud rakenduse kasutamise lihtsustamiseks. Detailanalüüsi põhjal on kliendi nõuded järgmised:

- Luua uus kuva seadistuste vaatamiseks ja muutmiseks.
- Luua keskkonda uus roll ja vastavad vajalikud privileegid.
- Kasutaja saab rakenduses kasutusel olevate sõnumitüüpide töötlemise intervallide väärtusi muuta ja kustutada.
- Kasutaja saab lisada sõnumitüübile töötlemise intervalle.
- Kasutaja saab määratud sõnumite taaskäivitamise intervalle muuta.
- Kasutaja saab määratud taustatööde ajastuste väärtusi muuta.

3.2 Kasutatavad tehnoloogiad

Käesoleva lõputöö probleemi lahendamisel kasutatavaid tehnoloogiad lõputöö autor ise ei vali. Arendatav funktsionaalsus peab sobituma olemasolevate süsteemidega nii andmebaasi, tagarakenduse kui ka kasutajaliidese tasandil. Järgnevalt on kirjeldatud antud süsteemis eelnevalt valitud tehnoloogiaid.

3.2.1 Andmebaas

Rakendus kasutab PostgreSQL andmebaasi. PostgreSQL on avatud lähetkoodiga objekt-relatsiooniline andmebaasihaldussüsteem. Kuna tegemist on vabavaralise tasuta süsteemiga, siis on see muutunud aina populaarsemaks ja leiab üha rohkem kasutust nii väikeste kui ka suuremate rakenduste ehitamisel. [6]

PostgreSQL-i tugevaks küljeks on selle kasutatatvus erinevatel platvormidel: Windows, Linux ja macOS. PostgreSQL-i on lihtne ja mugav kasutada ning seda saab hallata läbi erinevate tööriistade: käsurealt, rakendustest. Lõputöös kasutatavas rakenduses on andmebaasi haldamiseks kasutusel tööriist Liquibase. [7]

Liquibase on samuti avatud lähtekoodiga tööriist, mis võimaldab hallata andmebaasiskeeme ja migratsiooni. Liquibase kasutamine lisab mugava võimaluse hallata uuenduste versioone, millega muudetakse andmebaasi. Liquibase on kasutatav mitmete erinevate andmebaasi süsteemidega: Oracle, MySQL, PostgreSQL jm. [8]

Liquibase abil on võimalik andmebaasiskeemi muudatuste rakendamine automatiseerida, see võimaldab arendajatel vältida vigade tegemist, mis võiksid andmebaasi lõhkuda. See muudab Liquibase väärtuslikuks tööriistaks tarkvaraarenduse meeskondadele, kes soovivad hallata oma andmebaase kiirelt, mugavalt ja tagada nende terviklikkuse. [9]

3.2.2 Tagasüsteem

Rakenduse tagasüsteem on Java veebisüsteem, mis on ehitatud kasutades tööriista Gradle ja Spring Boot raamistikku. Java Spring on üks maailmas enim kasutatavatest Java raamistikest [10]. Spring Boot ja Gradle on võetud kasutusele, sest see muudab rakenduse arendamise mugavamaks ja vähendab rakenduse ehitamiseks vajamineva konfiguratsiooni loomist. See võimaldab arendajatel tegeleda rohkem uute funktsionaalsuste loomisega.

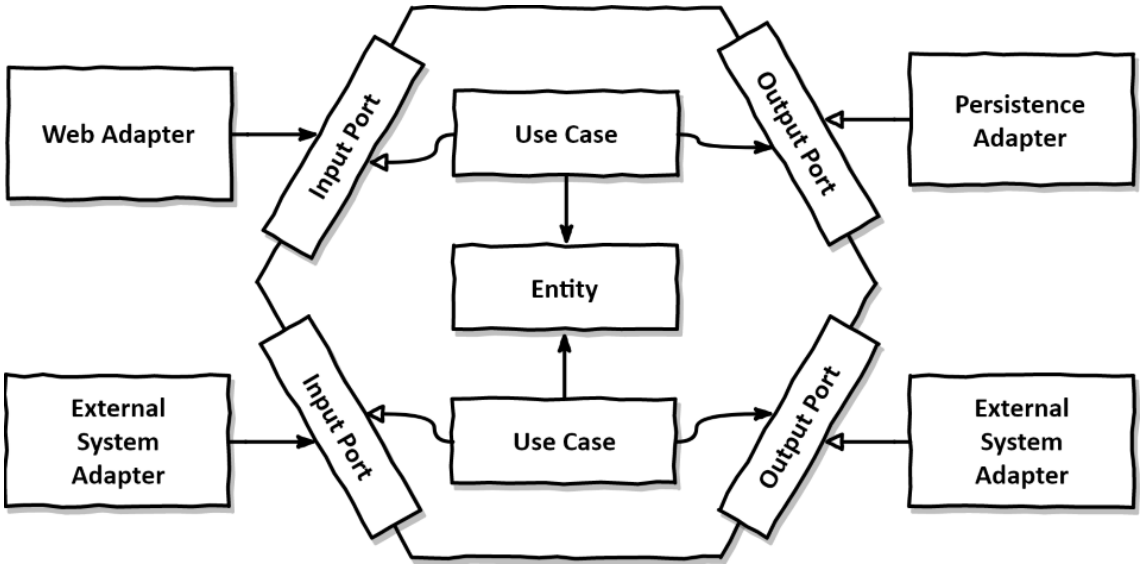
Tagasüsteemi ehitamisel on kasutatud heksagonaalset rakenduse struktuuri. Heksagonaalse struktuuri ehitamisel jälgitakse „väljast sisse“ reegleid. Sellise struktuuri keskmes paiknevad domeeni moodulid, mis kirjeldavad kõiki klasse, mida rakendus vajab ärilise funktsionaalsuse täitmiseks. Kui ärireeglid nõuavad paljude erinevate domeenide kasutamist, siis võib lugeda antud domeeni moodulit ka rakenduse elujõu allikaks. Domeeni klassid on kirjutatud puhta Java-ga ja neil puuduvad välised sõltuvused. Tänu väliste sõltuvuste puudumisele ei mõjuta ülejäänud rakenduse kihtide muudatused domeeni moodulit. [11]

Domeeni moodulitega on ühendatud kasutuslugude moodulid. Kasutuslugude moodulid sisaldavad klasse, mis tegelevad äriliste protseduuride realiseerimisega. Iga kasutajaloo klass tegeleb ühe kindla tegevusega. Kasutajaloo klassid on samamoodi isoleeritud nagu domeeni klassid, mis tähendab, et saamaks vajalikku informatsiooni väljaspoolt peame kasutama ühendusliideseid. [12]

