TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kristjan Keskküla 212890IADB

Ühtse euromaksete piirkonna maksete küsimise skeemi teenusepakkuja tarkvara

Bakalaureusetöö

Juhendaja: Toomas Lepikult
PhD

Tallinn 2024

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: Kristjan Keskküla

04.01.2024

Annotatsioon

Antud töös käsitletakse ühtse euromaksete piirkonna maksete küsimise skeemi teenusepakkuja lahenduse arendamist. Töös tutvustatakse lahenduse protsessialastest ning tehnilisest piirangutest tulenevaid nõudeid (dokumentatsiooni analüüs), luuakse tarkvarasüsteemi kontseptsioon ning selgitatakse rakenduse arendamisel tehtud valikuid nõuete kontekstis.

Töö tulemus on maksete küsimise skeemi teenusepakkuja tarkvara prototüüp, mis võimaldab edastada maksete küsimise skeemi sõnumeid ning on vastavuses Euroopa Maksenõukogu poolt kehtestatud reeglitega.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 62 leheküljel, 6 peatükki, 45 joonist, 8 tabelit.

Abstract

Single Euro Payments Area Request-to-Pay Scheme Service Provider Software

The body of work in the thesis involves the development of a Request-to-Pay (RTP) service provider solution for the Single Euro Payment Area (SEPA). The thesis describes the requirements driven by the procedural and technical limitations (documentation analysis), concerns the creation of the software system conception and explains the choices made during the development of the prototype according to the derived requirements.

The result of the work is a software prototype of the Request-To-Pay (technical) service provider, which enables the transmission of SEPA Request-to-Pay messages and complies with the rules established by the European Payments Council.

The thesis is in Estonian and contains 62 pages of text, 6 chapters, 45 figures, and 8 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface* – rakendusliides
BLL	*Business Logic Layer* – äriloogika ning teenuste kiht
CRUD	*Create, Read, Update, Delete* **–** andmetega suhestumise põhitegevused
Callback URL	Tagasikuulde ressursi aadress (lokaator). Kindla ressursi, operatsioonide jaoks loodud (ajutine) aadress võimaldamaks REST rakendusliidese puhul asünkroonset kommunikatsiooni
DAL	*Data Access Layer* – andmejuurdepääsukiht
DI	*Dependency Injection; Inversion Of Control* **–** sõltuvuste pööratud kasutus
EPC	*European Payments Council* – Euroopa Maksenõukogu
HATEOAS	*Hypermedia As The Engine of Application State*
HTTP	*The Hypertext Transfer Protocol* – Hüperteksti edastusprotokoll
JSON	*JavaScript Object Notation* – JavaScripti objektide notatsioon
OSM	*Operational Scheme Manager* – tegevuskava haldur
POCO	*Plain Old C(#) Object* – C# objekt (klass)
QSealC	*Qualified Electronic Seal Certificate* – kvalifitseeritud elektroonilise pitseri sertifikaat (ETSI EN 319 412-3)
QWAC	*Qualified Web Authentication Certificate* – kvalifitseeritud veebiautentimissertifikaat (ETSI EN 319 412-4)
REST	*Representational state transfer* – tarkvaraarhitektuuri muster, mis seab veebirakenduse loomisel kindlad piirid
RfC	*Request for Cancellation* – tühistamispalve
SRTP/RTP	*Single Euro Payments Area Request-to-Pay* – Ühtse euromaksete piirkonna maksete küsimise skeem
SEPA	*Single Euro Payments Area* – Ühtne euromaksete piirkond
TLS	*Transport Layer Security* – transpordikihi turve
UI	*User Interface* – kasutajaliides (viidatud rakenduses kui API)
URL	*Unified Resource Locator* – universaalne ressursilokaator
XML	*eXtensible Markup Language* – laiendatav märgistuskeel
XSD	*XML Schema Definition* – XML skeemdefinitsioon

Sisukord

1 Sissejuhatus 11
1.1 Probleem 11
1.2 Eesmärk 12
1.3 Erisused ja töö skoop 12
2 Maksete küsimise skeem 13
2.1 Taustsüsteem 14
3 Metoodika 15
3.1 Dokumentatsiooni analüüs 15
3.1.1 Maksete küsimise skeemi reeglite raamat 15
3.1.2 Maksete küsimise skeemi rakendusjuhised 19
3.1.3 Maksete küsimise skeemi rakendusliidese spetsifikatsioonid 20
3.1.4 Rakendusliidese turvaraamistik 22
3.2 Funktsionaalsed ja mittefunktsionaalsed nõuded. Piirangud. 24
3.3 Programmeerimismeetodite analüüs 25
3.3.1 Programmeerimismeetodite ja raamistike valik 26
3.4 Rakendusliideste parimate praktikate analüüs 27
3.4.1 Rakendusliidese rakendamise parimate praktikate valikud 28
4 Tarkvara realiseerimise kontseptsioon 31
4.1 Süsteemi konteksti vaade 31
4.2 Konteineri vaade 32
4.3 Teenusepakkuja konteineri komponentvaade 33
4.3.1 Andmebaasi kasutamine ning programmi struktuur 34
4.3.2 SRTP teenusepakkuja teenuste kirjeldus 35
4.4 Rakendusliideste disain 36
5 Rakendus ja tulemused 38
5.1 Kasutatud tehnoloogiad ja tööriistad 39
5.2 Rakenduse koodi kirjutamise põhimõtted 40

5.3 Rakenduse koodivaramu struktuur 41
5.4 Andmejuurdepääsukiht (DAL) 42
5.4.1 Andmete olemid 42
5.4.2 Repositooriumid ja objektrelatsiooni kaardistamine 44
5.5 Äriloogikakiht (BLL) 46
5.5.1 Tulemuste vahendamine teenuskihil ja veateate struktuur 46
5.5.2 Valideerimise teenus 47
5.5.3 Taustal toimuvad teenused – olekureporter ja olekujälgija 55
5.5.4 Sõnumihalduri ja marsruutimise teenused. Konnektorid. 56
5.6 Rakendusliidese ehk kasutajaliidese kiht (API / UI) 60
5.6.1 Veebiserver Kestrel 60
5.6.2 Autentimine ja autoriseerimine 62
5.6.3 Logimine 65
5.6.4 HATEOAS rakendamine 68
5.6.5 Asünkroonsed vood. Tagasikuulde aadress (*Callback URL*) 69
5.6.6 Rakendusliidese dokumentatsioon 71
6 Kokkuvõte 73
Kasutatud kirjandus 74
Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks 77
Lisa 2 – Maksete küsimise protsessi töövoo viieteistkümne põhilise sammu kirjeldus. 78
Lisa 3 – RTP tühistamispalve protsessi detailne kirjeldus 81
Lisa 4 – Funktsionaalsed ja mittefunktsionaalsed nõuded detailides 83
Lisa 5 – EFBaseRepository klassi rakendamise kood 86
Lisa 6 – CertAuthService autoriseerimisteenuse kood 87
Lisa 7 – Marsruuditava sõnumi loomise teenuse kood 89
Lisa 8 – Teenusepakkuja konnektori kood 91
Lisa 9 – Logimise vahevara kood 93

Jooniste loetelu

Joonis 1 – 4 nurga mudel („4 corner model“) 13
Joonis 2 – Maksete küsimise skeemi protsessi komponendid 16
Joonis 3 – Maksete küsimise skeemi peamine protsess 17
Joonis 4 – Maksete küsimise olekuraporti küsimise protsess 18
Joonis 5 – Maksete küsimise skeemi tühistamispalve protsess 19
Joonis 6 – Süsteemi kontekstvaade 32
Joonis 7 – Süsteemi konteinervaade 32
Joonis 8 – Süsteemi komponentvaade 34
Joonis 9 – Rakendatud prototüüpsüsteemi ülevaade 38
Joonis 10 – Teenusepakkuja tarkvara instantsi konfiguratsioon 39
Joonis 11 – Sertifikaadi olemi koodinäide 43
Joonis 12 – Olemiidentifikaatori abstraktse klassi koodinäide 43
Joonis 13 – Baasrepositooriumi liidese koodinäide 44
Joonis 14 – Sertifikaadirepositooriumi liidese koodinäide 45
Joonis 15 – Sertifikaadirepositooriumi koodinäide 45
Joonis 16 – Teenuste tulemuste objekt (ServiceResult) koodis 47
Joonis 17 – Veateate objekti kood 47
Joonis 18 – Ühe tavalise rakendusliidese sõnumi struktuur 48
Joonis 19 – Ühe tavalise rakendusliidese kontrolleri valideerimiskood 49
Joonis 20 – Sõnumite valideerimisteenuse kood 50
Joonis 21 – RTP sõnumi valideerimine skeemidefinitsiooni vastu 51
Joonis 22 – Spetsifikatsiooni mustri koodinäide äriloogika kontrollimiseks 52
Joonis 23 – Spetsifikatsiooni mustri kasutus maksetüübi valideerimisel 53
Joonis 24 – Spetsifikatsioonide sidumine läbi Booli loogika maksetüübi kontrollil 54
Joonis 25 – Perioodilise SRTP tühistamise taustateenuse kood 55
Joonis 26 - Marsruuditava sõnumi liides (leping) 57
Joonis 27 - Uue RTP päringu objekt (RtpRequest) 57
Joonis 28 – IMessageHandlerService sõnumihalduri liides 58
Joonis 29 – Marsruuditava sõnumi loomisteenuse liides 58

Joonis 30 - Marsruuteri tööpõhimõtte kirjeldus 59
Joonis 31 – Konnektorteenuse leping 60
Joonis 32 – Klientsertifikaadi autentimine ja Host-päise põhine sertifikaadi valik 61
Joonis 33 – Sertifikaadipõhise autoriseerimise veebiserveri vahevara kood 63
Joonis 34 – Poliitikapõhise nõuete halduri autoriseerimise kood 64
Joonis 35 – Poliitikapõhise autoriseerimiseteenuse konfiguratsioon 64
Joonis 36 – Ressursipõhise autoriseerimise poliitika rakendamine kontrolleris 65
Joonis 37 – Teenusepakkujate vaheliste rakendusliideste veateate struktuur 66
Joonis 38 – Rakendusliidese veahalduse sõnumite vabriku kood 67
Joonis 39 – HATEOAS teenuse konfiguratsioon 68
Joonis 40 – HATEOAS teenus rakendusliidese kontrolleris 68
Joonis 41 – HATEOAS sõnumi näide 69
Joonis 42 – Tagasikuulde aadressi (Callback URL) loomise koodinäide 70
Joonis 43 – Tagasikuulde aadressi (Callback URL) kontrolleri koodinäide 70
Joonis 44 – Rakendusliidese kontrolleri koodipõhine dokumenteerimine 71
Joonis 45 – Automaatselt loodud rakendusliidese dokumentatsiooni näide 72

Tabelite loetelu

Tabel 1 – Teenusepakkujate vahelise standardse rakendusliidese määratlus 21
Tabel 2 – Teenusepakkujate vahelise rakendusliidese HTTP vastuste olekukoodid 22
Tabel 3 – Programmeerimiskeelte ja raamistike võrdluse tabel 27
Tabel 4 – SRTP teenusepakkuja teenuste kirjeldused 35
Tabel 5 – Makse küsija ning maksja teenusepakkuja ressursid ja meetodid määratlus . 37
Tabel 6 – Maksete küsimise töövoo viieteistkümne põhilise sammu kirjeldus 78
Tabel 7 – RTP tühistamispalve (RfC) protsessi detailne kirjeldus 81
Tabel 8 – Funktsionaalsete ja mittefunktsionaalsete nõuete detailid 83

1 Sissejuhatus

Vaid 25 aastat tagasi tuli elektriarve maksmiseks minna pangakontorisse või lausa energiaettevõttesse kohale. Oli aeg kui enamikes poodides sai maksta vaid sularahas. Vahepeal on pangandus ja maksesüsteemid arenenud jõudsalt edasi ning tänapäeval on internetipank ja kaardimakse meile iseenesestmõistetavad. Maksekeskkond on pidevas arengus ning uued innovaatilised lahendused tulevad tehnoloogia arenedes turule. Alles viimati lisandusid välkmaksed, viipemaksed ja avatud pangandusel põhinevad maksed [1].

Antud lõputöö keskendub Euroopa Maksenõukogu poolt välja töötatud uuele maksesüsteemile maksete küsimise skeemile. Lõputöö raames analüüsitakse maksete küsimise skeemi teenusepakkuja lahendust, analüüsile põhinevalt luuakse tarkvara kontseptsioon ning arendatakse välja maksete küsimise skeemi teenusepakkuja prototüübi tarkvara.

Töö käigus otsitakse vastust küsimusele: „kuidas luua vastavalt parimatele praktikatele maksete küsimise skeemi teenusepakkuja tarkvara, mis on vastavuses Euroopa Maksenõukogu poolt määratud nõuetega?“

1.1 Probleem

Euroopa Maksenõukogu tutvustas maksete küsimise skeemi kontseptsiooni 15. juunil 2021 [2]. 2023. aasta lõpu seisuga pole skeemi veel kasutusele võetud, teadaolevalt puudub laiem konsensus skeemi rakendamise tuleviku plaanidest ning järgmistest sammudest.

Euroopa Maksenõukogu on töötanud välja skeemi töötamiseks vajaliku reeglistiku ning tehnilised elemendid (sh. andmestruktuurid ja sõnumid), mille rakendamisel skeemi osalised võimaldavad makse küsijal maksjalt makset küsida. Samuti on loodud sõnumeid vahendavatele teenusepakkujatele võimalus tõestada enda võimekust skeemis sõnumeid vastavalt reeglitele vahendada. Selleks on ametlik protsess, reeglite raamatus kirjeldatud „homologeerimise“ protsess.

Euroopa Maksenõukogu osaliste registri järgi on 2023. aasta lõpu seisuga skeemil kolm osalist, kuid laiemat kandepinda skeem ei ole saavutanud. Eksisteerib üks ametliku valideerimise protsessi läbinud tehniline teenusepakkuja – Hispaaniast pärit *Iberpay* [3]. Teadaolevalt puudub avaliku lähtekoodiga tehnilise teenusepakkuja tarkvara, samuti puuduvad teadmised skeemi rakendamisest elulistes protsessides.

Maksete küsimise skeemi funktsiooniks on maksesoovi sõnumite vahendamine. Kuid lisaks sõnumite edastamisele on edukaks maksetehingu sooritamiseks vajalikud mitmed erinevad teised komponendid näiteks tehnoloogiad ja tööriistad, et siduda kaupmehi (makse küsijaid), kliente (maksjaid) kui makse läbiviijaid (näiteks panku). Et oleks võimalik neid tehnoloogiaid ja tööriistu välja arendada on vajalik maksete küsimise skeemi sõnumite töötlemiseks standardiseeritud tarkvara – maksete küsimise skeemi teenusepakkuja tarkvara. Tarvilik on luua maksete küsimise skeemi teenusepakkuja liides, mis võimaldab edastada maksete küsimise skeemi sõnumeid ning on vastavuses Euroopa Maksenõukogu poolt kehtestatud reeglitega (maksete küsimise reeglite raamatu, rakendamise juhiste ja rakendusliidese spetsifikatsiooni ning turvaraamistikuga).

1.2 Eesmärk

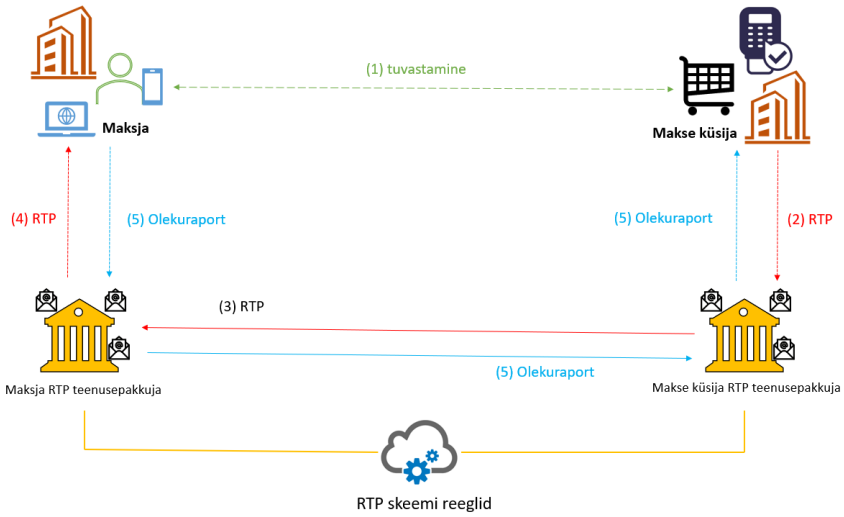
Luua maksete küsimise skeemi (RTP) teenusepakkuja tarkvara prototüüp, mis võimaldab edastada maksete küsimise skeemi sõnumeid ning on vastavuses Euroopa Maksenõukogu poolt kehtestatud reeglitega (maksete küsimise reeglite raamatu, rakendamise juhiste ja rakendusliidese spetsifikatsiooni ning turvaraamistikuga).

1.3 Erisused ja töö skoop

Töö hulka ei kuulu andmebaaside, andmehulkade ega sõnumite detailne kirjeldamine. Sõnumites kasutatavad andmehulgad ja atribuudid on detailselt kirjeldatud Euroopa Maksenõukogu poolt väljastatud rakendusjuhistes [4] [5] [6].

2 Maksete küsimise skeem

Maksete küsimise skeem (RTP) on nelja-osapoole makseskeem („4 nurga mudel“ – Joonis 1) [7] mille osapooled on makse küsija, makse küsija teenusepakkuja, maksja ja maksja teenusepakkuja. Skeem võimaldab potentsiaalsel makse saajal (näiteks e-pood) kasutada maksja andmeid näiteks maksja nime ja rahvusvahelist pangakontonumbrit (IBAN), et alustada makse küsimise protsessi läbi maksesoovi edastamise teenusepakkujale. Küsijat toetab makse küsija teenusepakkuja, kelle ülesandeks on tõlkida maksesoovis sisalduv sõnum õigesse vormi, kontrollida sõnumi korrektsust ning edastada soov maksja teenusepakkujale. Maksjat toetav teenusepakkuja sooritab omakorda kontrollid, korrektsuse korral edastab maksesoovi maksjale. Maksja võib kinnitada maksesoovi või keelduda makse tegemisest. Kui maksja peaks nõustuma maksesooviga, siis maksja teenusepakkuja edastab kinnitussõnumi makse küsija teenusepakkujale, kes omakorda teavitab makse küsijat. Maksja seejärel võib makse sooritada olenevalt makseinstrumendist või makseskeemist, näiteks SEPA kiirkrediitkorralduse abil. Tähtis on usaldus, et maksekorraldus ka tegelikult ellu viiakse. Nimelt maksete küsimise skeem tegeleb vaid sõnumite juhtimisega, kliiring ja arveldamine toimub väljaspool skeemi. Seetõttu peavad maksekorralduse täitjad, arveldavad (finantsasutused) olema maksete küsimise skeemi „homologeeritud“ liikmed.