Kasutajaloo moodul on seotud ühendusliidestega, mis jagunevad kaheks: sisenevad ja väljuvad. Läbi ühendusliideste pääseb informatsioon kasutajaloo moodulisse ja sealt välja. Antud liideseid kasutades on meil kindlalt struktureeritud, kus ja millisel kujul saab informatsioon liikuda rakenduse sisemistesse kihtidesse ja sealt välja. [13]

Liidesed on seotud struktuuri välimises kihis paiknevate adaptermoodulitega. Adapterid seovad rakenduse välise maailmaga. Adaptereid saab vastavalt vajadusele lisada, eemaldada ja muuta, sest nad ei ole otse seotud rakenduse sisemiste kihtidega ega mõjuta neid. Adapteriteks võivad olla nii kasutaja veebiliides, andmebaasi liides, ajutine äriloogika täiendus või muu. [14]

Heksagonaalne struktuur teeb esialgse rakenduse planeerimise keerulisemaks, kuid tagab hilisemas kasutuses lihtsama võimaluse väliseid rakenduse kihte muuta ja seeläbi rakenduse kasutust täiendada ja optimeerida. Heksagonaalne rakenduse struktuur on kujutatud joonisel 2.



Joonis 2. Heksagonaalse rakenduse struktuur [15].

Kliendi rakenduse tagasüsteemi võib lugeda keskseks süsteemiks, sest see haldab ja töötleb teavet, mida liigutatakse läbi API. Tagasüsteemis paikneb peamine osa süsteemi äriloogikast, mis hõlmab suurt hulka erinevaid reegleid ja protsesse, mis on vajalikud erinevate sõnumite valideerimiseks, töötlemiseks, salvestamiseks ja saatmiseks. Kõiki neid ülesandeid lahendab tagasüsteem ööpäev läbi ja seetõttu on vajalik tagada antud API rakenduse kättesaadavus. Rakenduse kättesaadavust annab suurendada, kui täiendada rakenduse funktsionaalsust nii, et seeläbi väheneb kordade hulk, mille käigus on vajalik rakenduse taaskäivitamine.

3.2.3 Kasutajaliides

Rakenduse kasutajaliides on ehitatud kasutades raamistikku Angular. Angular on TypeScriptil põhinev avatud lähtekoodiga raamistik ühelehe rakenduste loomiseks. Angular leht ehitatakse kokku komponentidest. Selline struktuur lubab rakendust kiiresti suuremaks projekteerida ja laiendada vastavalt soovitule. Kõik ehitatud komponendid on korduvkasutatatavad, mis vähendab teatud hulga koodi dubleerimist. Angulari kasutamiseks on vajalik omada eelteadmisi nii TypeScrypt-i, HTML (*HyperText markup language*) kui ka CSS (*Cascading Style Sheets*) kohta. [16] Aastal 2022 oli Angular veebirakenduste Javascript raamistike kasutatavuses viiendal kohal. [17]

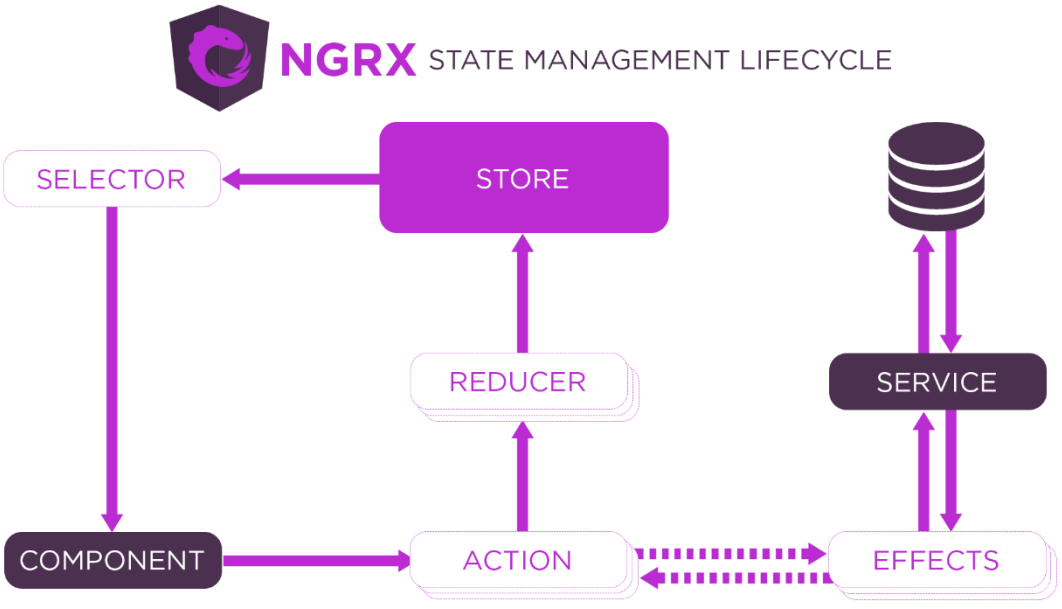
Veebirakenduses on vajalik hoida ka andmeid rakenduse jaoks vajalike parameetrite kohta. Need parameetrid kokku moodustavad rakenduse oleku. Oleku halduseks on kliendi rakenduses kasutusel vabavaraline raamistik NgRx. NgRx võimaldab andmeid

hallata keskselt ja organiseeritult läbi kõikide komponentide kasutajaliideses, mis on erinevate olekutega seotud. Olekumuutused NgRx süsteemis toimuvad sündmuspõhiselt. [18]

NgRx-i kasutades oleku haldamiseks on vajalik luua mitmed erinevad osad. Vajalikud osad ja nende funktsionaalsus on toodud tabelis 1. Antud osade omavaheline seos on kajastatud joonisel 3.

Tabel 1. Oleku haldamiseks vajalikud osad.

Osa	**Funktsionaalsus**
Konteiner (Store)	Hoiab vastava osarakenduse olekut.
Toimingud (Action)	Vajalikud protsessid poes hoitava oleku muutmiseks, mida on võimalik vastavalt vajadusele välja kutsuda.
Vähendajad (Reducer)	Kui mõni toimingu protsess välja kutsutakse, siis vähendajad muudavad poes olevat olekut.
Valijad (Selector)	On kasutusel poest parameetri pärimiseks.
Effektid (Effect)	On kasutusel teenuste käivitamiseks ja oleku muutmiseks kui rakendatakse vastav toiming.
Teenus (Service)	On kasutusel andmete pärimiseks ja saatmiseks süsteemist välja.



Joonis 3. NgRx oleku haldamise skeem [19].