Joonis 1 – 4 nurga mudel („4 corner model“)

Maksete küsimise skeemi funktsiooniks on sõnumite vahendamine. Lisaks sõnumite edastamisele on edukaks maksetehingu sooritamiseks vajalikud mitmed erinevad teised

komponendid näiteks tehnoloogiad ja tööriistad, et siduda kaupmehi (makse küsijaid), kliente (maksjaid) kui makse tegelike arveldajaid (näiteks panku).
RTP tööd võimaldavad teenusepakkujad, kelle ülesandeks on toimida lüüsina (adresseerida ja marsruutida sõnumeid) makse saaja ja maksja maksekorralduse vahendamisel.

2.1 Taustsüsteem

Euro on Euroopa Liidu kahekümne liikmesriigi ametlik valuuta. Neid riike kus ametlik valuuta on euro tuntakse eurotsooni või euroala nime all ning see hõlmab 344 miljonit elanikku 2023. aastal. Väljaspool Euroopa Liitu kasutavad eurot valuutana ka erinevad eriterritooriumid [8].

Euro võeti esmakordselt kasutusele 1. jaanuaril 1999 aastal üheteistkümne osapoole poolt. Euro üleste maksete paremaks haldamiseks ja opereerimiseks osapoolte vahel loodi sajandi alguses SEPA ehk ühtse euromaksete piirkonna initsiatiiv. Ühtse euromaksete piirkonna loomise eesmärgiks on teha sularahavabasid euro makseid kiiresti, turvaliselt ja efektiivselt ning luua üks maksete süsteem kogu euroala jaoks. Ühtne euromaksete piirkond alustas tööd krediitkorralduste (SEPA Credit Transfer) jaoks 2008. aastal, järgnesid deebetkorraldused (SEPA Debit Transfer) 2009. aastal ning täielikult rakendati initsiatiiv 2014. aastal. 2023. aastal on 36 Euroopa riiki ühtse euromaksete piirkonna liikmed [9].

SEPA paketis olevad erinevad funktsioonid on jagatud eraldi makseskeemideks. Makseskeemidele rakenduvad kindlad regulatiivsed ning töökorralduslikud nõuded. Näiteks SEPA kiirkrediitkorralduste (SEPA Credit Transfer Inst) näeb ette saaja kohese krediteerimise, tehingu viivitused peavad olema alla kümne sekundi ühtse euromaksete piirkonna alal ning pakkuma teenust 24/7/365. Skeemide, nende liikmelisuse haldamiseks on makseteenuse pakkujaid ühendav Euroopa Maksenõukogu (EPC) [10].

Lisaks olemasolevatele skeemidele on Euroopa Maksenõukogu töötanud uute innovaatiliste teenuste ning makseskeemide kallal (näiteks avatud pangandus) ning loodab tulevikus rakendada uut maksete küsimise skeemi (RTP).

Maksete küsimise skeem (RTP) on loodud eesmärgiga muuta Euroopas maksed lihtsamaks, mugavamaks ning turvalisemaks.

3 Metoodika

Maksete küsimise skeem on standardiseeritud Euroopa Maksenõukogu poolt ning suur osa teenusepakkuja tööst on reguleeritud. Reguleerimise aluseks on neli funktsiooni alusel määratletud põhidokumenti: maksete küsimise reeglite raamat [7], rakendamise juhiseid [4] [5] [6], rakendusliidese spetsifikatsioon [11] ning turvaraamistik [12].

Lahendusmetoodika hõlmab dokumentatsiooni analüüsi mille läbi leitakse teenusepakkuja lahenduse funktsionaalsed ja mittefunktsionaalsed nõuded ning piirangud. Vastavalt nõuetele analüüsitakse ning dokumenteeritakse rakendusliideste arendamise parimaid põhimõtteid, samuti programmeerimismeetodite parimaid praktikaid, luuakse ja dokumenteeritakse tarkvaralahenduse kontseptsioon

Lõpptulem on maksete küsimise skeemi teenusepakkuja prototüübi loomine ehk kontseptsiooni ellu panek arvestades nõudeid, tehtud valikuid ja parimaid praktikaid.

3.1 Dokumentatsiooni analüüs

Järgnevalt analüüsitakse maksete küsimise skeemi puudutavat dokumentatsiooni.

3.1.1 Maksete küsimise skeemi reeglite raamat

Maksete küsimise skeemi reeglite raamat [7] kirjeldab ära reeglid, praktikad ja standardid mis võimaldavad maksete küsimise skeemi teenuspakkujatel liituda, osaleda ja hallata maksete küsimise skeemi.

Maksete küsimise skeemi reeglite raamatu eesmärkideks on:

- olla üks ja ainus lähteallikas skeemi reeglite ja kohustuste jaoks.
- üheselt ära kirjeldada skeemi osalistele ja teistele osapooltele kuidas skeem funktsioneerib.

Reeglite raamat määrab ära ärilised nõuded ja maksete küsimise skeemi teenusepakkujate vahelise suhtluse reeglid.

Reeglite raamatus on ära kirjeldatud maksete küsimise skeemis kasutatavad äriprotsessid, äriprotsessi sammud (PS-*xx*-*yy*), andmehulgad (DS-*xxx*) ja atribuudid (AT-*xxxx*).

Maksete küsimise protsessi võib kirjeldada nelja komponendina (Joonis 2):

Joonis 2 – Maksete küsimise skeemi protsessi komponendid

Reeglite raamatu fookus on „*Request-to-Pay*“ ja „Aktsepteerimine/keeldumise“ osadel. Kommertstehing ja maksetehing toimuvad väljaspool skeemi ning nende jaoks on vaja eraldi lahendusi.

Osapooled ja rollid

Reeglite raamat sätestab neli erinevat rolli, kes osalevad maksete küsimise skeemi sõnumite saatmisel. Need rollid on kirjeldatud ka Joonisel 1.

1. **Makse küsija** – kes RTP protsessi alustab, potentsiaalne kasusaaja.
2. **Makse küsija RTP teenusepakkuja** – makse küsija poolne sõnumite haldur.
3. **Maksja** – kellele RTP on adresseeritud, kasuandja.
4. **Maksja RTP teenusepakkuja** – maksja poolne sõnumite haldur.

Protsessi üldine kirjeldus

Makse küsimise protsess sisaldab viit põhilist sammu:

1. **Maksja identifitseerimine** – maksja ja maksja teenusepakkuja identifikaatori välja selgitamine.
2. **SRTP saatmine makse küsijalt makse küsija teenusepakkujale** – makse kohta pärinev informatsioon pakitakse standardsele SRTP sõnumi kujule ning edastatakse makse küsija teenusepakkujale. Toimub kontroll ja valideerimine.

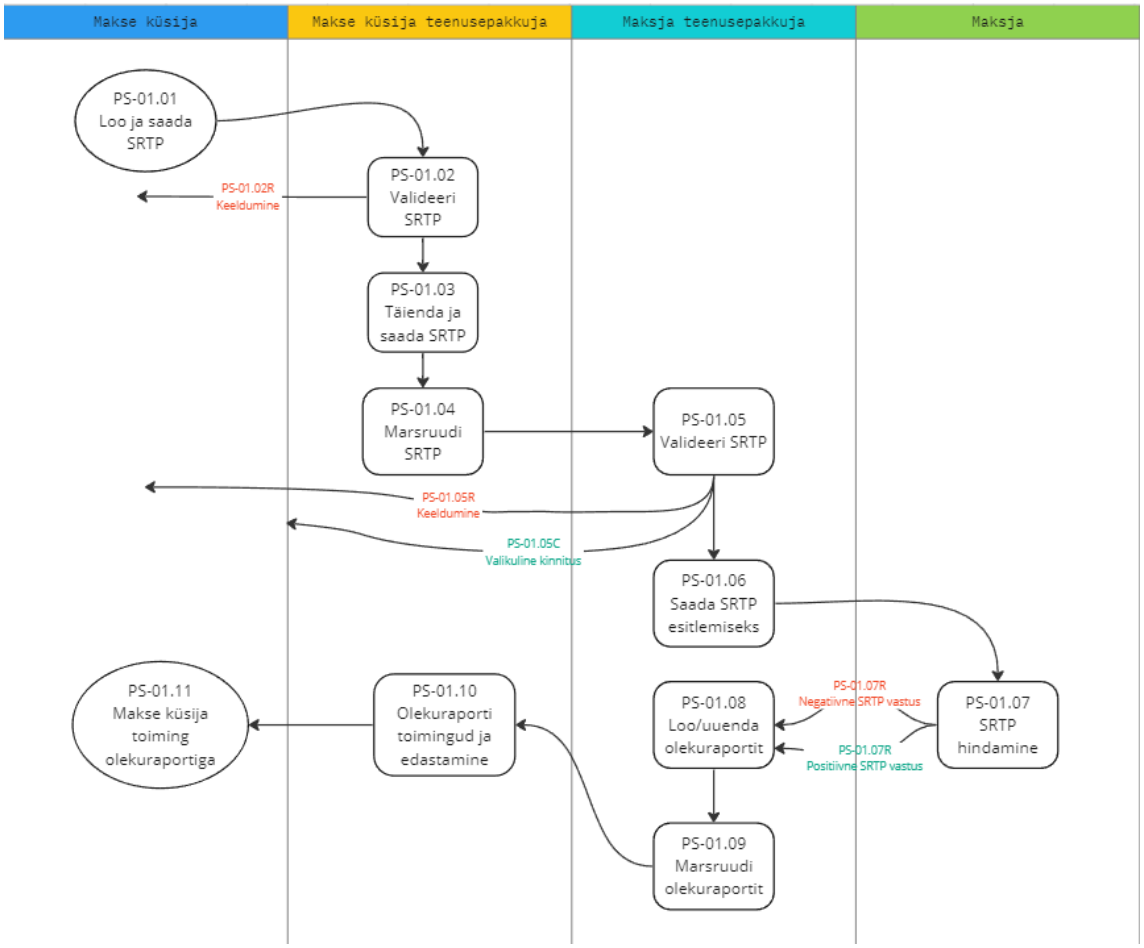
3. **SRTP saatmine makse küsija teenusepakkujalt maksja teenusepakkujale** – makse küsija teenusepakkuja edastab SRTP sõnumi maksja teenusepakkujale. Toimub kontroll ja valideerimine.

4. **SRTP esitlus maksjale** – makse küsimise sõnum esitletakse maksjale läbi tema valitud viisi (nt. veebibrauser, nutitelefon). Maksja saab aktsepteerida või keelduda maksesoovist.

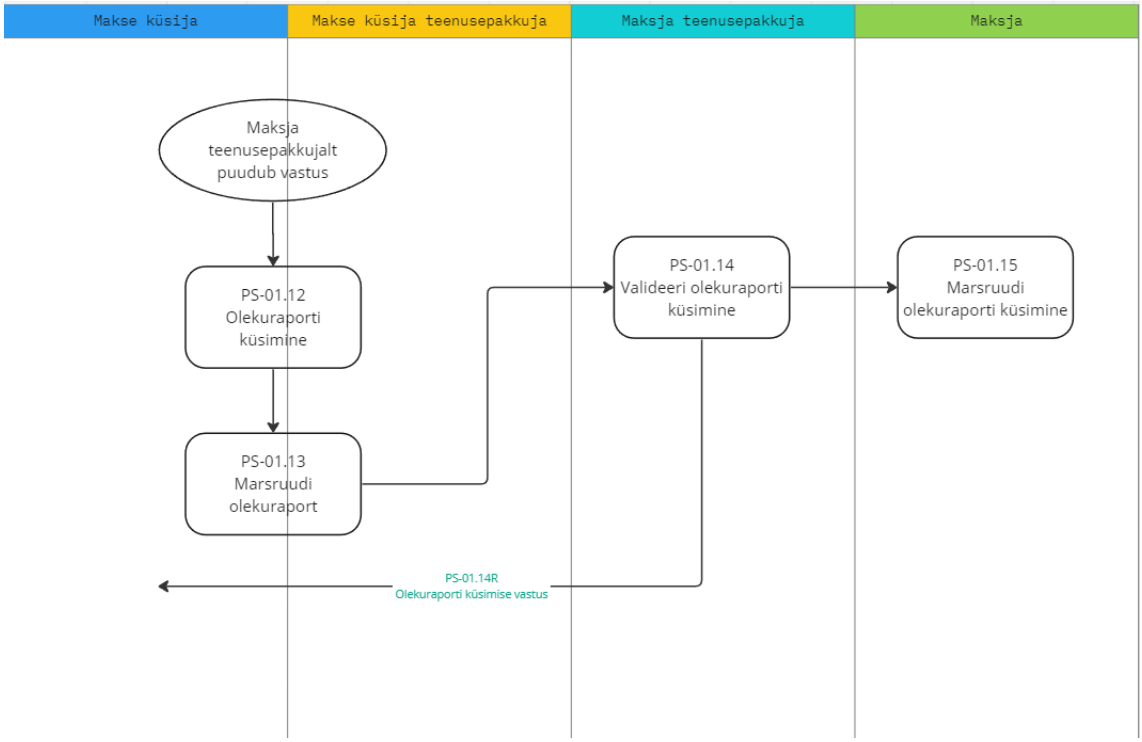
5. **Oleku raporti saatmine** – maksja poolt langetud SRTP aktsepteerimise või keeldumise otsus edastatakse läbi teenusepakkujate makse küsijale.

Protsessi sammud

Järgnevalt on kirjeldatud viisteistkümmet põhilist sammu mida üks makse küsimise töövoog sisaldab. Joonisel 3 ja joonisel 4 on välja toodud protsesside omavaheline suhestus töövoo diagrammil. Detailsema kirjelduse protsessidest leiab lisast Tabelist 2.



Joonis 3 – Maksete küsimise skeemi peamine protsess

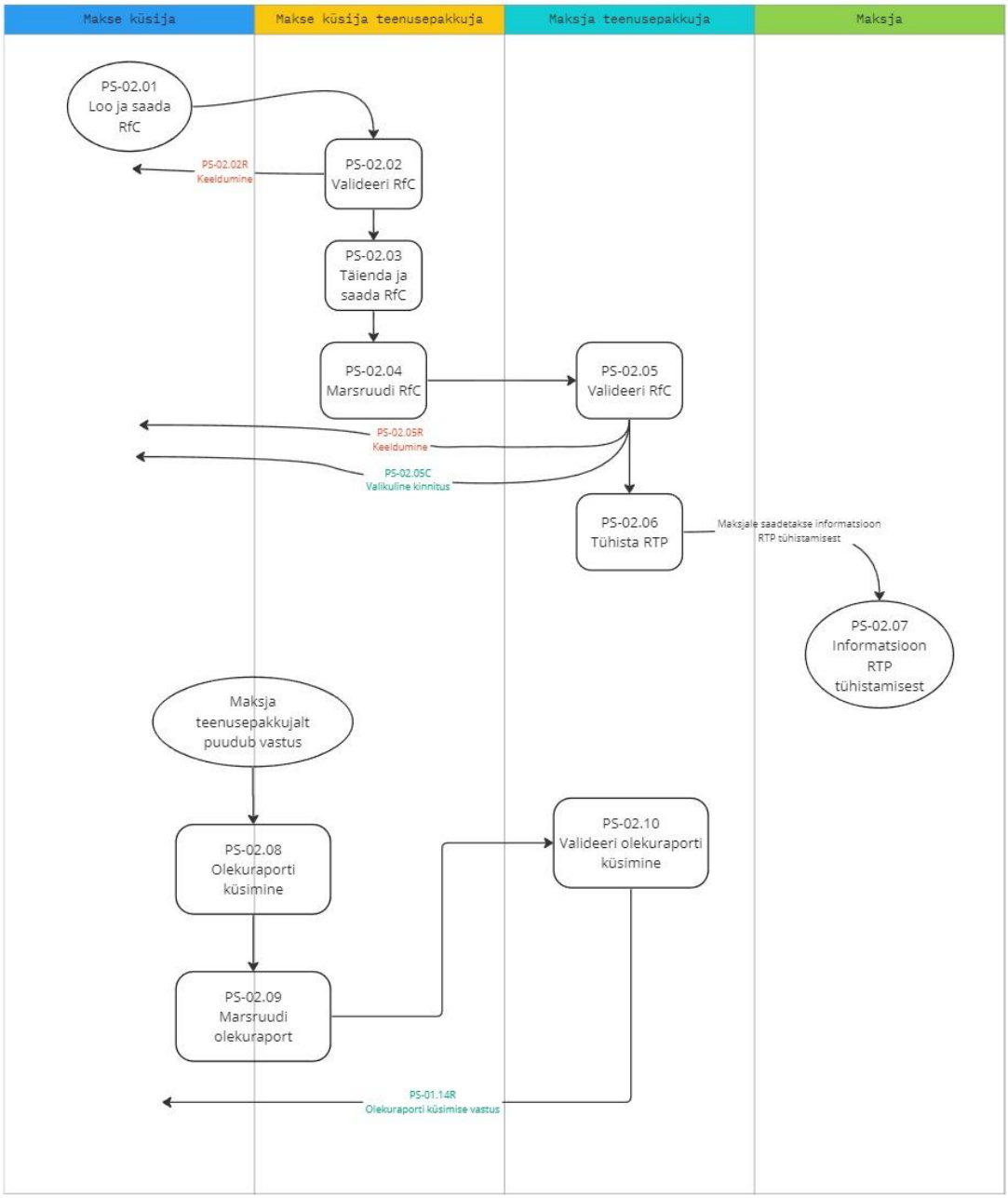


Joonis 4 – Maksete küsimise olekuraporti küsimise protsess

Tühistamispalve protsess

Lisaks maksete küsimise protsessile on tähtis ka tühistamise protsess. Tühistamispalve protsessi (*Request-for-Cancellation ehk RfC*) puhul makse küsija saadab makse küsimise tühistamispalve. Tühistamispalvet on võimalik täita kuni RTP sõnumis määratud aegumistähtajani. Tühistamise puhul maksesoovi ei täideta.

Detailse kirjelduse tühistamisprotsessi erinevatest sammudest leiab lisast tabelist nr. 3. Joonisel 5 on kirjeldatud tühistamispalve protsessi.



Joonis 5 – Maksete küsimise skeemi tühistamispalve protsess

3.1.2 Maksete küsimise skeemi rakendusjuhised

Rakendusjuhised [4] [5] [6] kirjeldavad ära maksete küsimise skeemi sõnumite vormid ja sõnumite töötlemise reeglistiku. Spetsiifiliselt on ära kirjeldatud kuidas reeglite raamatus sisalduvad äriprotsessid, äriprotsessi sammud, andmehulgad ja atribuudid on rakendatud ISO 20022 finantsteenuste standardis kirjeldatud vormi.

Dokumendid sisaldavad makse küsija ja makse küsija teenusepakkuja, teenusepakkujate vahelise, maksja ja maksja teenusepakkuja sõnumite definitsioone. Maksete küsimise

skeemi definitsioonid on saadaval ka XML kujul skeemdefinitsioonidena (XSD). Skeemi definitsioone kasutatakse RTP sõnumite korrektsuse kontrollimiseks ning need põhinevad ISO20022 finantsteenuste sõnumitel.

3.1.3 Maksete küsimise skeemi rakendusliidese spetsifikatsioonid

Rakendusliideste spetsifikatsioonid [11] määravad ära reeglid kuidas tuleb teenusepakkujate vahelised rakendusliidesed käiku panna. Alates 30. novembrist 2023 on teenusepakkujate vahel kohustuslik spetsifikatsioonidele vastavate rakendusliideste kasutamine.

Dokumentatsioon sisaldab *OpenAPI* vormingus teenusepakkujate vahelise rakendusliidese kirjeldust.

Tähtsamad tähelepanekud spetsifikatsioonist:

1. Teenusepakkujate liidese spetsifikatsioonide jaoks on transleeritud maksete küsimise skeemis määratud XML sõnumid API JSON (*application/json*) vormingusse.
2. API sõnumites kasutatakse järgnevat tähemärkide kodeerimise süsteemi:
    a. Ülem-kaamelkiri (Upper **C**amel**C**ase) – RTP komponentide definitsioonid.
    b. Alam-kaamelkiri (Lower **c**amel**C**ase) – JSON omadused (võti-väärtus paar).
    c. Lülikiri (Spinal-Case) – komponentide jaoks nende teekonna ehk adresseerimise jaoks.
3. Toetatud on sünkroonsed ja asünkroonsed tegevusvood kasutades tagasikuulde ressursi aadress ehk *Callback URL-e* (definitsioon mõistete sõnastikus).
4. Enne uue SRTP ressursi postitamist peab klient määrama *Callback URL-i.*
5. Loodud ressursside eristamiseks, näiteks RTP sõnumite identifitseerimiseks kasutatakse unikaalset identifikaatorit (mida loob rakendusliidese server).

6. Sõnumistruktuur toetab HATEOS ehk *Hypermedia as the Engine of Application State* blokki, mille puhul API serveri vastuses sisaldub kindla ressursiga seotud potentsiaalsete tegevuste jaoks eraldi blokk, mis viitab nende tegevuste tegemiseks tarvilikele aadressitele.

7. On kasutusel standardsed veakoodid ja veateate struktuur.

8. Kõikidel POST tegevustel on HTTP päises *Idempotency-key* ehk idempotentsusvõti – viidates ühele unikaalsele ressursile:

    a. API kliendi kohus on enne uue unikaalse saadetise saatmist luua unikaalne ID (UUID RFC4122) ning peab selle saadetisele külge haakima.

    b. API serveri kohus on kindlaks teha, et ei oleks konflikte ressursside vahel.

9. Kõikidel päringutel on eraldi korrelatsioonipäis *X-Request-ID*, mida kasutatakse ühe kindla päringu-vastuse sidumiseks.

Standardse rakendusliidese määratlus makse küsija teenusepakkuja kui kliendi ja maksja teenusepakkuja kui serveri vaatest:

Tabel 1 – Teenusepakkujate vahelise standardse rakendusliidese määratlus

Tegevus	**HTTP meetod + URL**
SRTP saatmine maksja SRTP teenusepakkujale	POST /sepa-request-to-pay-requests/
SRTP **ressursiinfo küsimine** maksja SRTP teenusepakkujalt	GET /sepa-request-to-pay-requests/**{ressursiId}**
SRTP **olekuraporti küsimine** maksja teenusepakkujalt	POST /sepa-request-to-pay-requests/**{ressursiId}**/status-update
SRTP **tühistamispalve saatmine** maksja SRTP teenusepakkujale	POST /sepa-request-to-pay-requests/**{ressursiId}**/cancellation-requests
SRTP **tühistamispalve (RfC) olekuraporti** küsimine maksja teenusepakkujalt	POST /sepa-request-to-pay-requests/**{ressursiId}**/cancellation-requests/**{rfcId}**/status-update

Standardse rakendusliidese HTTP vastuste olekukoodid:

Tabel 2 – Teenusepakkujate vahelise rakendusliidese HTTP vastuste olekukoodid

Tegevus	**HTTP vastusekood**	**Selgitus**
SRTP saatmine maksja SRTP teenusepakkujale	201	RTP ressurss on loodud korrektselt
	400	Ressursi ei saanud luua, halb sõnum
	401	Autentimine on puudulik, pole luba
	406	Server ei leidnud kliendi saadetud *Accept* headeriga sobituvat sisu
	409	Päringut ei saa täita, sest tegemist on juba töödeldava ressursiga
	415	Server ei toeta kliendi saadetud *Content-Type* tüüpi sisu
	422	Server ei suutnud sõnumit töödelda
	429	Klient on saatnud liiga palju päringuid (mahu-piirang)
SRTP **olekuraporti küsimine** maksja teenusepakkujalt ja SRTP **tühistamispalve (RfC) olekuraporti** küsimine maksja teenusepakkujalt	200	Olekuraport on korrektselt vastusena kaasas
	400	Päringut ei saanud täita, halb sõnum
	404	Päritud ressurssi ei leitud
	410	Ressurss ei ole enam saadaval
	Toetab sarnaselt eelnevale 401, 406, 415, 422, 429	
SRTP **tühistamispalve saatmine** maksja SRTP teenusepakkujale	201	SRTP tühistamispalve ressurss on loodud
	Toetab sarnaselt eelnevale 400, 401, 404, 406, 410, 415, 422, 429	

3.1.4 Rakendusliidese turvaraamistik

Turvaraamistik [12] määrab ära rakendusliidese minimaalsed turvaalased nõuded maksete küsimise skeemi jaoks.

Turvaraamistik toob lisaks varasemalt märgitud nelja osapoole mudelisse viienda osapoole. Tegemist on Operational Scheme Manager (OSM) ehk tegevuskava halduri rolliga. Tegevuskava haldur kogub, valideerib, haldab ja teeb kättesaadavaks lisaandmeid

skeemi osaliste kohta. Tegevuskava halduri roll on pakkuda kataloogiteenust skeemis osalejatele.

Turvaraamistik kirjeldab ära kuidas skeemi osalised üksteist tuvastavad, sh. nõuded autentimisel kasutada kvalifitseeritud veebiautentimise sertifikaati (QWAC) ning allkirjastamisel (pitseerimisel) kvalifitseeritud elektroonilise pitseri sertifikaati (QSealC).

Nõuded, mis on määratud turvaraamistikuga rakenduvad ainult skeemiosalejate vahelisele suhtluse haldamisele. Interaktsioonid, mis toimuvad skeemiosaliste ja nende klientide vahel ei ole selle raamistikuga ette kirjutatud.

Turvaraamistik sätestab ka selle, et kõik HTTP päringud ja vastused peavad olema logitud ning kättesaadavad 6 kuud pärast päringu tegemist.

Turvaraamistikust tulenevad nõuded ja rakendusliidese kasutamine täiendavad maksete küsimise skeemi protsessi järgnevalt:

1. Rakendusliidese klient ja server autenditakse kasutades TLS sertifikaate:
    a. Tegemist peab olema QWAC sertifikaadiga.
2. Sertifikaadist tuleneb osalise identifikaator (OSM ID):
    a. Kui ühel teenusepakkujal on mitu erinevat identifikaatorit kasutusel, siis peavad eksisteerima erinevad sertifikaadid iga identifikaatori jaoks.
    b. Identifikaatoreid haldab tegevuskava haldur.
3. Kontrolli API kliendi osalemist skeemis
    a. Võimalik on teha kontroll iga päringu kohta tegevuskava halduri (OSM) poole.
    b. või soovituslikult peaksid kõik rakendusliidese kliendid olema varem eelregistreeritud.
4. Kontrolli kliendi rakendusliidese ressursi ligipääsuõiguseid (autoriseeri).
5. Logi päringud ja vastused.

3.2 Funktsionaalsed ja mittefunktsionaalsed nõuded. Piirangud.

Dokumentatsioonist tulenevad funktsionaalsed, mittefunktsionaalsed nõuded ning piirangud on kirjeldatud järgnevalt. Detailsemalt kirjeldatud nimekirja funktsionaalsetest ja mittefunktsionaalsetest nõutest leiab Lisast 4.

Mittefunktsionaalsed nõuded:

- **Turvalisus –** Tuvastus, autentimine, autoriseerimine, andmete terviklus, konfidentsiaalsus ja käideldavus vastavalt turvaraamistikus sätestatule.
- **Käideldavus –** 24x7 käideldavus koos liiasuse- ja tõrkekindlusmehhanismidega.
- **Võimekus –** Reaalajas sõnumite vahendamise võimekus, võimekus suurema mahu sõnumite puhul tegutseda kui väiksema mahu puhul („skaleeritavus“).
- **Auditeerimine –** Sõnumite detailne logimine ja auditijälgede hoidmine.
- **Paindlikkus -** Võimekus ümber lülitada erinevate rakendusvormide vahel nagu 3 nurga mudel (üks teenusepakkuja) ja 4 nurga mudel (kaks teenusepakkujat).
- **Dokumenteeritus** - Rakendusliidese toimimine peaks olema dokumenteeritud ning avalikult kättesaadav.

Funktsionaalsed nõuded:

- Teenusepakkujal peab olema liides teiste teenusepakkujate poole.
- Teenusepakkujal peab olema liides makse küsija või maksja poole.
- Teenusepakkujal peab olema ühendus tegevuskava halduri (OSM) poole.
- Teenusepakkuja peab toetama põhilisi RTP sõnumivooge (PS-01.01 – PS-01.15, PS-02.01 – PS-02.10R).
- RTP teenusepakkuja peab toetama põhiliste sõnumivoogude edastamiseks vajalikke andmestruktuure (DS-) ja atribuute (AT-).
- Teenusepakkuja peab toetama kasutajate registreerimise, aktiveerimise / desaktiveerimisse protsesse.
- RTP XML sõnumeid peab saama konverteerida rakendusliideses JSON vormi.
- Iga RTP sõnumitüübi jaoks peab olema rakendusliidese puhul ressurss ja lõpp-punkti aadress.

- Teenusepakkuja peab toetama manuste saatmist (näiteks arve).
- Oleku, veateateid, näiteks keeldumissõnumeid tuleb hallata vastavalt standardile.
- RTP sõnumid peavad olema edastatud mööda turvalisi kanaleid ehk toetus krüptograafialastele meetoditele.
- RTP ressursid peavad olema kättesaadavad vaid autoriseeritud osapooltele.
- Teenusepakkuja funktsionaalsus peab võimaldama RTP skeemi toimimist vastavalt Euroopa Maksenõukogu RTP reeglite raamatus, rakendusjuhistes, turvaraamistikus või teistes väljastatud lisadokumentides määratletud tingimustele.

3.3 Programmeerimismeetodite analüüs

Rakendusliideste arendamine on mõistlik kui rakendusliideseid on võimalik kiirelt, lihtsa vaevaga luua. Liideste muutmine peaks samuti olema lihtne ettevõtmine. Liideste loomiseks vajalik raamistik peaks hõlpsalt toetama lisafunktsioonide rakendamist, see tähendab raamistikul oleks rikkalik teekide ja tööriistade valik.

Funktsionaalsetest ja mittefunktsionaalsetest nõuetest tulenevalt on meie rakenduse puhul tähtis kõrge turvalisus, käideldavus, samuti võimekus efektiivselt suure hulga sõnumite mahuga hakkama saada ning vajadusel võimekust suurendada. Seetõttu rakenduse arendamiseks on eelistatud programmeerimiskeel (raamistik), mis kuuluks ökosüsteemi, see tähendab lihtsalt oleks võimalik tarkvarasüsteemi rakendada, täiendada lisateenustega. Soovime arendada raamistikul millel on näha tulevikus tugevat arendajapoolset toetust.

Ära ei saa unustada ka eelarvet. Eelarve meil on ajaline, aga aeg on raha. Seetõttu on ka arendaja aeg arvel. Arendada on võimalik raamistikega millega praegused ja tulevased projektiga töötavad arendajad on sinapeal ning seesuguseid arendajaid on ka tulevikus leida.

Dokumenteerimine peab olema lihtne, eelistatult peaks saama lihtsasti rakendusliidese koodist toota mõne spetsifikatsiooni (nt. OpenAPI) laadis dokumentatsiooni.

3.3.1 Programmeerimismeetodite ja raamistike valik

IEEE Spectrumi 2023. aasta ülevaade erinevate programmeerimiskeelte ja raamistike populaarsusest annab alust valikuks [13]. Nimelt vaadeldakse seal kolme erinevat mõõdikut: kui populaarne on raamistik inseneride seas (*Spectrum index)*, kui suur on nõudlus selliste raamistike oskajate jaoks tööandjate juures ning hinnatakse ka tulevikuvaadet (trendi).

Ülevaate järgi joonistuvad välja viis populaarset – Python, Java, C#, Javascript (Node.js), Go. Vaatleme neid meie vajaduste tarbeks. Vaatluse tarbeks valime igast keelest ühe populaarseima raamistiku, mis võimaldab rakendusliidest luua. Tähtis on mainida, et tegemist on hinnangutega, mis põhinevad autori ning ülemaailmse veebi allikate sümbioosist ning kohanduvad just seesugusele projekti vajadustele.

Teenusepakkuja tarkvara põhiline funktsionaalsus on justnimelt turvalise rakendusliidese pakkumine. Autori analüüsist tuleneb, et kõikidel neil programmeerimiskeeltel on raamistikud, mis sobivad nõutud rakenduse arendamiseks. Pea kõikidel neil keeltel (v.a. Python) on toetus tugevalt määratletud tüüpidele (*strongly* typed), nende raamistikud võimaldavad lihtsalt rakendusliideste arendust ning dokumenteerimist. Seetõttu võib öelda, et kõik need raamistikud võivad meie funktsionaalseid ja mittefunktsionaalseid nõudeid täita. Autori kaalutlusel langeb valik nende keelte ja raamistike vahel arendaja enda varasemate teadmiste (õppimiskiirus), jõudluse, turvalisusraamistiku hindamisele ning võimalusega siduda loodud rakendus ökosüsteemi, mis võimaldaks tulevikus lihtsasti rakenduse funktsionaalsust muuta ja laiendada.

Jõudluse võrdlemisel on kasutatud *The Benchmaker Web Frameworks Benchmark* tulemusi [14] kus on rakendusliideste puhul hinnatud päringuid sekundis (TPS), 90-protsentiili viite tulemused, keskmise viite ning maksimaalse viite tulemused. Vastavalt kohtadele nendes kategooriates on antud hinnang raamistiku jõudlusele.

Kõik analüüsitud raamistikud toetavad turvaelementide integratsiooni. Spring Security poolt pakutavad parimad praktikad ehk mustrid võimaldavad lihtsalt arendada turvalist rakendust [15]. Sarnane lugu on ASP.NET turvaraamistikuga, mis toetab autentimise, autoriseerimise vahevarasid, pakub raamistiku poolt kaitset erinevate rünnakvektorite vastu [16].

Integratsiooni võimalus ökosüsteemi vaatleb pilveteenuspakkujate lisateenuste integratsiooni võimalust erinevatel raamistikel. Kuigi kõiki raamistikke on võimalik läbi virtualiseerimise ja konteinerteenuste orkestreerimine erinevatel taristutel käitada ning laiendada, siis ASP.NET Core ühilduvus Microsoft Azure platvormiga ning tugev sellealane dokumentatsioon annab sellele raamistikule lisapunkte. Spring raamistik omakorda sisaldab laialdaselt võimalusi erinevate mustrite ja tööriistade abil enda rakendust laiendada, näiteks Spring Cloud toetab arendajaid hajussüsteemide ehitamisel. Pythonil ja Node.js on tugev teekide baas ning läbi teekide on võimalus samuti sarnast funktsionaalsust saavutada.

Kõik analüüsitud raamistikud on sobivad rakendusliideste arendamiseks. Võrdluse käigus sai loodud Tabel 3, mis kirjeldab ära erinevate raamistike sobilikkuse hindamist autori ning spetsiifilise projekti vaatest eelnevalt kirjeldatud meetoditel.

Tabel 3 – Programmeerimiskeelte ja raamistike võrdluse tabel

Programmeerimiskeel / raamistik	**Varasem kogemus / õppimiskiirus**	**Jõudlus**	**Turvalisus**	**Ökosüsteem**	**Summa**
C# (ASP.NET Core)	8	9	9	9	35
Java (Spring Boot)	6	5	9	8	28
Python (Django)	5	2	8	6	21
Go (Gin)	2	9	7	5	23
Node.js (Express.js)	4	5	7	6	22

Tulemustest võib järeldada, et seoses varasema kogemuse olemasolu ning tugeva jõudlusega pakub ASP.NET Core sobivat raamistikku tarkvara arendamiseks. Justnimelt ASP.NET Core (C#) raamistikul arendame maksete küsimise teenusepakkuja prototüüpi.