3.2.4 Testimine

Kliendi rakenduses on kasutajaliides ja tagarakendus valdaval määral testidega kaetud. Tihti on see ka nõudena sisse kirjutatud rakenduse tellimisel. Rakenduse testidega katmine tagab kindlustunde, et vajalik funktionaalsus ja võimalik tööprotsessid rakenduses toimivad nagu reeglid ette näevad. [20]

Rakendust arendades on võimalik testimisele läheneda mitmel viisil. Kasutatakse nii võimalust kirjutada testid esimeses etapis kui ka viimases. Esimeses etapis testide kirjutamist nimetatakse testipõhiseks arenduseks (*test-driven development*), selle käigus luuakse algselt testid, mille põhjal on võimalik hakata rakenduse koodi kirjutama. Sellisel juhul on juba rakendusse ette seatud ärilised reeglid, mida loodav funktsionaalsus peab järgima. Kui aga testid kirjutada lõpus peale funktsionaalsuste loomist, siis seda nimetatakse koodipõhiseks arenduseks (*code-driven development*). Koodipõhisel arendusel kinnitatakse testidega peale funktsionaalsuse loomist selle täpne seis ja olek, mis võimaldab hinnata tulevikus, et rakenduse koodi muutmisel oleks rakenduse funktsionaalsus endiselt kooskõlas ärireeglitega, juhul kui neid pole muudetud. Kui rakenduse ärireegleid muudetakse, on vajalik ajakohastada ka rakenduse testid. Tänapäeval on peamiselt kasutusel nende kahe meetodi hübriidvariant ja teste kirjutatakse jooksvalt vastavalt vajadusele ja võimalusele. [21]

Kliendi tagarakenduses on kasutusel ühik-ja integratsioonitestid. Testide kirjutamiseks on kasutatud vabavaralist raamistikku JUnit. JUnit on loodud 1997. aastal ja leidnud palju kasutust. JUnit on kasutusel ühiktestide loomisel. Ühiktestid võimaldavad kontrollida rakenduse erinevate funktsionaalosade korrektset töötamist. Selleks valitakse testitav funktsionaalne osa ja kirjutatakse sellele erinevad testjuhud, mis testivad funktsiooni õiget käitumist erinevates olukordades. [22]

Kuna ühiktestid on isoleeritud, siis on vajalik vahel kasutada ka Mockito raamistikku. Mockito on samuti vabavaraline raamistik, kirjutatud Java-s. Mockito võimaldab luua simulatsiooniobjekte funktsiooni välistest objektidest. Seeläbi on võimalik simuleerida teisi objekte, mida testitaval funktsioonil on vaja kasutada enda protsessi käigus. Mockito kasutamine võimaldab luua paremaid teste, mis katavad rohkem võimalikke tekkivaid olukordi. Selle tarbeks on võimalik Mockitot kasutades juhtida rakenduse käitumist ja simuleerida välista funktsioonide vastuseid kui nende poole pöördutakse. Kui loodav

funktsioon on seotud teiste funktsioonidega, siis peab ta käituma korrektselt ka juhul kui mõni teine seda ei tee. [23]

Kliendi rakenduse kasutajaliidese testide jaoks on kasutusel Jest JavaScript raamistik. Jest on loodud algselt React-i kasutavate rakenduste testimiseks, kuid see leiab kasutust ka teiste raamistike juures. Kliendi rakenduses on kasutusel Angular raamistik ja selle juures sobib kasutada Jest-i. Jest-i peamised eelised on selle kiirus ja lihtsus. Lisaks võimaldab Jest sarnaselt Mockito-le simuleerida erinevaid tekkivaid situatsoone rakenduse töös. [24]

4 Lahenduse loomine

Siin peatükis kirjeldatakse erinevate rakenduse osade arendust, mis on vajalikud soovitud lahenduse realiseerimiseks.

4.1 Andmebaasi uue tabeli loomine

Sõnumitöötlusintervallide salvestamiseks andmebaasi on vaja luua sinna uus tabel, mis oleks sobiv salvestama vajalikke andmeid. Kliendi rakenduse andmebaasi muudatuste sisseviimiseks on kasutusel Liquibase andmebaasihaldusrakendus, mis võimaldab lihtsalt ja kiirelt andmebaasi uuendada.

Selleks, et alustada uue tabeli loomist, peab paika panema uue tabeli struktuuri ja hindama, millist ning mis tüüpi infot on plaanis seal hoida. Arvestades, et loodavad intervallid koosnevad nimest ja väärtusest, siis valitakse uue tabeli jaoks minimaalne vajalik tabeli struktuur. Tabeli väljad ja nende kirjeldused on esitatud tabelis 2.

Tabel 2. Intervallide tabeli andmebaasi read.

Nimetus	**Tüüp**	**Kirjeldus**
Type	varchar(50)	Sõnumitüübi nimetus, mille järgi väärtust pärida
Values	Integer[]	Väärtused
Creator	jsonb	Looja
Created	timestamp(6)	Loomise aeg
Modifier	jsonb	Muutja
Modified	timestamp(6)	Muutmise aeg

Vastavalt eelnevalt koostatud väljadele luuakse edasi Liquibase stsenaarium, mis loob vastava uue tabeli andmebaasi. Kuna arenduse käigus on kasutusel lokaalne PostgreSQL andmebaas, siis testitakse stsenaariumi nii uue andmebaasi loomiseks, kui ka olemasoleva andmebaasi uuendamiseks süsteemis veendumaks, et uus loodud stsenaarium oleks kasutatav korrektselt mõlemal juhul. Liquibase stsenaarium on toodud joonisel 4.

```
<changeSet id="add_message_processing_intervals_table" author="gk">
  <createTable tableName="message_processing_intervals"
schemaName="cor"
  remarks="Sõnumitöötluse intervallid">
    <column name="type" type="varchar(50)">
      <constraints primaryKey="true"
primaryKeyName="message_processing_interval_pkey"
      nullable="false"/>
    </column>
    <column name="value" type="integer[]" remarks="Väärtus">
      <constraints nullable="false"/>
    </column>
    <column name="creator" type="jsonb" defaultValue='{"system":true}'
remarks="Looja">
      <constraints nullable="false"/>
    </column>
    <column name="created" type="timestamp(6) with timezone"
    defaultValueComputed="now()" remarks="Loomisaeg">
      <constraints nullable="false"/>
    </column>
    <column name="modifier" type="jsonb"
defaultValue='{"system":true}'
    remarks="Muutja">
      <constraints nullable="false"/>
    </column>
    <column name="modified" type="timestamp(6) with timezone"
    defaultValueComputed="now()" remarks="Muutmisaeg">
       <constraints nullable="false"/>
    </column>
  </createTable>
</changeSet>
```

Joonis 4. Liquibase stsenaarium.