3.4 Rakendusliideste parimate praktikate analüüs

Rakendusliidesed on ühenduskohad kahe erineva rakenduse vahel, et suhelda ning vahetada andmeid. Meie kirjeldame veebiteenuse põhiseid rakendusliideseid ehk rakendusliideseid, mis on ehitatud HTTP protokolli abil või peale. Seoses rakendusliideste spetsifikatsioonidega, mis said kirjeldatud varasemas dokumentatsiooni

analüüsi osas põhineb meie rakenduse veebiteenuse tehnoloogia REST arhitektuurilisel raamistikul.

Veebiteenuste rakendusliideseid (*Web API*), mis vastavad REST arhitektuurilistele piirangutele kutsutakse *RESTful* rakendusliidesteks. Kuna RESTi põhised veebiteenused pole standardiseeritud, siis leidub erisuguseid veebiteenuseid, mõned täidavad rohkem või vähem REST-i arhitektuurilisi piiranguid – on vähem või rohkem *RESTful*. Selleks, et hinnata rakendusliideste vastavust REST-i arhitektuurilistele piirangutele on kasutusel mitmeid mudeleid. 2008. aastal pakkus Leonard Richardson välja *RESTful* rakendusliideste küpsuse hindamise mudeli [17]:

1. Tase 0 – kasutatakse HTTP protokolli kui transpordiprotokolli andmetele, teenustele.

2. Tase 1 – luua individuaalsete ressursside jaoks eraldi ressursilokaatorid (URI).

3. Tase 2 – kasutada HTTP meetodeid (POST, GET, …), et eristada ressursi peal tehtavaid toiminguid.

4. **Tase 3 – kasutada HATEOAS ehk Hypermedia as the Engine of Application State funktsionaalsust.**

HATEOAS on rakendusliidese oluline osa, mis võimaldab kliendil suhelda ressurssidega dünaamiliselt. See annab kliendile informatsiooni ressursist, selle olekust, võimalikest ressursi ning ressursiga seotud operatsioonidest. Informatsioon on *hüperlinkide* kujul, mille sihtkohad võimaldavad ressursi peal teha erinevaid tegevusi. Kliendid ei pea teadma eelnevalt kõiki ressurssidega seonduvaid teid (URL-e), vaid saavad dünaamiliselt navigeerida kasutades varasemalt ressursiga seonduva päringu vastuses olevat informatsiooni. **Analüüsist johtus, et maksete küsimise skeemi teenusepakkujate vahelised liidesed peavad olema Richardsoni III küpsustasemel.** Rakendame ka ülejäänud liidesed kolmanda taseme küpsusastmel.

3.4.1 Rakendusliidese rakendamise parimate praktikate valikud

Peale Richardsoni kolme tasemega määratud praktikate on ka veel mitmeid teisi praktikaid, mida soovitatakse ja on kombekas kasutada. Järgnevalt kirjeldatu on rakendatud teenuspakkuja tarkvaras. Proovin valikud siin paremini lahti kirjeldada.

Standardile põhinevad olekukoodid

Päringutele vastamisel tuleks kasutada standardiseeritud HTTP olekukoode, et oleks võimalik hõlpsasti eristada toimingust tulenev olek. Olekukoodid jagunevad: 100 - 199 informatsioonilised, 200 - 299 õnnestumise teavitused, 300 - 399 ümbersuunamised, 400-499 ja 500 - 599 probleemid [18].

Probleemidega tegelemise ehk veahalduse puhul eristatakse vigasid (*error*) ja defekte (*fault*). Vigadeks loetakse tavaliselt kliendipoolseid vigu ehk rakendusliidese valesti kasutamisest tekkivaid probleeme. Nendeks võib lugeda näiteks valede parameetrite kasutamist, valel kujul andmete esitamist, autoriseerimise puudumist jt. Selliste vigade puhul tagastatakse olekukoodid vahemikust 400 – 499. Defektide puhul leidub viga kuskil süsteemis ehk viga on serveripoolne. Defektide puhul tagastatakse olekukoodid vahemikust 500 – 599.

Veahaldus

Vigade puhul on tarvis klienti teavitada sellest, miks viga juhtus. Selleks tarbeks loodud sõnumid peaksid olema standardiseeritud üle kogu rakendusliidese ning sisaldama vähemalt: veakoodi, vea kirjeldust (sõnumit), vea juhtumise aega, vea subjekti. Võimalusel ka veaga seonduvaid detaile või viidet kohale kus neid detaile võib leida. Veakirjeldused peaksid olema lihtsasti ja üheselt arusaadavad.

Versioonihaldus

Rakendusliidese arendamisel peaks järgima „vähima üllatuse printsiipi“. Vähima üllatuse printsiip tähendab, et liidese kasutajatele ei tohiks rakendusliidese arendamisel tehtud muudatused tekitada üllatusi. Kõik suured teenust muutvad muudatused (*breaking change*), mis muudavad rakendusliidese „lepingut“ näiteks muutused veateadetega seoses, muutused rakendusliidese käitumises, rakendusliidese parameetrite muutmine või nende eemaldamine on üllatust tekitavad ja võivad katki teha klientrakenduse töö [19].

Kui arendamisel tehakse suuri teenust muutvaid muudatusi, siis peab olema kasutajatel olema kindel tee ja teadmus muudatustega kohanemiseks. Seetõttu on soovitatav:

- toetada (kasutada) samaaegselt erinevaid rakendusliidese versioone.

- rakendusliideste dokumentatsioon peab sisaldama erinevate versioonide dokumentatsiooni (ning kirjeldust erinevusest eelmise versiooniga).

Turvalisus

Turvalisuse puhul on tähtis, et mitte leiutada ratast uuesti, kasutada teada tuntud raamistikudes nagu OWASP soovitatut [20]. Mõned spetsiifilisemad nõuded:

- suhtlus kliendi-serveri vahel peab olema krüpteeritud (TLS).
- kõik URL-id millele rakendusliideses viidatakse kasutavad HTTPS-i.
- autentimine ja autoriseerimine peab olema lubatud nii lõpp-punktide põhiselt, meetodite põhiselt kui ressursside põhiselt.
- teenused, mis kasutavad *Callback URL-e* peavad aktsepteerima ainult HTTPS aadresse.
- *Callbacke* tohib sooritada vaid usaldatud aadressitele, peab eksisteerima usaldatud aadresside kontroll.
- Valideeri alati sisu ja sisendeid - parameetreid, sisu, sisu tüüpi kasutades turvalist sõnumite töötlejat (*parserit*).

Kliendihõlpsus

- rakendusliideses kasutatavad nimed (identifikaatorid, URL-id, parameetrid jt) peavad olema ühtselt reeglite päraselt määratletud (standardiseeritud), lihtsasti loetavad ja arusaadavad rakendusliidese kasutajale.
- ressursilokaatorid (URL) peavad olema lihtsasti loetavad ja konstrueeritavad.

4 Tarkvara realiseerimise kontseptsioon

Nõuded tarkvarasüsteemile tulenevad Euroopa Maksenõukogu poolt määratud RTP teenusepakkuja spetsifikatsioonidest. Dokumentatsiooni analüüsi käigus sai kirjeldatud skeemi töötamise alused, osapooled, leitud funktsionaalsed ja mittefunktsionaalsed nõuded, kirjeldatud ära teenusepakkuja põhiprotsess ja turvanõuded. Vastavalt nendele paneme paika tarkvara põhialused.

Tarkvara lihtsamaks kirjeldamiseks kasutame C4 mudelit, mis on lihtne, arendajasõbralik tarkvara arhitektuuri kirjeldamise ja kujutamise viis. C4 mudel võimaldab kirjeldada hierarhiliselt abstraktsioone (tarkvarasüsteemid, konteinerid, komponendid ja kood) ning esitada neid lihtsasti arusaadavate diagrammidena [21].

C4 mudel viitab neljale C-le:

1. Süsteemi kontekst (*Context*) – kirjeldab suurt pilti seostades kindla tarkvarasüsteemi tema kasutajate ning teiste süsteemidega millega suhestub.

2. Konteiner (*Container*) – vaade kindla tarkvarasüsteemi sisse. Konteiner on ühe kindla tarkvarasüsteemi osade kirjeldus, näiteks veebirakendus, andmebaas, mobiilirakendus. Konteiner on eraldiseisev üksus, mis käitab koodi või salvestab andmeid.

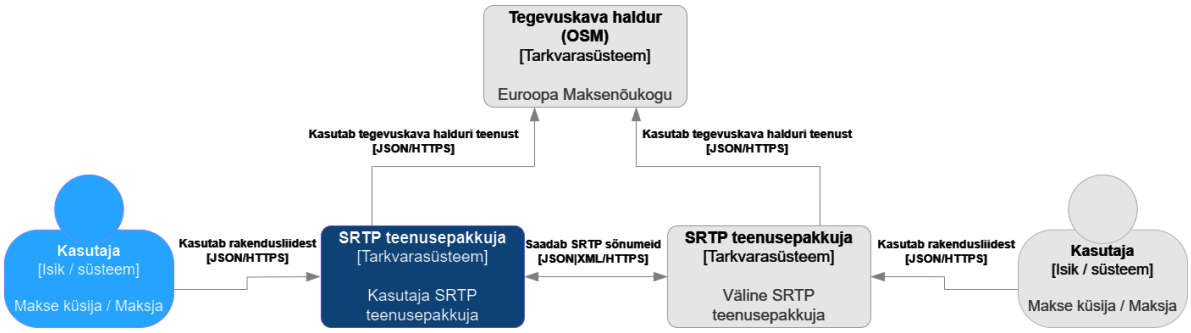
3. Komponent (*Component*) – vaade kindla konteineri sisse ehk kirjeldab eraldiseisva tarkvaraüksuse komponendid, näiteks rakendusliideste kontrollerid, turvakomponendid, teenused jt.

4. Kood (*Code*) – kirjeldab iga komponendi sisse kasutades erinevaid UML ja olemi-diagramme, kirjeldades näiteks klasse või liideseid. Tavaliselt luuakse koodibaasist automaatselt.

4.1 Süsteemi konteksti vaade

Süsteemi konteksti vaatest joonistub välja tuttav nelja nurga (Joonis 1 – 4 nurga mudel („4 corner model“) süsteem, aga ühe lisandusega. Nimelt lisaks makse küsijale, makse küsija teenusepakkujale, maksjale, maksja teenusepakkujale lisandus turvaraamistikust

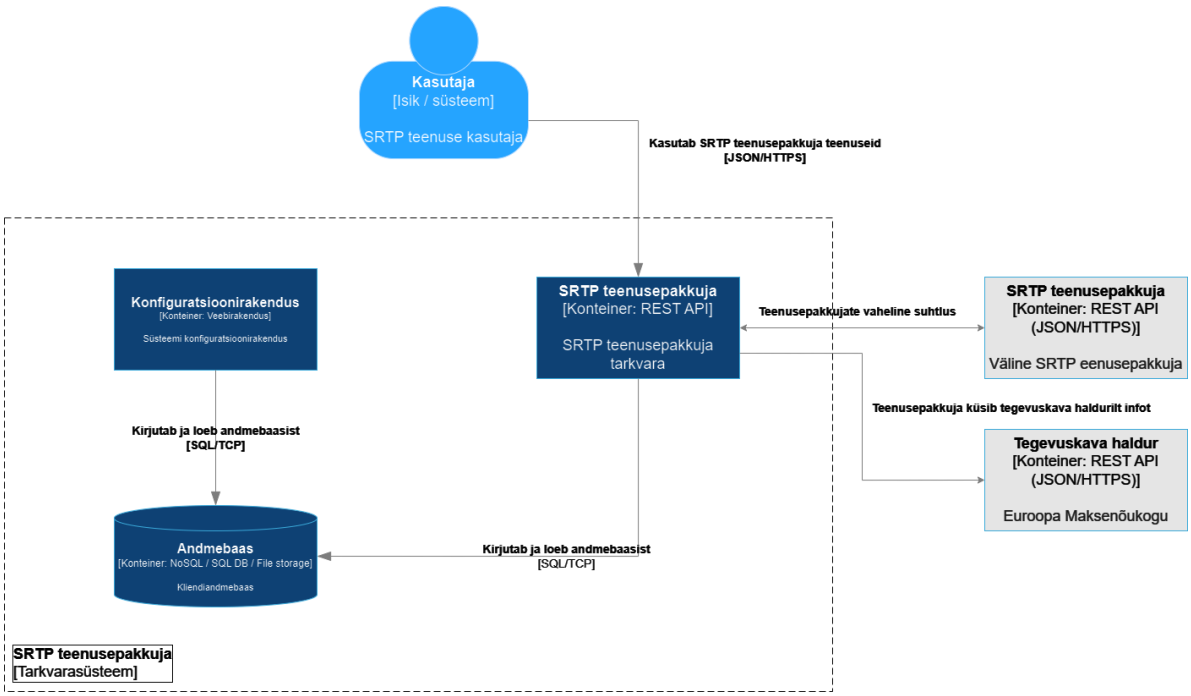
tegevuskava halduri roll. Kaks SRTP teenusepakkuja tarkvarasüsteemi on pildil identsed, sest teenusepakkuja tarkvara peab võimaldama nii makse küsija kui maksja teenindamist. Kui makse küsija ja maksja kasutavad sama teenusepakkujat, siis võib lihtsustatult vaadelda vaid ühte kasutaja, SRTP teenusepakkuja ja tegevuskava halduri kolmikut („3-corners“ mudel).



Joonis 6 – Süsteemi kontekstvaade

4.2 Konteineri vaade

Vaatame maksete küsimise skeemi teenusepakkuja tarkvarasüsteemi sisse. Abstraktselt vaadatuna koosneb süsteem kolmest põhikomponendist: SRTP teenusepakkuja komponent, konfiguratsioonirakendus ning andmebaasi komponent.



Joonis 7 – Süsteemi konteinervaade

Andmebaasi komponendi disain pole selle lõputöö raames, aga võib mainida, et kuna funktsionaalsed nõuded sätestavad sertifikaatide säilitamise, siis on tarvis mingit andmekandjat, kus sertifikaate hoiustada ning samuti logimise nõuded on väga karmid (kõik HTTP päringud ja vastused tuleb logida ning säilitada kuueks kuuks), siis tulevikus suure mahuga liiklusega SRTP teenusepakkuja jaoks võib-olla vajalik näiteks lisa vahekihtide (*cache*) kasutamine või/ja dokumendipõhise andmebaasi kasutamine. Siit tuleneb nõue, et andmebaasi peaks saama hõlpsalt tulevikus muuta.

Konfiguratsioonirakendus on lihtne veebirakendus, mille abil on võimalik SRTP teenusepakkuja tarkvara konfigureerida (näiteks lubatud partnerteenusepakkujate seadistusi ja nende sertifikaate hallata, muuta tarkvara spetsiifilist käitumist jt.). Tegemist on selle lõputöö raames väga lihtsa rakendusega, et mille abil spetsiifilisi andmebaasiväärtuseid on võimalik muuta ning täpsemalt seda rakendust lõputöös ei kirjeldata.

4.3 Teenusepakkuja konteineri komponentvaade

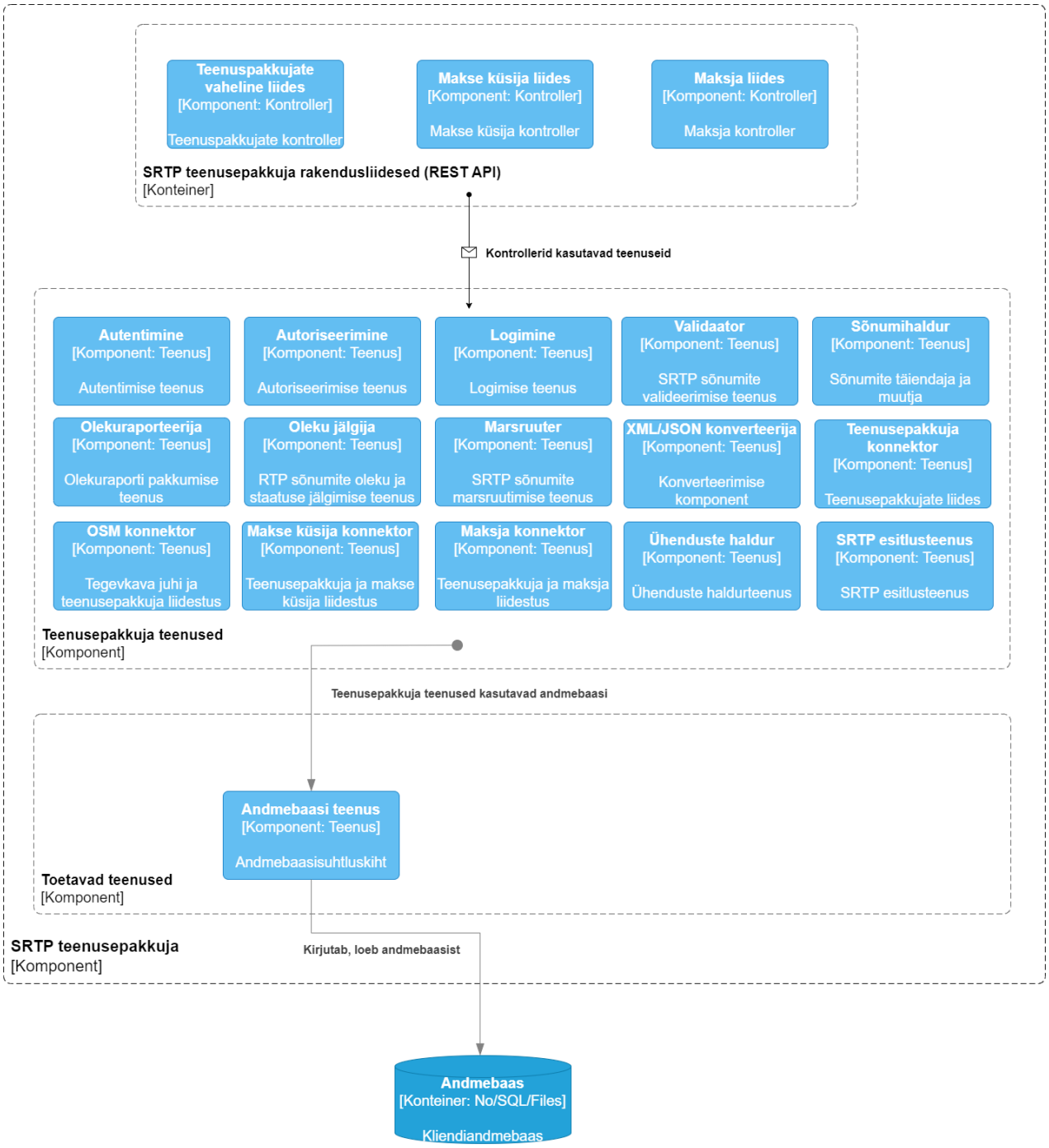
SRTP teenusepakkuja komponentide kirjeldus on meie dokumentatsiooni ning parimate praktikate analüüsi tulem - funktsionaalsete ja mittefunktsionaalsete nõuete, protsessi sammude ning osapoolte süntees.

SRTP teenusepakkuja rollid vahelduvad. Vahel võib SRTP teenusepakkuja olla klient (sõber-teenusepakkujatega suhtlemisel või C*allback*-funktsionaalsust kasutades) teisalt server (rakendusliideste teenindamisel). Teenusepakkuja peab suutma tagada nii sünkroonsed kui asünkroonsed vood.

SRTP teenusepakkujal on kolm tähtsat rakendusliidest – makse küsija liides, teenusepakkujate vaheline liides ning maksja liides. Need liidesed rakendatakse REST kontrolleritena (standardi põhiselt).

Kontrollerid suhtlevad teenusepakkuja alusteenustega. Alusteenused on protsess sammude järgi jaotatud alamteenusteks. Arhitektuur ei kirjuta ette kindlat rakendusviisi (nt. mikroteenuste kasutamist), kuid eesmärk on tulevikus seda võimaldada, kui see peaks vajalikuks osutuma. Seetõttu on ka alamteenused piisava suurusega väikesed tükid. Alamteenused omakorda suhtlevad omavahel ning kasutavad andmejuurdepääsukihti.

Joonisel nr. 8 on kujutatud teenusepakkuja tarkvarasüsteemi konteineri komponendid



Joonis 8 – Süsteemi komponentvaade

4.3.1 Andmebaasi kasutamine ning programmi struktuur

Süsteem peaks toimima eraldiseisvalt andmebaasi spetsifikatsioonist. Andmebaasi disain võib tulevikus muutuda, tähtis on eristada selle töötamine teiste teenuste toimimisest, seetõttu andmebaasiga suhtlus käib samuti eraldi teenuse kaudu. Hiljem peab olema lihtne andmeallikad (andmebaasid) välja vahetada.

Andmemudel, operatsioonid andmetel ning andmed (ressursid), mida soovime rakendusliideste kaudu esitame ei ole üks-ühele (representatsioonid), seega on tarvis eraldi kaardistamise- ja konsolideerimismehhanismi (kihti) andmehoidlast esitluskihini.

Et eelpool kirjeldatud nõudeid täita on soovitav kasutada rakendamisel kihistatud lähenemist – eristada andmetele juurdepääsu kiht (DAL), äriloogika kiht (BLL) ja kasutajaliidese kiht (UI / API).

4.3.2 SRTP teenusepakkuja teenuste kirjeldus

Teenusepakkuja tarkvara teenuste (komponentide) kirjeldus on toodud Tabelis nr. 3:

Tabel 4 – SRTP teenusepakkuja teenuste kirjeldused

Teenus	**Funktsioon**	**Seos protsess sammudega**
Autentimine	Klientide autentimine kasutades TLS ja sertifikaate, kontrollid tegevushalduri poole.	Tuleneb turvaraamistikust, eelneb üldisele protsessile nii makse küsija, teenusepakkuja kui maksja rakendusliidese suhtluse puhul.
Autoriseerimine	Kliendile vastavate õiguste lubamine, kontrollid tegevushalduri poole.	Tuleneb turvaraamistikust, eelneb üldisele protsessile nii makse küsija, teenusepakkuja kui maksja rakendusliidese suhtluse puhul.
Logimine	Süsteemis toimiva kirja panek.	Kõikide protsess sammudega
Validaator	Valideerida sõnumite vastavust nõuete ja XML skeemi definitsioonidega.	PS-01.02/02R, PS-01.05/02CR, PS-02.02, PS-02.05/05CR, PS-02.10
Sõnumihaldur	Lisada, kustutada ja muuta RTP sõnumite sisu vastavalt nõuetele	PS-01.03, PS-01.12, PS-02.03, PS-02.08
Olekureporter	Koostada olekuraporteid.	PS-01.05, PS-01.08, PS-01.12, PS-01.14
Olekujälgija	Jälgida RTP sõnumite olekut, näiteks aegumistähtaega ning koordineerida asünkroonseid tegevusi	PS-01.02-PS-01.10, PS-01.12 – PS-01.15, PS-02.02-PS-02.06, PS-02.08-PS-02.10/10R
Marsruuter	Leida suund ning suunata sõnumeid õigete adressaatideni	PS-01.04, PS-01.05R, PS-01.06, PS-01.09, PS-01.10, PS-01.13, PS-0.14/14R, PS-01.15, PS-02.04, PS-02.06, PS-02.09, PS-02.10*

Tabel 4 – SRTP teenusepakkuja teenuste kirjeldused

Teenus	**Funktsioon**	**Seos protsess sammudega**
XML / JSON konverter	XML ja JSON sõnumite vaheline konversioon, objektide *serialiseerimise ja deserialiseerimise* teenus	PS-01.02, PS-01.08, PS-01.12, PS-01.15, PS-02.02, PS-02.06, PS-02.08.
Teenusepakkuja konnektor	Hoolitseb, et teenusepakkujaga ühenduda. Teenusepakkuja kui kliendi roll.	Marsruuter kasutab konnektori teenuseid: PS-01.04, PS-01.05R, PS-01.09, PS-01.13, PS-01.14/14R, PS-02.04, PS-02.05CR, PS-02.08.
OSM konnektor	Hoolitseb, et tegevuskava halduriga on ühenduda. Teenusepakkuja kui kliendi roll	Autentimise ja autoriseerimise korral.
Makse küsija konnektor	Hoolitseb, et makse küsijate teenustega ühenduda. Teenusepakkuja kui kliendi roll	Marsruuter kasutab konnektori teenuseid: PS-01.02/02R, PS-01.10, PS-01.12, PS-01.14/14R
Maksja konnektor	Hoolitseb, et maksjate teenustega ühenduda. Teenusepakkuja kui kliendi roll.	PS-01.06, PS-01.07/07R, PS-01.14, PS-02.07.
Ühenduste haldur	Hoolitseb, et osapooltega on võimalik ühendusi luua (seob ühenduse loomiseks tarvilikud detailid nagu sertifikaadid ja konnektorid).	Kõikide protsess sammudega, mille puhul kasutatakse konnektoreid.
SRTP esitlusteenus	Hoolitseb, et maksjale vajalikke RTP esitlemise sõnumeid saaks vastu võtta, edastada, hallata.	PS-01.06, PS-01.15, PS-02.07.

4.4 Rakendusliideste disain

Meil on kolm tähtsat rakendusliidest – makse küsija – teenusepakkuja, teenusepakkuja – teenusepakkuja ning teenusepakkuja – maksja liides.

Dokumentatsiooni analüüsi käigus selgus, et SRTP teenusepakkujate vahelise rakendusliideste määratluses on kasutusel nii individuaalsed ressursilokaatorid, HTTP meetodipõhised ressursioperatsioonid kui ka toetus HATEOAS funktsionaalsusele – tegemist on Richardsoni III taseme rakendusliidesega. Seetõttu eksisteerib ka eeldus, et

ülejäänud rakendusliidesed võiksid vastata selle taseme küpsusastmele. Meie ülejäänud rakendusliidesed tuleb rakendada justnimelt sedamoodi. Tabelis 4 on toodud välja makse küsija ning maksja teenusepakkuja liideste definitsioon (ressurssi URI ja meetodiga seotud tegevuse kirjeldus).

Tabel 5 – Makse küsija ning maksja teenusepakkuja ressursid ja meetodid määratlus

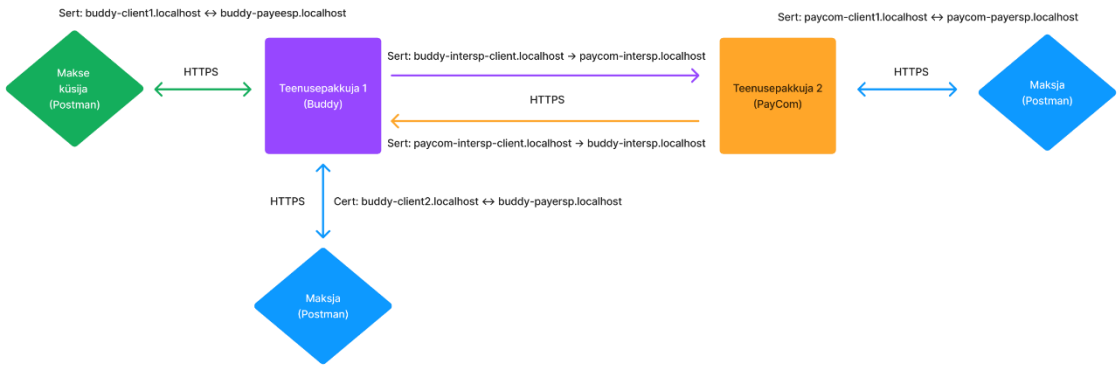
Ressurss (URI) / Meetod	**POST**	**GET**
Makse küsija liides		
payee/srtp/	Makse küsija loob uue SRTP (PS-01.01)	Nimekiri kõikidest makse küsija poolt avatud SRTP ressurssidest
payee/srtp/**{srtp-id}**	-	Makse küsija küsib kindla SRTP ressursi informatsiooni
payee/srtp/**{srtp-id}**/status-update	Makse küsija küsib uut olekuraportit (PS-01.12)	Makse küsija küsib olemasolevat olekuraportit (viimane teadaolev olukord enda teenusepakkuja juures)
payee/srtp/**{srtp-id}**/cancellation-request	Makse küsija loob uue RfC ehk SRTP tühistamispalve	-
payee/srtp/**{srtp-id}**/cancellation-request/**{rfc-id}**/status-update	Makse küsija küsib uut RfC olekuraportit ehk SRTP tühistamispalve olekukuraportit	Makse küsija küsib olemasolevat olekuraportit (viimane olukord enda teenusepakkuja juures)
payee/srtp/enrollment	Kasutaja registreerimine	
payee/srtp/activation	Makse küsija aktiveerimise protsess	Nimekiri kõikidest aktiveeritud partneritest
Maksja liides		
payer/srtp/	-	Nimekiri kõikidest avatud SRTP-dest
payer/srtp/**{srtp-id}**	-	Kindla SRTP informatsioon
payer/srtp/**{srtp-id}**/assess	Vastus RTP päringule (aktsepteeri / keeldu)	-
payer/srtp/enrollment	Kasutaja registreerimine	-
payer/srtp/activation	Maksja aktiveerimise	Nimekiri aktiveeritud partneritest
Rakendub mõlemale liidesele	**DELETE**	**GET**
…/srtp/activation/**{srtp-id}**	Maksja desaktiveerimine	Kindla aktiveeringu detailid

5 Rakendus ja tulemused

Lõputöö raames luuakse maksete küsimise skeemi teenusepakkuja tarkvara prototüüp. Prototüüp on arendatud kui monoliitne rakendus kuid struktureeritud nii, et tulevikus oleks võimalik erinevaid mooduleid (teenuseid) jagada alamprogrammideks (näiteks mikroteenusteks). Samuti on tehtud prototüübi arendamise käigus mõned teadlikud otsused, mis lihtsustavad prototüübi ehitamist kuid produktsiooni viies peaks arvatavasti muutma. Näiteks prototüüp kasutab sertifikaadipõhiseks autentimiseks ning autoriseerimiseks *Kestreli* veebiserveri võimalusi kuid produktsioonis oleks mõistlik need tegevused delegeerida kas rakendusliidese lüüsile (või mõnele *proksile*), mis võimaldaks ka koormuse jagamist, seeläbi võimaldaks paremat käideldavust, mitme rakendusliidese instantsi samaaegset kasutamist jt. Niisamuti on ka sertifikaadid ja paroolid hoiustatud *Kestreli* veebiserveri juures kuid tulevikus võiks kasutada mõne spetsiifilise salajaste võtmete salvestamiseks mõeldud turvateenuse nagu *Azure Key Vault* teenuseid. [22] Sellised lahendused ei ole lõputöö prototüübi arendamise skoobis ning toote produktsiooni viimine on eraldiseisev projekt.

Prototüübi rakendus kasutab kahte teenusepakkuja instantsi, mis kasutavad identset andmebaasi (andmestruktuuri) ning erinevad vaid konfiguratsiooni ning andmebaasis sisalduvate andmete kaudu. Kahe teenusepakkuja tarkvara instantsi kasutamine võimaldab meil kontrollida lahenduse toimimist erinevatel stsenaariumitel, kui makse küsija ja maksja on ühe teenusepakkuja kasutajad või kui makse küsija ja maksja on erinevate teenusepakkujate kasutajad.

Joonisel 9 on kirjeldatud prototüüpsüsteemi rakendus.



Joonis 9 – Rakendatud prototüüpsüsteemi ülevaade

Rakenduse jaoks on konfigureeritud kaks erinevat teenusepakkujat *Buddy* ja *PayCom*. Teenusepakkuja konfigureerimine toimub läbi veebiteenuse instantsi *AppSettings.{teenusepakkuja}.json* konfiguratsiooni. Konfiguratsioon võimaldab määrata instantsile iseloomulikud identifikaatorid, näiteks kliendi ning tegevuskava juhi (OSM) poolt määratud identifikaatorid. Neid identifikaatoreid kasutatakse, et rakenduses eristada ennast kui teenusepakkujat teistest teenusepakkujatest. Lisaks võimaldab konfiguratsioon määrata ära erinevate rakendusliideste jaoks kasutatavad (serveripoolsed) sertifikaadid. Tarkvara konfiguratsioon on määratletud järgnevalt (Joonis 10):

```
"ServiceProviderSettings" : {
    "ClientId": "Buddy",
    "OsmId": "OSM-BUDDY1",
    "BicFi": "BUDDYC123",
    "DefaultPayeeCertName": "buddy-payeesp.localhost",
    "DefaultPayerCertName": "buddy-payersp.localhost",
    "DefaultInterSpCertName": "buddy-intersp.localhost",
    "DefaultCertName" : "buddy-intersp.localhost"
  }
```

Joonis 10 – Teenusepakkuja tarkvara instantsi konfiguratsioon

5.1 Kasutatud tehnoloogiad ja tööriistad

Programmeerimisraamistik ja programmi kirjutamise keel: .NET Core, ASP.NET Core. [23] Keel on C#.

Veebiserver: Kestrel [24].

DI ehk sõltuvuste pööratud kasutus: ASP.NET Core raamistikku sisseehitatud DI mootor [25].

Objektrelatsiooni kaardistamine (ORM): Entity Framework Core (EF Core) [26].

Andmeedastusobjektide (DTO) transleerimise-kaardistamine: – AutoMapper [27] ning HATEOAS linkide loomise jaoks RiskFirst.Hateoas [28].

Logimine: Serilog [29], Serilog.Sink.ElasticSearch [30] (adapter) ja ELK (Elasticsearch, Logstash, Kibana) logimismootor (andmebaas, andmehaldur, visualeerimistöörist) [31].

API dokumenteerimise raamistik – Swagger UI (OpenAPI spetsifikatsioon) koos Swashbuckle dokumenteerimise tööriistadega [32].

Andmevormide konverteerimine (*serialiseerimine*): Microsofti .Net raamistiku System.Text.Json ja System.Xml.

Veahaldus: Vigade avastamiseks Microsofti sisseehitatud Model Validator [33] ning XML skeemi valideerimise süsteem. Vigade standardiseeritud kujul raporteerimiseks jaoks loodud eraldi vahevara.

POCO ehk C# objektide loomine: Euroopa Maksenõukogu XML skeemidest (XSD) POCO ehk C# objektide loomiseks XMLSchemaClassGenerator (xscgen) tööriist [34].

Testid: XUnit raamistik [35]. Moq [36] objektide jäljendamiseks. Github Copilot Chat ühiktestide kirjutamisel aitamiseks [37].

Arenduskeskkond: JetBrains Rider (koodi kirjutamine), Docker (andmebaas, ELK logimismootor jooksutamiseks), Visual Studio Code (XML skeemide analüüs)**.**

Kõik, mis pole siin nimekirjas mainitud (näiteks erinevate konfiguratsioonide haldus) on .NET Core raamistikku sisse ehitatud.

5.2 Rakenduse koodi kirjutamise põhimõtted

Rakenduse funktsionaalsete ning mittefunktsionaalsetest nõuetest tuleneva arhitektuuri ning disaini rakendamine koodis on võimalik mitmeti. Koodi kirjutamisel on eesmärk koodibaasi hea hooldusvõimekus ehk tuleviku muutuste tegemisel tekkivate defektide potentsiaali vähendamine, hõlpsus hiljem laiendada rakendust uue funktsionaalsusega, koodi taaskasutus ning paindlikus, testitavus, üldiselt lihtsasti organiseeritud ning arusaadav kood.

Nende eesmärkide saavutamiseks on mitmeid võimalusi. Laialt levinud ning teada tuntud lähenemine objektorienteeritud koodi organiseerimiseks eelkirjeldatud eesmärkide saavutamiseks põhineb SOLID printsiipidel. SOLID printsiibid on järgnevad:

- **Ühe vastutuse printsiip** (***S****ingle Responsibility*) – klassil peaks olema vaid üks kindel põhiline vastutus. Ei tohiks eksisteerida *super*-klasse, mis täidavad mitut eraldiseisvat ülesannet, näiteks töötlevad kui ka salvestavad andmeid ning mille muutmine on seetõttu ka suurema tõenäosusega vajalik. Muutmine võib tuua kaasa ootamatuid tulemusi ehk defekte tarkvara töös.