4.2 API päringute loomine

Selleks, et rakenduse kasutajaliidese kaudu oleks võimalik lisada, pärida, muuta ja kustutada sõnumiintervalle andmebaasis, on vaja luua uued päringud tagarakendusse.

Kuna rakendus on ehitatud heksagonaalse struktuuriga, algab funktsionaalsuse loomine rakenduse sisemisest ehk domeenikihist. Domeenikihti lisatakse klassid „MessageProcessingInterval“ ja „MessageProcessingIntervalValidator“. Validaator luuakse tagamaks, et andmebaasi ei jõuaks intervall, mida ei ole üldises intervallide nimekirjas. Vastavad klassid on esitatud vastavalt joonisel 5 ja joonisel 6.

```
public class MessageProcessingInterval {
private MessageType type;
private List<Integer> values;
}
```

Joonis 5. Klass MessageProcessingInterval.

```
public class MessageProcessingIntervalValidator {
  static void
validateMessageProcessingInterval(MessageProcessingInterval
messageProcessingInterval) {
    if (messageProcessingInterval == null) {
      throw new Invalid(MESSAGE_PROCESSING_INTERVAL_DOES_NOT_EXIST);
    }
  }
  public static class Invalid extends BusinessException {
    public Invalid(Violation violation) {
      super(violation.name());
    }
    public enum Violation {
      MESSAGE_PROCESSING_INTERVAL_DOES_NOT_EXIST,
      MESSAGE_PROCESSING_INTERVAL_ALREADY_EXIST
    }
  }
}
```

Joonis 6. Klass MessageProcessingIntervalValidator.

Teises etapis täiendatakse rakenduse kasutajaloo kihti, kuhu lisatakse intervalli lisamise ja kustutamise kasutajaloo klassid vastavalt „AddMessageProcessingInterval“ ja „DeleteMessageProcessingInterval“.

Sõnumiintervalli lisamisel algab protsess intervalli koodi valideerimisega, mis kindlustab, et intervalli kood eksisteerib, vastasel juhul tagastab vea. Edasi kontrollitakse, et antud intervalli kood ei ole juba andmebaasi lisatud, selleks saadetakse päring intervallide leidmise liidesele. Juhul kui andmebaasis eksisteerib antud intervalli kood, siis tagastatakse vastav veateade. Juhul kui validatsioonid vigu ei tuvasta, saadetakse uue intervalli andmed intervalli lisamise liidesele. Intervalli kustutamise kasutajaloos kasutatakse ühte validatsiooni. Valideeritakse, et intervall, mida soovitakse kustutada eksisteerib andmebaasis. Kui validatsioon viga ei tagasta siis saadetakse kustutatava intervalli andmed intervalli kustutamise liidesele.

Kolmandas etapis lisatakse rakenduse liideste kihti intervallide lisamise ja kustutamise liidesed. Nendeks liidesteks kirjutatakse „AddMessageProcessingIntervalPort“ ja „DeleteMessageProcessingIntervalPort“ mis on toodud vastavalt joonistel 7 ja 8. Antud liidesed on vajalikud, et liigutada informatsiooni kasutajalugude ja teenuste kihi vahel. Intervallide lisamise ja kustutamise liidesed on seest-välja suunaga ja on mõeldud informatsiooni edastamiseks rakenduse teenuste kihti.

```
public interface AddMessageProcessingIntervalPort {
  void addMessageProcessingInterval
(MessageProcessingInterval messageProcessingInterval);
}
```

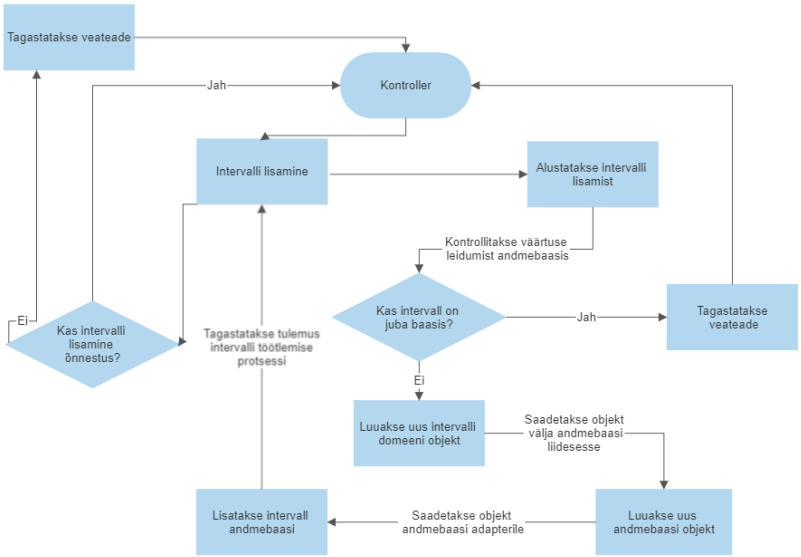
Joonis 7. Liides AddMessageProcessingIntervalPort.

```
public interface DeleteMessageProcessingIntervalPort {
  void delete(MessageType type);
}
```

Joonis 8. Liides DeleteMessageProcessingIntervalPort.

Rakenduse teenuste kihis on erinevad väravad. Väravad seovad kokku erinevad liidesed ja nendele vastavad teenused, mis on grupeeritud vastavalt funktsionaalsustele. Sõnumi intervalli lisamise ja kustutamise liidestest liigub informatsioon sõnumitöötluse väravasse, kuhu on seotud kokku kõik sõnumitöötlusega seotud liidesed. Kui informatsioon jõuab liidestest väravasse, siis suunatakse see edasi andmehoidla teenuse kihti, kus toimub suhtlus andmebaasiga.

Lõputöö raames realiseerib töö autor intervallide lisamise ja kustutamise funktsionaalsuse. Koodi lugemise lihtsustamiseks otsustatakse luua uus REST (*representational state transfer*) kontroller, mis koondab kõik intervallidega seotud päringud kokku ühte klassi. Kontrollerist liiguvad andmed kasutajaloo kihti ja algab kogu lisamise või kustutamise protsess. Intervalli lisamise lihtsustatud skeem on esitatud joonisel 9.



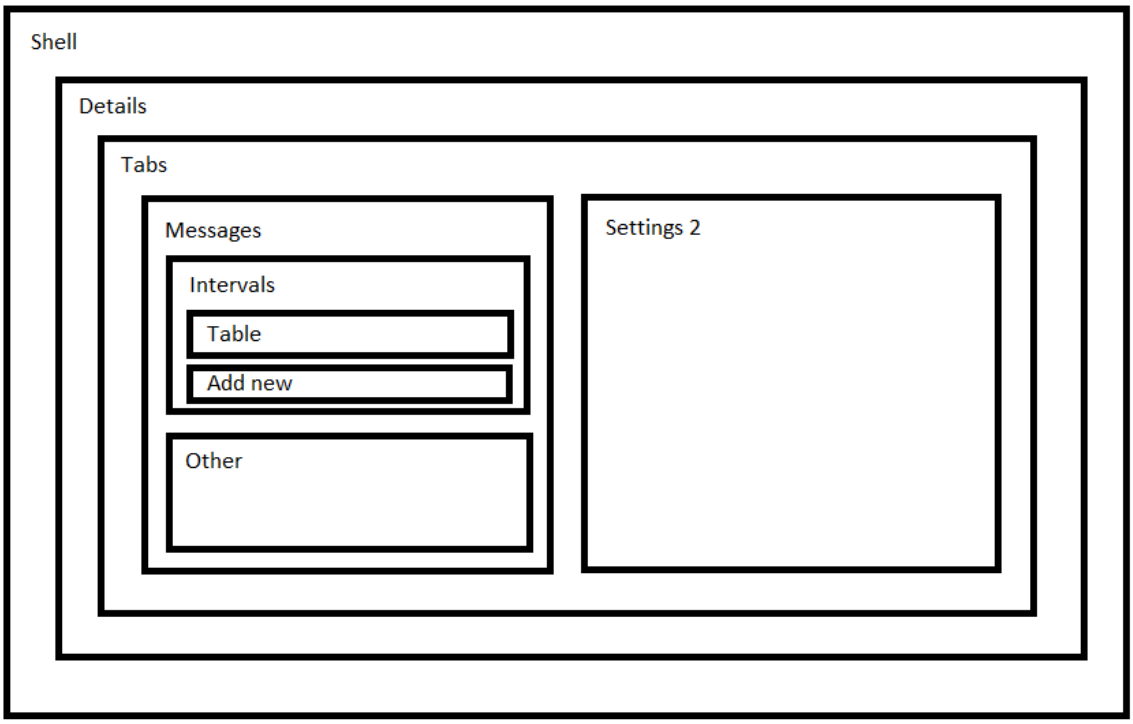
Joonis 9. Intervalli lisamise lihtsustatud skeem.

4.3 Kasutajaliidesesse vaadete loomine

Käesolevas peatükis on kirjeldatud samme uute vaadete loomiseks seoses rakenduse kasutajaliidese arendusega.

4.3.1 Komponendid

Esimeses etapis on vajalik luua kasutajaliidesesse uute vaadete jaoks vajalikud komponendid. Komponendid võimaldavad mugavalt hallata ja vajadusel taaskasutada lehel olevat infot. Vajalike komponentide leidmiseks luuakse eelneva analüüsi põhjal struktuur, mis visualiseerib vajalikke komponente vajaliku vaate ehitamiseks. Loodud visuaal on kujutatud joonisel 10.



Joonis 10. Vaate loomiseks vajalike komponentide skeem.

Vastavalt skeemile tuvastatakse, et vaate loomiseks ja kasutatavuse saavutamiseks on vajalik luua minimaalselt seitse uut komponenti. Edasi realiseeritakse need komponendid vastavalt planeeringule. Komponendid ja nende kirjeldused on esitatud tabelis 3.

Tabel 3. Kasutajaliideses vaate loomiseks vajaliku komponendid.

Komponent	**Kirjeldus**
Shell	Komponent, mis hoiab kogu antud leheosa komponente ja võimaldab ühelerakendusel navigeerimist ühest komponendist teise.
Details	Komponent, mis hoiab endas lehtede valikuid.
Tabs	Komponent, mis hoiab endas vajalike lehtede pealkirju.
Messages	Komponent, mis hoiab endas sõnumiga seotud komponente.
Intervals	Komponent, mis hoiab endas intervallide tabeli ja uue intervalli loomiseks vajalikku komponenti.
Table	Komponent, millega kuvatakse välja andmebaasi salvestatud intervallid.
Add new	Komponent, millega on võimalik lisada andmebaasi uus intervall.

4.3.2 NgRx konteiner

Konteiner peab hoidma kogu informatsiooni, mis antud komponendi kasutamiseks on vajalik. Selleks valitakse välja muutujad, mis kirjeldavad kogu olulist informatsiooni. Muutujad ja nende kirjeldused on esitatud tabelis 4.

Tabel 4. NgRx oleku haldamiseks vajalikud parameetrid.

Muutuja	**Kirjeldus**
MessageIntervalls	Väärtustatakse peale intervallide pärimist andmebaasist.
Loaded	Väärtused tõene/väär vastavalt kas tagarakenduselt on andmed edastatud.
Loading	Väärtused tõene/väär vastavalt kas toimub tagarakenduselt andmete pärimine.
SelectedIntervallId	Väärtusnumber, selekteeritud intervalli objekti ID.
SearchFilter	Väärtusklass mis loodi väärtuste filtreerimiseks.
Pager	Väärtuspaginaator mis võimaldab tabelil read jagada mitmele leheküljele.

Komponendi laadimisel on vajalik konteineri esmane initsialiseerimine. Konteineri loomiseks ja esmaseks initsialiseerimiseks kasutatud kood on esitatud joonisel 11 ja joonisel 12.

```
export interface State extends
EntityState<OfficialSettingsMessageIntervalEntity> {
  loaded: boolean;
  loading: boolean;
  selectedIntervalId?: string | null;
  searchFilter: OfficialSettingsMessageIntervalFilter;
  pager: PaginationInstance;
}
```

Joonis 11. Konteineri loomiseks kasutatud kood.

```
export const initialState: State =
officialSettingsMessageAdapter.getInitialState({
      loaded: false,
      loading: false,
      searchFilter: {
        size: 10,
        page: 0,
        sort: 'type,ASC'
      } as OfficialSettingsMessageIntervalFilter,
      pager: {
        itemsPerPage: 10,
        totalItems: 0,
        currentPage: 0,
        id: 'OfficialSettingsMessageIntervalTable'
      } as PaginationInstance
    });
```

Joonis 12. Konteineri esmane initsialiseerimine.