- **Avatud-suletud printsiip** (***O****pen-Closed*) – klass (komponent) peaks olema avatud laiendamiseks kuid suletud muutmiseks. Avatuse eesmärk on taaskasutus läbi sellelt pärimise (laiendamise), suletuse eesmärk on vähendada vanemklassi muutmisega tekkivaid potentsiaalseid defekte, mis omakorda laienevad tulenenud klasside tööle.

- **Liskovi asendusprintsiip** (***L****iskov substitution*) – vanemklass, mida on laiendatud peaks olema asendatav tulenenud klassidega koodis nii, et programmi funktsioneerimine säiliks, programmi töö ei rikneks. Võimaldab suurendada koodi taaskasutust ehk ei pea iga tulenenud klassi jaoks uut funktsionaalsust kirjutama.

- **Liideste segmenteerimine (*I****nterface segregation***) **–** piisavalt väikesed liidesed ehk lepingud, mida vastavalt vajadusele saavad komponendid (klassid) rakendada. Eesmärk on koodi taaskasutus, kuid viisil kus komponendid ei peaks endale sobimatuid või liiaseid liideste poolt ette kirjutatud nõudeid (lepinguid) täitma.

- **Sõltuvuste pööratud kasutus (*D****ependency injection; Inversion of Control)*** – kõrgema taseme komponendid ei peaks sõltuma madalama taseme komponentide rakendamise detailidest, st. kõrgema taseme komponendid ei tohiks sõltuda otseselt madalama taseme komponentidest (objektidest) vaid sõltuma nende abstraktsioonidest ehk liidestest.

Maksete küsimise teenusepakkuja tarkvara kirjutamisel on kasutatud SOLID printsiipe.

5.3 Rakenduse koodivaramu struktuur

Rakenduse koodivaramu on jaotatud erinevateks .NET projektideks igal ühel neist on eraldiseisev roll.

- Domeen – sisaldab erinevaid domeeniüleseid objekte näiteks olemeid ning liideseid mida võivad kasutada rakenduseüleselt erinevad komponendid.

- DAL – andmejuurdepääsukiht ehk kiht, mis ühildab olemid ning andmevaramu (andmebaasi). Siin eksisteerivad repositooriumid, mille ülesandeks on abstraheerida erinevate andmeolemite töötlemine ettemääratud meetodite abil.

- BLL – äriloogikakiht ehk teenuskiht. Siin kihis on erinevad äriloogikaga rakendavad teenused, mis suhtlevad andmejuurdepääsukihi kui ka kasutajaliidesega (API).

- BLL.DTO – DTO ehk andmeedastusobjektide transleerimise-kaardistamise kiht andmejuurdepääsukihi ning äriloogikakihi vahel. Siin hoiustatakse DTO objekte ning kaardistamise rakenduse konfiguratsiooni.

- API – kasutajaliidese-rakendusliidese kiht millega teenusepakkuja kliendid suhestuvad.

- API.DTO – DTO ehk andmeedastusobjektide transleerimise-kaardistamise kiht äriloogikakihi ning kasutajaliidese-rakendusliidese kihi vahel. Siin hoiustatakse DTO objekte ning kaardistamise rakenduse konfiguratsiooni.

- Testid – rakenduse korrektse toimimise kontrollimiseks vajalikud testid.

5.4 Andmejuurdepääsukiht (DAL)

Tarkvara rakendamisel kasutatud andmejuurdepääsukihi ja äriloogikakihi koodi struktuur järgib Tallinna Tehnikaülikooli õppejõu Andres Käveri „Hajussüsteemid ICD0021“ õppeaine loengus ning harjutustundide raames loodud kihte siduvat koodistruktuuri [38].

5.4.1 Andmete olemid

Andmete hoiustamise objektid ehk olemid kirjeldavad ära üksuse, mis võib eksisteerida eraldiseisvalt. Andmebaaside mõttes võib lugeda selleks organiseeritud elementide kogu. Kuigi andmebaasi mudeli kirjeldamine ei kuulu selle lõputöö koosseisu siis andmejuurdepääsukihi toimimise kirjeldamiseks on tarvis tuua välja ühe olemi osaline kirjeldus.

```
public class Certificate : DomainEntityId
{
    public Guid TypeId { get; set; }
    public Type? Type { get; set; }
    public ICollection<UserCertificate>? Certificates { get; set; }
    [MaxLength(128)]
    public string Name { get; set; } = default!;
    [MaxLength(2048)] public string PublicKey { get; set; } =
default!;
    [MaxLength(256)] public string FingerPrint { get; set; } =
default!;
    …
    public DateTime AddedWhen { get; set; }
    public DateTime? ExpiresWhen { get; set; }
    public bool ServiceEnabled { get; set; }
}
```

Joonis 11 – Sertifikaadi olemi koodinäide

Koodiplokis on kirjeldatud sertifikaadi olemi struktuuri. C# keelel on toetus nullitavate (*nullable*) objektide jaoks (referentsobjekt, mis võib viidata nullile), kõik selliste objektide tüübid on tähistatud ? märgiga. Vastavalt nullitavate (*NULL*) ja mitte nullitavate (*NOT NULL*) objektide määratlusele luuakse andmebaasi struktuur (tabelid) ning määratakse piirangud selles olevatele andmetele.

Tähtis on siinjuures märgata ka sertifikaadi olemi suhet *DomainEntityId* objektiga. *DomainEntityId* on abstraktne klass, mis on loodud eesmärgiga määratleda andmebaasis kasutatavatele olemitele (mida on rohkem kui üks) ühtne struktuur. *IDomainEntityId* on sellele vastav leping.

```
public abstract class DomainEntityId : DomainEntityId<Guid>,
IDomainEntityId
{
}
public abstract class DomainEntityId<TKey> : IDomainEntityId<TKey>
where TKey: struct, IEquatable<TKey>
{
    public TKey Id { get; set; }
}
```

Joonis 12 – Olemiidentifikaatori abstraktse klassi koodinäide

Tähtsamad tähelepanekud:

- DomainEntityId klassist tuleb Guid tüüpi primaarvõti Id (*primary key*).
- **Guid TypeId** kirjeldab võõrvõtit (*foreign key*). See on viide sertifikaadi tüüpi kirjeldavale väljale.
- **ICollection<UserCertificate> Certificate** viitab olemile, kus spetsiifilist sertifikaati võib olla kajastatud mitmeid kordi. Vastav UserCertificate olem sätestab kasutaja ning sertifikaadi suhte läbi aja.

- On olemas väljad, mis ei ole nõutud, näiteks ExpiresWhen. Kõik ilma nullitava märgiga (?) tähistatud väljad on nõutud (*NOT NULL*).
- [MaxLength(xxx)] dekoraator määrab ära välja pikkuse piirangu.

5.4.2 Repositooriumid ja objektrelatsiooni kaardistamine

Selleks, et määratletud olemitest saaks andmebaasi sobilikud objektid kasutame objektrelatsiooni kaardistamise (ORM) raamistikku. ORM raamistik võimaldab arendajatel töötada andmebaasidega, kasutades nendele sobilikku arenduskeskkonda ja programmeerimiskeelt, ilma otsest andmebaasiloogikat (või ka SQL keelt) tundmata.

Tuntud ORM raamistik .NET platvormile on Entity Framework Core (EF Core), mida kasutatakse ka selles projektis. EF Core võimaldab töötada vahekihina ning andmeid transleerida-kaardistada C# objektide (POCO) ning andmebaasi skeemi vahel, toetab mitmeid erinevaid andmebaasiplatvorme, annab paindlikkuse andmebaasi valikul, andmebaasimudeli loomisel, andmepäringute käitamisel ja muul andmebaasisuunalisel suhtlusel. EF Core raamistiku abil loome olemite definitsioonist mudeli ning selle mudeli rakendame juba spetsiifilises andmebaasis.

Repositooriumite ülesanne on andmeid rakendusele vahendada – see tähendab andmeid pärida, otsida, lisada, muuta ja kustutada (CRUD) ilma, et selle kasutaja peaks kokku puutuma otseselt andmebaasi juurdepääsu spetsiifikaga.

Ka repositooriumite rakendamisel taotleme koodi puhtust läbi SOLID printsiipide.

```
public interface IBaseRepository<TEntity, in TKey>
    where TEntity : class, IDomainEntityId<TKey>
    where TKey : struct, IEquatable<TKey>
{
    Task<IEnumerable<TEntity>> AllAsync();
    TEntity? Find(TKey id);
    Task<TEntity?> FindAsync(TKey id);
    TEntity Add(TEntity entity);
    TEntity Update(TEntity entity);
    TEntity Remove(TEntity entity);
    Task<TEntity?> RemoveAsync(TKey id);
}
```

Joonis 13 – Baasrepositooriumi liidese koodinäide

Meil eksisteerib baasileping (*IBaseRepository*), milles kirjeldame kõik põhilised andmebaasiga suhtlemisel vajalikud tegevused ehk CRUD (*Create-Read-Update-Delete*)

meetodid. Baasilepingut rakendavad spetsiifilise olemi leping ning seda omakorda spetsiifilise olemi repositooriumi rakendus (klass).

```
public interface ICertificateRepository :
IBaseRepository<Certificate>,
ICertificateRepositoryCustom<Certificate>
{
}

public interface ICertificateRepositoryCustom<T>
{
    public Task<T?> FindCertificateByThumbprint(string thumbprint);
}
```

Joonis 14 – Sertifikaadirepositooriumi liidese koodinäide

Lisaks baasilepingust tulevatele nõuetele (meetoditele) tahame rakendada ka repositooriumi spetsiifilisi meetodeid nagu näiteks *FindCertificateByThumbprint()* ning selleks on loodud *ICertificateRepositoryCustom* liides.

```
public class CertificateRepository : EFBaseRepository<Certificate,
AppDbContext>, ICertificateRepository
{
    public CertificateRepository(AppDbContext dataContext) :
base(dataContext)
    {
    }
public async Task<Certificate?> FindCertificateByThumbprint(string
thumbprint)
    {
        return await RepositoryDbContext.Certificates.Where(a =>
a.FingerPrint == thumbprint).FirstOrDefaultAsync();
    }
}
```

Joonis 15 – Sertifikaadirepositooriumi koodinäide

Meie rakendus kasutab EF Core raamistikku ning *CertificateRepository* kasutab selle raamistiku võimalusi. Selleks süstitakse repositooriumisse EF Core raamistiku konteksti objekt *AppDbContext* (meie baasklassis nimega *RepositoryDbContext*). Selle objekti abil on võimalik raamistikku kasutada andmebaasiga suhtlemisel. ***AppDBContext*** on *DbContexti* laiendatud objekt, mis määrab soovitud andmebaasi (mudeli) konfiguratsiooni ning võimaldab pärida ning salvestada sellesse andmeid. **Läbi *DbContexti* muutmise (sõltuvuse pööratud kasutamise) on võimalik lihtsasti vahetada aluseks olevaid andmebaase ilma ülejäänud koodistruktuuri muutmata, mis oli üks seatud nõuetest.**

Klass *EFBaseRepository* rakendab põhilised ehk *IBaseRepository* liideses määratud CRUD meetodid ORM raamistiku (EF Core) spetsiifikat arvestades. *CertificateRepository* pärib sellelt klassilt. *EFBaseRepository* kood on leitav lisast 5.

5.5 Äriloogikakiht (BLL)

Siia kihti kuuluvad teenused. Teenused jaotuvad kaheks:

- Andmeteenused, mis suhtlevad otse repositooriumitega ning sealjuures transleerivad-kaardistavad äriloogikakihi andmestruktuure (DTO) olemitesse ja vastupidi.

- Põhiteenused, mis on seotud rakenduse toimimisega nagu näiteks Autoriseerimine, Valideerimine, Logimine, Marsruuter jt.

5.5.1 Tulemuste vahendamine teenuskihil ja veateate struktuur

Teenuskihi ja rakendusliidese kihi vahel vahendatakse sõnumeid erinevate teenuste toimimise ning tulemuste kohta. Sõnumid võivad olla erinevat tüüpi, sisaldada erinevaid objekte ning olenevalt juhust võivad tagastatavad tulemused olla õnnestunud või veakoodiga.

Kuna erinevate tulemuste edastamine on standardne operatsioon, siis rakendame tulemuste vahendamiseks tulemuste raporteerimise (*result report pattern*) mustrit. Selleks loome eraldi andmeobjekti *ServiceResult*, mis võimaldab kapseldada nii õnnestunud tulemused kui ebaõnnestunud tulemused (veateated).

Meie rakenduse puhul on tähtis, et maksete küsimise skeemi sõnumite valideerimise puhul edastaksime nimekirja probleemidest, mis valideerimise puhul ette tulid. Tulemuste kapseldamise objekt võimaldab seda teha.

```
public class ServiceResult<T>
{
    public HttpStatusCode? StatusCode { get; set; }
    public bool IsSuccess { get; set; }
    public T? Data { get; set; }
    public List<Error>? Errors { get; set; }

    public bool HasErrors()
    {
        if (Errors is null)
        {
            return false;
        }
        return Errors.Count > 0;
    }
    public AddErrorSetFail(){…} // lisab Errori ning IsSuccess = väär
    public static ServiceResult<T> Success(T? data){…}
    public static ServiceResult<T> Fail(T? data){…}
    …
}
```

Joonis 16 – Teenuste tulemuste objekt (ServiceResult) koodis

Teenuse ülesanne on uuendada tulemuste objekti. Kui teenuse tegevusel tekkisid vead, siis tagastatakse *ServiceResult* objekt, millel on bool ***IsSuccess*** määratud vastavalt. *ServiceResult* objekt võimaldab vigade puhul kaasa panna nimekirja vigadest ning ka kindlat tüüpi (T) andmed (*Data*). Veateated on samuti standardiseeritud ning kasutavad järgnevat struktuuri (Joonis 17):

```
public class Error
{
    public string? status { get; set; }
    public string? message { get; set; }
    public DateTime? timestamp { get; set; }
    public string? path { get; set; }
    public List<ErrorDetails>? details { get; set; }
}
```

Joonis 17 – Veateate objekti kood

5.5.2 Valideerimise teenus

Valideerimise teenuse eesmärk on võtta maksete küsimise skeemi JSON või XML kujul vahendatud sõnum ning selle struktuuri ning õiget kasutust kontrollida vastavalt maksete küsimise skeemi reeglite raamatule ning rakendusjuhistele.

Valideerimise teenus toimib järgnevalt:

1) Esialgne saabunud sõnum olgu see XML või JSON kujul proovitakse konverteerida POCO objektiks (järgnevalt viidatud kui mudel). Kui see ei

õnnestu, siis on esimene validaator ebaõnnestunud. Kasutatakse ASP.NET enda sisseehitatud POCO mudeliga sidumise tööriista (*Model binding ja validation* [39]**)** ja valideerimise võimekust. Tavaline sõnum, mis vahendatakse teenusepakkujate vahel on järgneva struktuuriga:

```
{
    "callbackUrl": "https://.../cb/srtp/324432",
    "resourceId": "324432",
    "_links": { // HATEOAS funktsionaalsus
        "sepaRequestToPayCancellationRequestUri": {
            "href": "https://.../324432/cancellation-requests"
        }
    },
    "Document": { // Siin asub ISO20022 vormi järgiv SRTP sõnum
        "CdtrPmtActvtnReq": { },
                "PmtInf": [ ]

     },
}
```

Joonis 18 – Ühe tavalise rakendusliidese sõnumi struktuur

Süsteemis tuleb valideerida dokument ehk Euroopa Maksenõukogu poolt ette määratud ISO20022 vormi järgiv sõnum kui teised parameetrid nagu näiteks ressursiidentifikaatorid ning URL-id (kas on õigel kujul ning alati HTTPS). Esimene samm tegeleb justnimelt teiste parameetrite valideerimisega ning dokumendistruktuuri POCO kujule ehk mudelisse sidumisega.

2) POCO ehk mudeli objekt konverteeritakse ettemääratult XML kujule ning XML kujul olev sõnum valideeritakse Euroopa Maksenõukogu poolt määratud XML skeemidefinitsiooni (XSD) vastu. Selle jaoks on eraldi **sõnumite valideerimisteenus**.

3) Kui sõnumi valideerimine skeemi vastu peaks õnnestuma, siis valideeritakse sõnumi struktuur äriloogika vastu (kas kindlaid atribuute võib koos kasutada, kas tegemist on kindla maksetüübiga, kas eksisteerivad kasutajad jt.) ning salvestatakse äriloogikast tulenevad andmed. Selle jaoks on loodud eraldi **äriloogika valideerimisteenus.**

Valideerimisteenused toimivad paketina ning on rakendatud kontrolleris. Andmete vahendamiseks kasutatakse eelnevalt kirjeldatud *ServiceResult* objekti.

Ühe tavalise rakendusliidese kontrolleri kood näeb välja selline (andmestruktuur DS-02 teenusepakkujalt-teenusepakkujale SRTP sõnum):

```
public async Task<IActionResult> PostSrtp([FromBody]
SRTP_InterSP_RTP_DS02_pain_013_001_10 model)
{
// sõnumi valideerimine XML skeemi vastu
var result =
    await _validationService
        .Validate<SRTP_InterSP_RTP_DS02_pain_013_001_10,
SRTP.RTP.pain_013_001_10.Document>(
            model);

if (!result.IsSuccess)
{
    ModelState.AddModelError(result.Errors);
    return ValidationProblem(ModelState);
}

// äriloogika valideerimine
var bsResult = await
_businessRulesValidatorService.DS01BusinessRuleValidator.Validate(mode
l);
if (!bsResult.IsSuccess)
{
    ModelState.AddModelError(bsResult.Errors);
    return ValidationProblem(ModelState);
}
// valideerimine õnnestus, järgmised sammud …
}
```

Joonis 19 – Ühe tavalise rakendusliidese kontrolleri valideerimiskood

Sõnumite valideerimisteenus

Kontrollerisse saabunud sõnumid konverteeritakse kõigepealt POCO objekti kujule. Kui esialgne konversioon toimib, siis jätkatakse sõnumite valideerimist XSD skeemi vastu. Kompleksust lisab nõue, et toetama peab nii XML kui JSON sõnumeid samaaegselt ning sõnumistruktuuris eksisteerib *choice* ehk valikulisi andmestruktuure. Samuti võib eksisteerida erinevaid versioone Euroopa Maksenõukogu XSD skeemidest ning nende järgi loodud andmestruktuuridest (mudelist). Eesmärk on toetada erinevaid XSD skeeme ning andmestruktuure ning mitte kasutada liiast koodi. Seetõttu valideerimisteenus abstraheerib kahte tüüpi T1 ja T2. T1 on ümbrise (*wrapper*) mudeli ehk ümbrise tüüp ning T2 on kindlast Euroopa Maksenõukogu sõnumi XML skeemi versioonist tuletatud POCO objekti (dokumendi) tüüp.