4.3.3 Tegevused

Tegevused käivitavad rakenduse konteineri muudatused ja vastavalt toimunud sündmusele kutsutakse neid kasutajaliideses välja. Intervallidega seotud tegevusi on kokku 14. Need jagunevad peagruppidena: pärimine, lisamine, muutmine ja kustutamine. Iga grupp koosneb eraldi veel kolmest tegevusest: peategevus, õnnestunud tegevus ja ebaõnnestunud tegevus. Nendest erinevad intervallide muutmisega seotud tegevused, sest seal on eraldi vaja määratleda ka muutmise alustamine, kinnitamine ja tühistamine. Muutmise eripärad tulenevad vajadusest hoida rakenduse konteiner vastavuses seotud sündmustega. Joonisel 13 on näide õnnestunud intervallide päringu tegevusest.

```
export const getMessageIntervalsSuccess = createAction(
  '[Official Settings] Request Message Intervals Success',
   props<{
     messages:OfficialSettingsMessageIntervalEntity[];
     pager: PaginationInstance }>()
  );
```

Joonis 13. Sõnumi intervallide päringu õnnestunud tegevus.

4.3.4 Reduktor

Reduktor kirjutatakse loodud tegevustele põhinedes. Iga aktiveeruva tegevuse peab reduktor kinni püüdma ja vastavalt ettenähtud plaanile alustama info protsessimist konteineris. Näiteks juhul kui alustatakse andmebaasist intervallide pärimist, muudame konteineri muutuja “Loading” tõeseks, sest antud hetkel meil veel infot ei ole ja saame kasutada antud muutujat kuvamaks kasutajale nähtava elemendina. Juhul kui aktiveerub intervallide pärimise õnnestumise või ebaõnnestumise tegevus, muudetakse vastavalt muutjua “Loading” konteineris tagasi vääraks. Iga tegevuse kohta kirjutatakse reduktorisse vajalikud muudatused, mis konteineris tekivad. Joonisel 14 esitatakse näitena õnnestunud intervallide päringu reduktor.

```
on(OfficialSettingsMessageActions
     .getMessageIntervalsSuccess, (state, { messages, pager
 }) =>
   officialSettingsMessageAdapter.setAll(messages, {
     ...state,
     loading: false,
     loaded: true,
     pager: {
       id: state.pager.id,
       totalItems: pager.totalItems,
       currentPage: pager.currentPage,
       itemsPerPage: pager.itemsPerPage
     } as PaginationInstance
   })
 )
```

Joonis 14. Andmebaasist intervallide pärimise õnnestumise muudatused konteineris.

4.3.5 Efektid

Efekti funktsioonid on vajalikud luua, et saaks teha asünkroonseid toiminguid. Asünkroonsed toimingud on antud moodulis vajalikud, et oleks võimalik teha päringuid, mis suhtlevad andmebaasiga. Andmebaasiga suhtlemine ei ole momentaalne. Lisaks on efektide ülesanne käivitada ka veaprotsess, juhul kui midagi ebaõnnestub seoses efekti rakendanud toimingu täitmisega. Näitena esitatakse joonisel 15 intervallide päringu efekt.

```
getMessageInterval$ = createEffect(() =>
this.action$.pipe(
    ofType(OfficialSettingsMessageActions.getMessageIntervals),
    concatMap((action) =>
      of(action).pipe(
        withLatestFrom(
          this.store.pipe(
            select(OfficialSettingsMessageSelectors
                   .getOfficialSettingsMessageSearchFilter)
          )
        )
      )
    ),
    fetch({
      run: (action, filter: OfficialSettingsMessageIntervalFilter) =>
{
        return this.officialSettingsMessageService
                   .getConfiguredMessageIntervals(filter).pipe(
          map((result:
PageableResult<OfficialSettingsMessageIntervalEntity>) =>

OfficialSettingsMessageActions.getMessageIntervalsSuccess({
              messages: result.content,
              pager: toPaginationInstance(result)
            })
          )
        );
      },
      onError: (action, error) =>
        BusinessViolationHandler.errorHandler(
          error,
          OfficialSettingsMessageActions.getMessageIntervalsFailure()
        )
    })
  )
);
```

Joonis 15. Intervallide pärimise efekt.

4.3.6 Selektorid

Selektorid on vajalikud selleks, et oleks võimalik konteinerist pärida erinevaid muutujaid. Vastavalt eelnevalt ehitatud konteinerile on vajalik luua minimaalselt kuus selektorit. Vajalikud selektorid on intervallide, laetud staatuse, laadimisel staatuse, otsingu filtri, paginaatori ja selekteeritud intervalli pärimiseks. Näidisena on joonisel 16 esitatud kõikide intervallide selektaator.

```
export const getAllConfiguredMessageIntervals =
  createSelector(
    getOfficialSettingsMessageState, selectAll
  );
```

Joonis 16. Intervallide pärimise selektaator

4.4 Arendatud funktsionaalsuse testimine

Lõputöö raames valitakse testide kirjutamiseks viimane etapp ehk „kood esimesena“ arendus. Peale sõnumi intervallide andmebaasi viimist ja nendega seotud toimingute funktsionaalsuse loomist asub lõputöö kirjutaja neid testima. Testimist alustatakse rakenduse kasutajaliidese poolel. Kasutajaliideses valitakse välja testimiseks loodud efektid, konteiner, vähendaja, valija ja uue intervalli lisamise komponendid. Komponendid ja nendele loodud testide kirjeldused on toodud tabelis 5. Kõigi komponentide esimene test oli ühesugune, mis kontrollis, et komponent on võimalik luua.

Tabel 5. Kasutajaliidese testlood.

Komponent	**Testlood**
Efektid	Kontrollime, et kui rakendub efekt andmebaasis olevate intervallide leidmiseks, siis kutsutakse välja sellele vastav teenus.
Konteiner	Kontrollime, et kui toimub sõnumi intervallide pärimine, siis uuenevad need ka konteineris. Kontrollime, et kui tuleb päring sõnumi intervallide laadimiseks, siis kutsutakse välja toiming intervallide laadimiseks. Kontrollime, et kui tuleb päring sõnumi intervalli kustutamiseks siis kutsutakse välja toiming intervalli kustutamiseks koos intervalli identifikaatoriga. Kontrollime, et kui tuleb päring sõnumi intervalli muutmise alustamiseks, siis kutsutakse välja toiming intervalli muutmise alustamiseks identifikaatoriga, mida soovitakse muuta. Kontrollime, et kui tuleb päring sõnumi intervalli muutmise tühistamiseks, siis kutsutakse välja toiming intervalli muutmise tühistamiseks. Kontrollime, et kui tuleb päring sõnumi intervalli muutmise kinnitamiseks, siis kutsutakse välja toiming intervalli muutmiseks koos uue väärtusega ja identifikaatoriga mida soovitakse muuta.
Vähendaja	Kontrollime, et kui intervallide laadimine õnnestub, siis muutub „Loaded“ tõeseks. Kontrollime, et kui intervallide laadimine õnnestub, siis muutub „Loading“ vääraks. Kontrollime, et kui intervallide laadimine ebaõnnestub, siis muutub „Loaded“ vääraks. Kontrollime, et kui intervallide laadimine ebaõnnestub, siis muutub „Loading“ vääraks.
Valija	Kontrollime, et poe algolekus oleks „Loaded“ väärtus väär. Kontrollime, et poe algolekus oleks „Loading“ väärtus väär. Kontrollime, et poe algolekus ei ole intervalle. Kontrollime, et peale intervallide laadimist on poes uuendatud intervallid.
Uue intervalli lisamine	Kontrollime, et komponendi initsialiseerimisel luuakse uus vorm intervalli sisestamiseks. Kontrollime, et kui vorm esitatakse siis kutsutakse välja tegevus intervalli lisamiseks. Kontrollime, et kui esitatakse tühi vorm siis ei kutsuta välja tegevust intervalli lisamiseks. Kontrollime, et kui esitatakse vigane vorm siis ei kutsuta välja tegevust intervalli lisamiseks.

Teises testimise etapis on vajalik lisada teste ka tagarakenduse jaoks. Tagarakenduse juures kasutab lõputöö autor integratsiooniteste. Integratsioonitestid annavad parema ülevaate, kuidas loodud funktsionaalsus käitub koos teiste seotud teenustega. Lõputöö raames oli autori osa tagarakenduse sõnumi intervallide lisamise ja kustutamise funktsionaalsuse loomine, seega loob töö autor testid ainult antud funktsionaalsusetele. Kokku luuakse neli testjuhtumit:

- Kontrollime, et uue intervalli lisamisel jõuab see andmebaasi.
- Kontrollime, et andmebaasi ei saa lisada intervalli, mis seal juba on olemas.
- Kontrollime, et intervalli kustutamisel see eemaldatakse andmebaasist.
- Kontrollime, et andmebaasist ei saa kustutada intervalli, mida seal ei ole.

Joonistel 17 ja 18 on näitena esitatud kaks testjuhtumit.

```
@Test
void newMessageIntervalIsAdded() {
  AddMessageProcessingInterval.Response response =
    addMessageProcessingInterval.execute(
    MessageType.MES1, List.of(1)
  );
  assertThat(response.errors().isEmpty()).isTrue();
}
```

Joonis 17. Uue intervalli lisamise testimine.

```
@Test
void canNotAddSameMessageInterval() {
  MessageType type = MessageType.MES1;
  List<Integer> value = List.of(1);
  addMessageProcessingInterval.execute(type, value);
  AddMessageProcessingInterval.Response response =
    addMessageProcessingInterval.execute(type, value);
  assertThat(response.errors().isEmpty()).isFalse();
}
```

Joonis 18. Intervalli lisamisel dublikaadi vea kontroll.

5 Töö tulemused ja võimalused edasiarendamiseks

Käesolevas peatükis on välja toodud lõputöö praktilise osa käigus loodud funktsionaalsuse kasutamise tulemused ja kirjeldatakse loodud funktsionaalsuse võimalikku edasiarendust.

5.1 Töö tulemus

Enne sõnumitöötluse intervallide viimist andmebaasi paiknevad need konfiguratsioonifailis. Intervallide muutmiseks on vajalik kliendil esitada teade plaanitavast rakenduse taaskäivitamise vajadusest. Saades kinnituse, et on lubatud ja võimalik rakenduse ajutine peatamine konfiguratsiooni muutmiseks, on vajalik süsteemi administraatoril, kellel on selleks vajalikud oskused ja õigused, muuta käsitsi konfiguratsioonifailis intervalle ja seejärel rakendus taaskäivitada.

Lõputöö raames viiakse sõnumitöötluse intervallid konfiguratsioonifailist andmebaasi. Loodud lahendus lubab kasutajal, kellel on selleks vajalikud oskused ja õigused, muuta intervalle, kasutades selleks veebipõhist kasutajaliidest. Intervallide muutmine on lihtsustatud ja võimalik ilma rakenduse seiskamiseta. Kasutajaliidesesse on lisatud validatsioonid, mis vähendavad võimalikke tekkivaid vigu kasutaja andmete sisestamisel, muutmisel ja kustutamisel.

5.2 Võimalused edasiarenduseks

Tänasel päeval on kasutajal võimalik sõnumitöötlemise intervalle muuta lihtsalt ja kiirelt kasutades bakalaureusetöö käigus loodud veebiliidest.

Tulevikus on võimalik loodud funktsionaalsust edasi arendada, luues kõrvale eraldi rakenduse, mis analüüsib andmevoogusid ja hindab nende alusel rakenduse ressursivajadusi erinevate sõnumitöötlemise protsesside vahel. Selline rakendus annab võimaluse kontrollida intervalle ööpäevaringselt ja sekkuda automaatselt, juhul kui rakenduse koormus vajab hajutamist.

Automaatne rakenduse koormuse jälgimine andmevoogude põhjal on kasulik edasiarendus, mis lihtsustab kasutaja tööd vähendades vajalikkust kasutajal endal

sekkuda rakenduse töösse läbi sõnumitöötlusintervallide. Protsessi automatiseerimisega kaasneks ka mitmeid riske, mis nõuavad analüüsi ja hilisemat testimist, et vähendada võimalikke probleemide ja vigade tekkimise võimalust.

Loodud funktsionaalsuse edasiarendus automatiseerimise läbi annaks võimaluse arendada analüüsi, arenduse ja testimise võimekust. Käesolevas lõputöös loodud lahendust annaks kasutada ka teiste rakenduste juures kus eksisteerib sarnane olukord. Teiste rakenduste puhul ei ole nii oluline nende kasutatavate tehnoloogiate kattuvus kui ettemõtlemine tuleviku probleemide vältimiseks läbi selleks sobivate süsteemsete parameetrite hoiustamise andmebaasi uue rakenduse arendamise algfaasis.

6 Kokkuvõte

Käesoleva töö raames on kirjeldatud kliendi rakenduse täiendamise protsessi, mille käigus liigutati sõnumitöötluse intervallid rakenduse konfiguratsioonifailist andmebaasi ja võimaldati nende reaalajas muutmine kasutades veebiliidest.

Lõputöö esimene eesmärk oli kasutajamugavuse tõstmine. Rakenduse sõnumitööltemise intervallide muutmiseks loodi kasutajaliides sobivate vaadetega, mis võimaldavad kasutajal kiirelt ja mugavalt intervalle muuta. Enam ei ole vajalik minna serverisse rakenduse kausta faili muutma ja seejärel rakendust taaskäivitada.

Teine eesmärk oli suurendada rakenduse kättesaadavuse aega. Eesmärk saavutati, vähendades rakenduse maasolekuaega, mis oli varem vajalik rakenduse taaskäivitamiseks pärast intervallide muutmist rakenduse konfiguratsioonifailis.