Valideerimisteenus on rakendatud järgnevalt:

```
public interface IValidationService
{
public Task<ServiceResult<T1>> Validate<T1, T2>(T1 model) where T1 :
IBaseRtpDocument<T2>;
}

public async Task<ServiceResult<T1>> Validate<T1, T2>(T1 model) where
T1 : IBaseRtpDocument<T2>
{
    try
    {
        var serializedPoco = _xmlSerializer.Serialize(model.Document);
        var result = await ValidateXmlSchema<T1,T2>(serializedPoco);

        if (result.IsSuccess)
        {
            return ServiceResult<T1>.Success(result.StatusCode);
        }

        return ServiceResult<T1>.Fail(result.StatusCode,
result.Errors);
    }
    catch (Exception ex)
    {
        return ServiceResult<T1>.Fail(ex.ToString(),
ErrorMessage.SchemaValidationError,
HttpStatusCode.InternalServerError);
    }
}
```

Joonis 20 – Sõnumite valideerimisteenuse kood

XML skeemidefinitsioone on mitmeid ning need võivad olla erinevad vastavalt sõnumistruktuuri versioonile. Näiteks Euroopa Maksenõukogu poolt on teenusepakkujate vaheliste sõnumite (v3.0) jaoks on ette määratud kolmteist erinevat sõnumite skeemidefinitsiooni (XSD).

Tahame valideerida erinevat tüüpi mudelite sisu (rakendusliidesest saabunud JSON või XML sõnum) erinevate XML skeemide vastu. Selle jaoks on loodud eraldi sõnumite valideerimisteenus *SchemaValidationService*, mis järgib *IValidationService* lepingut. *SchemaValidationService* võimaldab leida etteantud ümbrise T1 tüübi ning dokumendi tüübi T2 järgi vastava XML-i skeemi, mille vastu seda kindlat sõnumitüüpi valideerida.

SchemaValidationService kasutab õige skeemidefinitsiooni leidmiseks skeemide leidmise teenust *SchemaResolverService*. Skeemide leidmise on kataloogteenus, mis võimaldab vastavalt objekti tüüpidele leida XSD skeemifaili asukoha ning nimeruumi. Nii skeemifaili asukohta kui nimeruumi on tarvis, et mudelit vastavalt skeemile valideerida.

Skeemi valideerimine on koodis kirjeldatud *ValidateXmlSchema* meetodis järgnevalt: esiti leitakse vastavalt sõnumitüüpidele õige skeemidefinitsioon ja vastava nimeruumi ning nende andmete põhjal luuakse uus skeemivalideerimise XML luger. XML lugeri instants salvestab vigade tekkimisel vead *ServiceResult* objekti (*result.AddErrorSetFail(…)*) ning seeläbi seab *ServiceResult* booli IsSuccess väärtuse vastavalt.

```
private async Task<ServiceResult<string>> ValidateXmlSchema<T1,
T2>(String xmlContent, XmlSchemaSet? schema = null){
…
// leiame ümbrise (wrapper) ja tüübile vastava XSD skeemi
var schemaInfo = await
_resolverDictionaryService.SchemaResolverService.GetByType(typeof(T1),
typeof(T2));
schema = new XmlSchemaSet();
schema.Add(schemaInfo.Uri, schemaInfo.FilePath);
…
// uus XML lugeja, mis valideerib XML sõnumit XSD skeemi vastu
var xmlReaderSettings = new XmlReaderSettings()
{
    Schemas = schema,
    ValidationType = ValidationType.Schema,
    Async = true,
    IgnoreWhitespace = true,
};
// valideerimisel vigade leidmisel lisatakse vead ServiceResult objekt
xmlReaderSettings.ValidationEventHandler += (sender, e) =>
{
    result.AddErrorSet($"{e.Message} on line {e.Exception.LineNumber}
column {e.Exception.LinePosition}",
        $"{sender}: schema validation error");
};
// lugeda kuni pole enam midagi lugeda
using (XmlReader reader = XmlReader.Create(new
StringReader(xmlContent), xmlReaderSettings))
{
        while (await reader.ReadAsync()) { }
}
…
return result;
}
```

Joonis 21 – RTP sõnumi valideerimine skeemidefinitsiooni vastu

Äriloogika valideerimisteenus

Äriloogika valideerimisteenus kasutab spetsifikatsiooni mustrit [40]. Spetsifikatsiooni muster võimaldab määratleda ärireeglid eraldiseisvatesse struktuuridesse ning vajadusel

neid kombineerida läbi Booli loogika (AND, OR, NOT, ..) seeläbi võimaldades loogikat efektiivselt taaskasutada.

Rakendusel on tarvis kontrollida maksete küsimise skeemi sõnumeid, et aru saada millist tüüpi sõnumiga on tegu ning millist äriloogikat peaks sõnumile rakendama. Näiteks SRTP sõnum ***CdtrPmtActvtnReq*** (DS-01, DS-02, DS-03) võib sisaldada erinevat tüüpi makset. Et suurendada keerukust, siis makse tüüpi ei määratleta ära ühe atribuudi või muutujana sõnumis vaid see tuleneb sõnumistruktuurist endast:

1. Kui sõnumis on ainult üks *PmtInf* blokk ja ainult üks *CrdtTrxTf* blokk, siis on tegemist ühekordse maksega (üks summa ja üks krediidikanne).

2. Kui sõnumis on üks *PmtInf* blokk kuid mitu *CrdtTrxTf* blokki, siis võib tegemist olla osamaksetega (järelmaksuga ehk mitu krediidikannet).

3. Kui sõnumis on mitu *PmtInf* blokki, esimeses blokis on ainult üks *CrdtTrfTx* blokk ning teises blokis on osamaksete blokk (*PmtInf* blokk mitme eri *CrdtTrfTx* blokiga), siis on tegemist hübriidmaksega (lubatud ühekordne kui järelmaks, maksja valib).

4. Kui sõnumis on makse summa *Amt* määratletud kui 0 (atribuut AT-T002), eksisteerib atribuut AT-019 (*CrdtTrxTf->RmtInf->RfrdDocAmt->CdtNoteAmt*), siis võib tegu olla krediiditeavitusega (makse küsija lubab maksjale krediiti).

Järgnevalt on kirjeldatud ühekordse makse spetsifikatsiooni kontroll koodis.

```
public class OneTimePaymentSpecification<T> : LinqSpecification<T>
where T : SRTP.RTP.Document
{
    public override (Expression<Func<T, bool>>, Error error)
AsExpression()
    {
        var error = Error.Create(ErrorMessages.NotOneTimePay,
            ErrorMessages.PaymentStructureNotValidStatus);
        return (model => model.CdtrPmtActvtnReq.PmtInf.Count == 1 &&

model.CdtrPmtActvtnReq.PmtInf[0].CdtTrfTx.Count == 1 &&
                        model.CdtrPmtActvtnReq.PmtInf[0].ReqdExctnDt
!= null, error) } }
```

Joonis 22 – Spetsifikatsiooni mustri koodinäide äriloogika kontrollimiseks

Ning spetsifikatsiooni kasutus äriloogika valideerimisteenuses:

```
protected virtual async Task ValidatePaymentType(T model,
ServiceResult<RequestMessage> result)
{
    var oneTimePaymentSpec = new OneTimePaymentSpecification<T>();
    var installmentPaymentSpec = new
InstallmentPaymentSpecification<T>();
    var mixedPaymentSpec = new MixedPaymentSpecification<T>();
    var creditNoteSpec = new CreditNoteSpecification<T>();

    if (oneTimePaymentSpec.IsSatisfiedBy(model))
    {
        result.Data.PaymentType = EPayType.SRTP_ONETIME;
        return;
    }

    if (installmentPaymentSpec.IsSatisfiedBy(model))
    {
        result.Data.PaymentType = EPayType.SRTP_INSTALMENT;
        return;
    }

    if (mixedPaymentSpec.IsSatisfiedBy(model))
    {
        result.Data.PaymentType = EPayType.SRTP_MIXED;
        return;
    }

    if (creditNoteSpec.IsSatisfiedBy(model))
    {
        result.Data.PaymentType = EPayType.SRTP_CREDITNOTE;
        return;
    }

    result.AddErrorSetFail(
        ErrorMessages.PaymentStructureNoMatch,
        ErrorMessages.PaymentStructureNotValidStatus)
    }
```

Joonis 23 – Spetsifikatsiooni mustri kasutus maksetüübi valideerimisel

Spetsifikatsiooni mustri kasutamine võimaldab kasutada Booli loogikat erinevate spetsifikatsioonide vahel, et siduda need üheks spetsifikatsiooniks. Seda ühte spetsifikatsiooni on võimalik vastavalt mudelile valideerida.

Näiteks selleks, et määrata maksete küsimise skeemile omaseks saanud aktsepteeri nüüd - maksa kohe (ANPN), aktsepteeri nüüd – maksa hiljem (ANPL), aktsepteeri hiljem – maksa kohe (ALPN) või aktsepteeri hiljem – maksa hiljem (ALPL) maksetüüpi, siis tuleb võrrelda makse küsija poolt määratud aegumistähtaega ja soovitud maksekorralduse täitmise aega. Olenevalt sõnumistruktuurist (kas on tegemist järelmaksu võimalusega või mitte) võivad need tähtajad olla määratud erinevates andmestruktuuride osades ning eksisteerib järjekordne erisus.

Joonisel 24 on toodud välja koodinäide, mis kirjeldab kuidas spetsifikatsiooni mustrit on võimalik kasutada, et läbi Booli loogika siduda mitmed erinevad tingimused kokku ning seeläbi parandada koodi loetavust ning taaskasutamise võimalust.

```
protected virtual async Task ValidatePaymentDateTimes(T model,
    ServiceResult<RequestMessage> result)
{
// Spetsifikatsioonide initsialiseerimine

          var xpryDtTmSpecified = new RtpXpryDateIsSetAndValid<T>();
          …
          var instalmentPaymentSpec = new
       InstallmentPaymentSpecification<T>();
          var acceptLaterPayNowSpec = new AcceptLaterPayNowSpecified<T>();
          var acceptLaterPayLaterSpec = new
       AcceptLaterPayLaterSpecified<T>();
          var acceptNowPayLaterSpec = new AcceptNowPayLaterSpecified<T>();
          var acceptNowPayNowSpec = new AcceptNowPayNowSpecified<T>();

       // Spetsif. sidumine Booli loogikaga Aktsepteeri hiljem – Maksa kohe
          var alpnNotInstalment =
       creationDtTmSpecified.And(xpryDtTmSpecified).And(pmtInfReqdExctDtSpeci
       fied).And(acceptLaterPayNowSpec).AndNot(instalmentPaymentSpec);
          var alpnInstalment =
       creationDtTmSpecified.And(xpryDtTmSpecified).And(cdtrTrfTxReqdExctDtSp
       ecified).And(acceptLaterPayNowSpec).And(instalmentPaymentSpec);
          var alpn = alpnInstalment.Or(alpnNotInstalment);

       // Mudeli valideerimine spetsifikatsiooni vastu
          if (alpn.IsSatisfiedBy(model))
          {
              result.Data.PaymentCondition =
       EPayConditionType.SRTP_COND_ALPN;
              return;
          }

       // Spetsif. sidumine Booli loogikaga Aktsepteeri hiljem – Maksa hiljem
          var alplInstalment = creationDtTmSpecified.And(xpryDtTmSpecified)
       .And(pmtInfReqdExctDtSpecified).And(acceptLaterPayLaterSpec).AndNot(in
       stalmentPaymentSpec);
          var alplNotInstalment =
       creationDtTmSpecified.And(xpryDtTmSpecified)
       .And(cdtrTrfTxReqdExctDtSpecified).And(acceptLaterPayLaterSpec).And(in
       stalmentPaymentSpec);
          var alpl = alplInstalment.Or(alplNotInstalment);

       // Mudeli valideerimine spetsifikatsiooni vastu
          if (alpl.IsSatisfiedBy(model))
          {
              result.Data.PaymentCondition =
       EPayConditionType.SRTP_COND_ALPL;
              return;
          }
       …
       result.AddErrorSet(…)
       }
```

Joonis 24 – Spetsifikatsioonide sidumine läbi Booli loogika maksetüübi kontrollil

5.5.3 Taustal toimuvad teenused – olekureporter ja olekujälgija

Reeglite raamatus on kirjeldatud nõue, mis sätestab, et aegunud kuid mitte aktsepteeritud maksete küsimise sõnumid tuleb maksja teenusepakkuja poolt tühistada ning selle kohta saata tühistamisteavitus.

Sõnumijälgija sõnumite tühistamisteenus *SrtpRejectService* rakendab Microsofti *BackgroundService* (*IHostedService*) [41] baasklassi, et luua taustal toimiv teenus. Taustateenus toimib nii, et iga minuti alguses päritakse andmebaasist nimekiri SRTP sõnumitest (aegumise järgi järjestatud), mis tuleb tühistada ning vastavalt ajakavale tühistatakse. Ka teised taustal toimuvad teenused kasutavad *IHostedService* võimalusi.

```
public class SrtpRejectService : BackgroundService
{
    …
    protected override async Task ExecuteAsync(CancellationToken
stoppingToken)
    {        while (!stoppingToken.IsCancellationRequested)
        {
            var currentTime = DateTime.UtcNow;
            var jobsToRun = await
scheduler.GetJobsToRunAsync(currentTime);
            foreach (var job in jobsToRun)
            {
                var delay = (job.ExpiryTime - DateTime.UtcNow);
                if (delay <= TimeSpan.Zero)
                {
                    await executor.ExecuteJobAsync(job);
                }
                else
                {
                    await Task.Delay(delay.Value, stoppingToken);
                    await executor.ExecuteJobAsync(job);
                }
                if (stoppingToken.IsCancellationRequested)
                {
                    break;
                }
            }
            var nextMinute = currentTime.AddMinutes(1).AddSeconds(-
currentTime.Second).AddMilliseconds(-currentTime.Millisecond);
            var delayUntilNextMinute = nextMinute - DateTime.UtcNow;
            if (delayUntilNextMinute < TimeSpan.Zero)
                delayUntilNextMinute = TimeSpan.Zero;
            await Task.Delay(delayUntilNextMinute, stoppingToken);
        }
    }
}
```

Joonis 25 – Perioodilise SRTP tühistamise taustateenuse kood

5.5.4 Sõnumihalduri ja marsruutimise teenused. Konnektorid.

Sõnumihaldur

Sõnumihaldur seob kokku mitmed erinevaid sõnumite loomise, muutmise, täiendamisega seotud teenuseid. Üheks sõnumite haldamise teenuse ülesandeks on rakendusliidesest saabunud sõnumite töötlemine pärast valideerimise sammu ja enne edastamist. Teiseks ülesandeks on sõnumistruktuuri andmebaasikõlbulikuks muutmine enne salvestamist. Kolmandaks ülesandeks on uute standardsete sõnumite loomine, mida oleks võimalik partneritele (teenusepakkujad, kliendid) edastada ehk marsruuditavate sõnumite loomine.

Marsruuditav sõnum on sõnum, mida on võimalik läbi *HttpClient* rakenduse HTTP päringu kujul saata ehk marsruuditav sõnum sisaldab *HttpRequestMessage* kujul andmeid, mida klientrakendus oskab kasutada. *HttpRequestMessage* läbi määratakse ära HTTP päringu päised, päringu aadress (URI), HTTP protokolli sätted ning päringu sisu. Lisaks *HttpRequestMessage* sisalduvale on teenustel ning HTTP klientrakendusel tarvis teada ka informatsiooni ka kliendi kui serveri sertifikaatide kohta, sihtkoha ja teiste parameetrite kohta.

Kuna *HttpRequestMessage* loomisel tegutsevad mitmed erinevad teenused (sõnumihaldur loob sõnumi, marsruuter leiab sihtkoha ning konnektor saadab sihtkohta), siis marsruuditavat sõnumit ning andmeid, millest marsruuditav sõnum loodi läheb tarvis kasutada mitmel erineval teenusel. Selle jaoks, et see informatsioon oleks meie sõnumites saadaval oleme loonud *IRoutableMessage* kontrakti (liidese). Kõik klassid, mis rakendavad seda liidest, sisaldavad andmeid ning meetodeid, et uus marsruuditav sõnum oleks võimalik luua, see saata ning selle vastust ka tagastada standardsel viisil.

IRoutableMessage liides määratleb nii päringu kui vastuse jaoks eraldi objektid ning seeläbi võimaldab sõnumite saatmisega tegelevatel teenustel siduda seda liidest rakendava objektiga nii HTTP päringu kui vastuse.

```
public interface IRoutableMessage
{
    public EAddressType? AddressType { get; set; }
    public HttpRequestMessage? Request { get; set; }
    public HttpResponseMessage? Response { get; set; }

    public X509Certificate2? ClientCertificate { get; set; }
    public string? ServerCertificateThumbprint { get; set; }

    public bool UseClientCertificate { get; set; }
    public bool ValidateServerCertificate { get; set; }
    public AddressBookDTO? To { get; set; }
}
```

Joonis 26 - Marsruuditava sõnumi liides (leping)

Kirjeldatud *EAddressType* määrab ära sihtaadressi tüübi – kas tegemist on RTP päringu, tühistamispalve (RfC) päringu, olekuraporti, *callbacki* või mõne teise aadressi tüübiga kuhu tuleb ühenduda. Aadressi tüüp on tähtis määratleda seetõttu, et eristada ühel teenusepakkujal olevaid erinevaid lõpp-punktide aadresse, mis võib olla igal teenusepakkujal ning päringutüübil (sõnumil) erinev. Aadressi tüüpi kasutab marsruuterteenus, et leida õige lõpp-punkti aadress andmebaasist ning uuendada *HttpRequestMessage (RequestURI)* vastavalt.