Kolmas eesmärk oli turvalisuse tõstmine. Tänu lõputöö raames tehtud muudatustele on rakenduse kasutajatel võimalik veebiliidese abil kiirelt ja lihtsalt sõnumitöötlemise intervalle muuta. Vigade vältimiseks lisati kasutajaliideses intervallide muutmise ja lisamise juurde ka validaatorid, mis kontrollivad sisestatud andmete õigsust. Kui failis käsitsi intervalle muuta, siis on võimalik teha mõni kirjaviga, mis ei pruugi ilmneda enne, kui rakendus püüab failist lugeda antud intervalli. Sealt võib tekkida probleem vigade analüüsil, juhul kui rakenduse töökonflikti peab lahendama keegi kolmas isik, kes ei ole kursis muudetud intervallidega.

Töö käigus loodud funktsionaalus sai valmis ning on kliendile üle antud. Funktsionaalsus läbis edukalt testimise testkeskkonnas ja on valmis sobival ajal tootesse minema. Valminud lahendus täitis etteseatud eesmärgid ja tulevikus on võimalik antud lahenduse edasiarendusena täiendada viies sisse intervallide automaatse analüüsi ja muutmise.

Kasutatud kirjandus

[1] M. Churylov, “How to Build an API: a Comprehensive Guide” [Võrgumaterjal]. Available: https://www.mindk.com/blog/how-to-build-an-api/ (vaadatud 18. märts 2023)

[2] Ana Djurovic “20+ Astonishing Agile Adoption Statistics for 2023” [Võrgumaterjal]. Available: https://goremotely.net/blog/agile-adoption/ (vaadatud 18. märts 2023)

[3] P. Parmakson, „Agiilne arendus“ [Võrgumaterjal]. Available: https://agiil.github.io/IT/Agiil (vaadatud 18. märts 2023)

[4] Buildd, „Story Grooming: Your Guide to Agile Backlog Grooming“ [Võrgumaterjal]. Available: https://buildd.co/product/agile-story-backlog-grooming (vaadatud 22. aprill 2023)

[5] Simplilearn „What is Planning Poker: Understanding the Consensus-based Estimating Technique“ [Võrgumaterjal]. Available: https://www.simplilearn.com/what-is-planning-poker-article (vaadatud 22. aprill)

[6] The PostgreSQL Global Development Group, „What is PostgreSQL?“ [Võrgumaterjal]. Available: https://www.postgresql.org/about/ (vaadatud 27. märts 2023)

[7] Integrate.io, „PostgreSQL vs MySQL: The Critical Differences“ [Võrgumaterjal]. Available: https://www.integrate.io/blog/postgresql-vs-mysql-which-one-is-better-for-your-use-case/ (vaadatud 27. märts 2023)

[8] Knoldus, „Liquibase Features and Advantages“ [Võrgumaterjal]. Available: https://blog.knoldus.com/liquibase-features-and-advantages/ (vaadatud 27. märts 2023)

[9] E. Kalar, „The Benefits of Using Liquibase Over a Custom Built Database Tool“ [Võrgumaterjal]. Available: https://www.liquibase.com/blog/using-liquibase-vs-custom-built-database-tool (vaadatud 22. aprill 2023)

[10] R. Winter, „10 Best Java Frameworks 2023“ [Võrgumaterjal]. Available: https://blog.hubspot.com/website/java-frameworks (vaadatud 25. märts 2023)

[11] T. Hombergs, „Hexagonal Architecture with Java and Spring“ [Võrgumaterjal]. Available: https://reflectoring.io/spring-hexagonal/ (vaadatud 26. märts 2023)

[12] E. Schaffer, „Hexagonal architecture tutorial: Build maintainable web apps“ [Võrgumaterjal]. Available: https://www.educative.io/blog/hexagonal-architecture-tutorial (vaadatud 7. mai 2023)

[13] T. K. Dogantekin, „Hexagonal (Ports & Adapters) Architecture“ [Võrgumaterjal]. Available: https://medium.com/idealo-tech-blog/hexagonal-ports-adapters-architecture-e3617bcf00a0 (vaadatud 7. mai 2023)

[14] D. Frak, „Ports & Adapters (aka hexagonal) architecture explained“ [Võrgumaterjal]. Available: https://codesoapbox.dev/ports-adapters-aka-hexagonal-architecture-explained/ (vaadatud 17. aprill 2023)

[15] D. Rajput, „A Practical Example of Hexagonal Architecture in Java“ [Võrgumaterjal]. Available: https://www.dineshonjava.com/a-practical-example-of-hexagonal-architecture-in-java/ (vaadatud 30. märts 2023)

[16] Angular, „What is Angular?“ [Võrgumaterjal]. Available: https://angular.io/guide/what-is-angular (vaadatud 25. märts 2023)

[17] R. Salnik, „Angular vs React. Which JS Framework Is Better in 2022?“ [Võrgumaterjal]. Available: https://brocoders.com/blog/angular-vs-react/ (vaadatud 25. märts 2023)

[18] L. Onikadze „A Deep Dive Into NgRx Advantages and Features“ [Võrgumaterjal]. Available: https://www.toptal.com/angular/why-use-ngrx (vaadatud 22. aprill 2023)

[19] NGRX, „Getting Started“ [Võrgumaterjal]. Available: https://ngrx.io/guide/store (vaadatud 18. märts 2023)

[20] A. Dua, „Unit Testing vs Integration Testing: 4 Key Differences Explained“ [Võrgumaterjal]. Available: https://www.turing.com/blog/unit-testing-vs-integration-testing-key-differences-explained/ (vaadatud 7. mai 2023)

[21] V. Khorikov, „To TDD or not to TDD“ [Võrgumaterjal]. Available: https://enterprisecraftsmanship.com/posts/tdd-not-tdd/ (vaadatud 15 aprill 2023)

[22] K. Vijay, „What is Junit and How it Works? An Overview and Its Use Cases“ [Võrgumaterjal]. Available: https://www.devopsschool.com/blog/what-is-junit-and-how-it-works-an-overview-and-its-use-cases/ (vaadatud 16. aprill 2023)

[23] Pankaj, „Mockito Tutorial“ [Võrgumaterjal]. Available:https://www.digitalocean.com/community/tutorials/mockito-tutorial (vaadatud 16. aprill 2023)

[24] Xoriant, „Jest – Fast and Furious Unit Testing Framework“ [Võrgumaterjal]. Available: https://www.xoriant.com/blog/jest-fast-and-furious-unit-testing-framework (vaadatud 16. aprill 2023)

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

Mina, Gert Kuldma

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Sõnumirakenduse intervallide salvestamine andmebaasi“ mille juhendaja on Kristiina Hakk
   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.