Säte bool *UseClientCertificate* määrab ära kas konnektor (*HTTPClient)* peaks kasutama objektis määratud kliendisertifikaati ühendamisel, bool *ValidateServerCertificate* määrab kas serveri sertifikaadi tuleb võrrelda objektis oleva sõrmejäljega (*thumbprint)*. Mõlemad on vajalikud teenusepakkujate vahelise HTTPS ühenduse valideerimiseks.

Üheks marsruuditavaks objektiks on RTP loomise sõnum *RequestMessage*, mis seob omavahel erinevad vajalikud objektid nagu näiteks andmebaasiobjekt (ResourceDTO) või HTTP kontekstiobjekt ning mida on tarvis vahendada erinevate teenuste vahel.

```
public class RequestMessage : IMessage, IRoutableMessage {
      public HttpContext Context { get; set; }
      public ResourceDTO Resource { get; set; }
      public EStatusType? MessageStatus { get; set; }
      public List<string> RejectErrors { get; set; } = new
List<string>();
      ...
      public HttpRequestMessage? Request { get; set; }
      public HttpResponseMessage? Response { get; set; }
      public X509Certificate2? ClientCertificate { get; set; }
      ...
}
```

Joonis 27 - Uue RTP päringu objekt (RtpRequest)

MessageHandlerService seob mitmed erinevad sõnumite haldamisega seotud teenused:

```
public interface IMessageHandlerService
{
    public IPayeeRtpResourceService PayeeRtpResourceService { get; }
// sõnumist SRTP ressursi loomine andmebaasi kujule (makse küsija)
    public IServiceProviderRtpResourceService
ServiceProviderRtpResourceService { get; } //  sõnumist SRTP ressursi
loomine andmebaasi kujule (teenusepakkujate vaheline)
    public IRoutableRequestService RoutableRequestService { get; } //
Marsruuditava SRTP palve sõnumi loomise teenus
    public IRoutableResponseService RoutableResponseService { get; }
// Marsruuditava SRTP vastuse sõnumi loomise teenus
    public IRtpRejectService RtpRejectService { get; } // "R" sõnumite
ehk keeldumise sõnumite loomise teenus
    ...
}
```

Joonis 28 – IMessageHandlerService sõnumihalduri liides

Järgnevalt kirjeldan teenuse *RoutableRequestService* toimimist, mille ülesanne on luua SRTP tüüpi sõnumist (DS-01, DS-02) marsruuditav sõnum.

```
public interface IRoutableRequestService
{
      // Loo DS-01 sõnumist DS-02 marsruuditav sõnum
      public Task<ServiceResult<RequestMessage>>
      Create(SRTP_Payee_RTP_DS01_pain_013_001_10 model,
      ServiceResult<RequestMessage> result);
      …
}
```

Joonis 29 – Marsruuditava sõnumi loomisteenuse liides

Teenus *RoutableRequestService* võtab sisendina mudeli DS-01 ehk makse küsija poolt loodud uue SRTP sõnumi ning konverteerib selle DS-02 vormi. DS-02 ehk uue SRTP ressursi loomise sõnum vormitakse ümber ning edastatakse maksja teenusepakkujale (PS-01.04).

Pärast andmestruktuuri konverteerimist luuakse vastavalt ressursile *Callback*-URL ja HATEOAS blokk, *serialiseeritakse* andmemudel JSON või XML vormi, lisatakse andmed *HttpRequestMessage* tüüpi objekti, lisatakse objekti ka tarvilikud HTTP päised.

Pärast sõnumiloome protsessi tagastatakse *ServiceResult* objekti mähitud *RequestMessage* tüüpi vastus. Vastavalt sellele kas sõnumi loomine õnnestus või mitte

(*ServiceResult.isSuccess*) jätkub programmi töövoog. Marsruuditava sõnumi loomise teenuse koodinäide on leitav lisast 7.

Marsruutimise teenus

Marsruutimise mooduli ülesanne on leida vastavalt sõnumiadressaadile õige sihtkoht kuhu sõnum edasi suunata. Maksete küsimise skeemi adressaat (maksja) sõnumis (DS-01, DS-02, DS-03) on tavaliselt määratud *DebtorAcct* või *DebtorAgent* blokis identifikaatorina ning vastavalt sellele identifikaatorile tuleb andmebaasist leida adressaadi teenusepakkuja. Olenevalt sõnumi tüübist võib teenusepakkuja poole pöördumiseks olla mitmeid erinevaid aadresse. Näiteks võib tegemist olla uue SRTP sõnumiga, SRTP olekuraporti sooviga, tühistamispalvega või tühistamispalve olekuraporti sooviga. Vastavalt sõnumile tuleb leida õige teenusepakkuja lõpp-punkti aadress.

Marsruuter kasutab sõnumiteenusest saabunud *ServiceResult<RequestMessage>* tüüpi objekti, et valmistada sõnum ette edastamiseks konnektori läbi ehk leida õige lõpp-punkti aadress andmebaasist vastavalt sõnumitüübile ning adressaadile ning lisada see andmeobjekti. Järgnevalt on kirjeldatud ruuteri tööpõhimõtet:

```
 public async Task<ServiceResult<T>> RouteMessage<T>(ServiceResult<T>
message) where T : IMessage, IRoutableMessage
    {
      // 1. Vastavalt RequestMessage (IRoutableMessage) sõnumis
olevale aadressile ja aadressitüübile leiame kellele peaks sõnumi
edastama (kasutaja)
      // 2. Leiame andmebaasist selle kasutaja (teenusepakkuja) seda
tüüpi aadressi (AddressDTO)
      // 3. Määrame RequestMessage.Request.RequestUri
(HttpRequestMessage.RequestUri) ehk HTTP päringu aadressi AddressDTO
järgi
      // 4. Määrame RequestMessage.To võrdseks AddressDTO objektiga,
mis võimaldab hilisemalt teistel teenustel aadressi taaskasutada
(näiteks sertifikaadi leidmiseks)
    }
```

Joonis 30 - Marsruuteri tööpõhimõtte kirjeldus

Konnektorteenus

Konnektorteenuse ülesandeks on ühenduse loomine kliendina ning selle ühenduse läbi andmete vahendamine. Kasutusel on kolme tüüpi konnektoreid: teenusepakkuja konnektor, tegevuskava juhi (OSM) konnektor ning kliendikonnektorid.

Konnektorteenused põhinevad *HttpClient* rakendamisel. *HttpClient* võimaldab luua HTTP päringuid.

Konnektorteenused järgivad *IConnector* lepingut. Konnektor võtab sisendiks *HttpRequestMessage* tüüpi objekti, sertifikaadi ning proovib ühenduda *HttpRequestMessage* objektis määratud aadressile (URI). Kui sertifikaati konnektorteenusele ette ei anta, siis kasutatakse ettemääratud vaikimisi sertifikaati!

Konnektorteenusele võib anda ette ka serveri sertifikaadi näpujälje (*thumbprint*), sellel juhul valideeritakse ka serveri näpujälg. See on vajalik teenusepakkujate vahelise ühenduse valideerimiseks, mille puhul tuleb kontrollida nii kliendi kui serveri sertifikaati!

Teenusepakkujate vahelise konnektorteenuse koodi võib leida lisast nr. 8. Konnektorteenus on rakendatud vastavalt *IConnector* lepingule (Joonis 32).

```
public interface IConnector
{
        Task<ServiceResult<HttpResponseMessage>>
MakeApiRequestResponseAsync(HttpRequestMessage request,
                X509Certificate2? clientCertificate = null, string?
serverCertificateThumbprint = null);
        Task<ServiceResult<T>>
MakeApiRequestResponseAsync<T>(ServiceResult<T> request) where T :
IMessage, IRoutableMessage;
        Task MakeApiRequestAsync(HttpRequestMessage request,
X509Certificate2? certificate = null);
}
```

Joonis 31 – Konnektorteenuse leping

5.6 Rakendusliidese ehk kasutajaliidese kiht (API / UI)

5.6.1 Veebiserver Kestrel

Veebiserver Kestrel on SRTP teenusepakkuja tarkvara alus, millele on peale ehitatud REST rakendusliidese kontrollerid. Kestrel on konfigureeritud lubama vaid HTTPS protokolli sõnumeid. Lisaks eksisteerib nõue, et veebiserver peab vastavalt rakendusliidese lõpp-punktile, mille poole pöördutakse valima sobiliku serverisertifikaadi. See on tähtis seetõttu, et peab toimima kahepoolne sertifikaadi autentimine- nii kliendi kui serveri puhul ning sertifikaadid erinevatel teenustel (teenusepakkuja, makse küsija, maksja) on erinevad. Kuna prototüübi puhul on kasutusel vaid üks kohalik arenduskeskkond (kaks *Kestreli* instantsi) ning puhverservereid-

proksisid (API lüüse) ei kasutata, siis peab seda ülesannet suutma täita see üks kohalik arenduskeskkond.

Erineva kontrolleri rakendusliides peab vastama päringule erineva sertifikaadiga. Üks võimalus seda saavutada on läbi serverinime näidustamise läbi HTTP *Host:* päise. Järgnev konfiguratsioon kontrollib iga päringu puhul päringult sissetulevat *Host* päist ning määrab vastavalt sellele serveri sertifikaadi.

Veebiserveri konfiguratsioon määratakse ära *appsettings.json* konfiguratsioonis, sealhulgas millist porti ning sertifikaati peaks instants kasutama. Prototüübil on iga teenusepakkuja instantsi jaoks on eraldi veebiserveri instants.

```
builder.WebHost.ConfigureKestrel(options =>
{
    options.ListenAnyIP(kestrelServerPort, listenOptions =>
    {
        listenOptions.UseHttps(httpsOptions =>
        {
// Nõua kliendipoolset autentimist
httpsOptions.ClientCertificateMode =
ClientCertificateMode.RequireCertificate;
// Määra serveri sertifikaadid
            var payeesp =
CertificateLoader.LoadFromStoreCert(payeeSpCertificateName,
                "My", StoreLocation.CurrentUser, allowInvalid: false);
            var payersp =
CertificateLoader.LoadFromStoreCert(payerSpCertificateName,
                "My", StoreLocation.CurrentUser, allowInvalid: false);
            var intersp =
CertificateLoader.LoadFromStoreCert(interspCertificateName,
                "My", StoreLocation.CurrentUser, allowInvalid: false);
            var certs = new Dictionary<string,
X509Certificate2>(StringComparer.OrdinalIgnoreCase)
            {
                [payerSpCertificateName] = payersp,
                [payeeSpCertificateName] = payeesp,
                [interspCertificateName] = intersp
            };
            // Sertifikaat valitakse vastavalt Host nimele
            httpsOptions.ServerCertificateSelector = (_, name) =>
            {
                if (name is not null && certs.TryGetValue(name, out
var cert))
                {
                    return cert;
                }
                return localHostCert;
            };
        });
    });
});
```

Joonis 32 – Klientsertifikaadi autentimine ja Host-päise põhine sertifikaadi valik

5.6.2 Autentimine ja autoriseerimine

Teenusepakkuja klientideks on makse küsijad, teised teenusepakkujad ning maksjad. Vastavalt turvaraamistikus sätestatule põhineb autentimine ja autoriseerimine vastavalt sertifikaatidele. Suhtlusel peab autentima nii klienti kui serverit.

Prototüübi puhul on esialgse autentimise ülesanne jäetud *Kestreli* veebiserverile, mis valideerib kas tema poole pöördutakse sertifikaadiga, mis on usaldatud ning kehtiv. Kui sertifikaadi omanikku või sertifikaadi väljastanud sertifitseerimisasutust ei usaldata, siis autoriseerimise sammuni ei jõuta ning rakendusliideseid kasutada kliendil ei õnnestu.

Autoriseerimine toimub kolmes etapis:

1) Kontrollida kas sertifikaat on andmebaasis ning lubatud (*ServiceEnabled*). See toimub sertifikaadi sõrmejälje (*thumbprinti*) võrdlemisega andmebaasis leitule. Vastavalt nõuetele on iga sertifikaadiga on seotud üks kindel klient ja kliendi identifikaator.

2) Vastavalt sertifikaadi sõrmejäljele (*thumbprintile*) ning teistele sertifikaadist tuletatud andmetele leitakse kasutaja ning määratakse temale vastavad õigused (*claims*).

3) Vastavalt kasutaja õigustele toimib ressursipõhine autoriseerimine (kasutaja peab saama ligi ainult ressurssidele, millele tal on õigus ligi saada). See on rakendatud kasutades Microsofti ASP.NET poliitikapõhist autoriseerimise võimekust [42].

Autoriseerimise esimene ning teine samm on rakendatud läbi veebiserveri vahevara *CertAuthMiddleware*, mida käitatakse enne igat päringut ning mille ülesanne on kontrollida sertifikaat, kontrolli õnnestumisel luua süsteemi kasutaja õigustega andmeobjekt ning lisada järgnevatele teenustele tarbimiseks kontekstiobjekti *HttpContext*.

CertAuthMiddleware palub autentimise ülesannet täita teenuste kihil asetseval Cert*AuthService* teenusel. *CertAuthService* kasutab ära *HttpContext* objekti (järgnevalt kontekst), mis sisaldab lisaks päringu-vastuse sisule suurt hulka lisainformatsiooni, sealhulgas päringu päiseid, ühenduse enda sealhulgas kliendisertifikaatide kohta käivat informatsiooni jpt. *CertAuthService* kood on leitav lisas nr. 6.

Järgnevalt on kirjeldatud *CertAuthMiddleware* toimimispõhimõtet, kus vahevara palub *CertAuthService* teenusel leida konteksti põhiselt kasutajale *ClaimsPrincipal* objekt, mis sisaldab kasutaja õiguseid. Teenuste vahel edastatakse informatsioon kasutades varasemalt kirjeldatud *ServiceResult* objekti. Kui autoriseerimine õnnestus, siis *ServiceResult.Data* andmeobjekt tüüpi ClaimsPrincipal ning *ServiceResult.IsSuccess* Bool on tõene, samuti on *ClaimsPrincipal* põhine *isAuthenticated* bool määratud kui tõene. Õnnestumisel seotakse *ClaimsPrincipal* konteksti objektiga *User* ning seejärel saab ülejäänud rakendus juba selle objekti abil kasutaja õiguseid kontrollida. Ebaõnnestumisel saadetakse rakendusliidese kasutajale *ServiceResult.Errors* tüüpi vastus, mis on kas XML või JSON vormis vastavalt kliendi poolt määratletud HTTP *Accept:* päisele.

```
public class CertAuthMiddleware : IMiddleware
{
    public async Task InvokeAsync(HttpContext context, RequestDelegate
next)
{
    // Autendi CertAuthService vastu
    var result = await _certAuthService.ValidateAuthorize(context);
    if (result.Data is not null)
        context.User = result.Data;
    // Kui autentimine peaks olema õnnestunud
    if (result is { IsSuccess: true, Data: not null } &&
context.User?.Identity?.IsAuthenticated is true)
    {
         await next(context);
    }
    else // Kui ebaõnnestub siis loo vastus (XML või JSON kujul)
    {
        context.Response.StatusCode =
StatusCodes.Status401Unauthorized;
        context.Response.ContentType = context.Request.Headers.Accept
== SRTP.Constants.ContentTypes.Xml
            ? SRTP.Constants.ContentTypes.Xml
            : SRTP.Constants.ContentTypes.Json;
        // Vastuse tüüp oleneb Accept päisest
        var responseBody = context.Request.Headers.Accept ==
SRTP.Constants.ContentTypes.Xml
            ? _xmlSerializer.Serialize(result.Errors)
            : _jsonSerializer.Serialize(result.Errors);
        … // Logimine ja teised tegevused
        await context.Response.WriteAsync(responseBody);
    }
}
```

Joonis 33 – Sertifikaadipõhise autoriseerimise veebiserveri vahevara kood

Kolmanda autoriseerimise sammu jaoks kasutame Microsofti poliitikapõhist autoriseerimist. Poliitikapõhine autoriseerimine võimaldab määratleda nõuded, kirjeldada ära nende nõuete sisu (nõuete haldurid) ning siduda need nõuded üheks

poliitikaks. Poliitikas võib olla mitmeid nõudeid koos ning poliitika võib vastavalt kasutaja õigustele lubada või mitte lubada ligipääsu kindlale ressursile. Näiteks on poliitikapõhise autoriseerimisega lahendatud erinevate kontrolleritele ligipääsu nõue, et autoriseeritud kasutaja peab olema kindlas rollis (näiteks teenusepakkuja).

Järgnev näide on kuidas sai rakendatud autoriseerimispoliitika ühe ressursi päringu kohta. Poliitika sätestab, et kui kasutaja pärib informatsiooni kindla ressursi kohta, siis kasutajal peab eksisteerima ressursiga lubav suhe (*SrtpReadRequirement*). *SrtpUserAclReqHandler* nõuete haldur määrab ära *SrtpReadRequirement* nõude kontrollimisel tehtavad tegevused. Kontrollitakse kas andmebaasist leitud ressursiga on kasutajal seos. Vastavalt on nõue täidetud (*context.Succeed*) või mitte täidetud (*context.Fail*).

```
public class SrtpUserAclReqHandler :
AuthorizationHandler<SrtpReadRequirement, ResourceDTO>,
IAuthorizationRequirement
{
    protected override Task
HandleRequirementAsync(AuthorizationHandlerContext context,
SrtpReadRequirement requirement, ResourceDTO resource)
    {   // Kui kasutaja on seotud ResourceDTOga siis luba ressursi
kasutada
        if (resource.Users != null && resource.Users.Any(r => r.UserId
== context.User.GetUserId()))
        {
            context.Succeed(requirement);
        }
        else // Vastasel juhul teavita kasutajat veast
        {
            context.Fail(new AuthorizationFailureReason(this, "User
does not have access to specified resource."));
        }
        return Task.CompletedTask;
    }
}
```

Joonis 34 – Poliitikapõhise nõuete halduri autoriseerimise kood

Poliitikapõhine autoriseerimisteenus seob nõude ning poliitika järgnevalt:

```
builder.Services.AddAuthorization(options =>
{
       options.AddPolicy("RequireServiceProviderPolicy",
           policy => policy.Requirements.Add(new
       ServiceProviderRequirement(ERole.SERVICE_PROVIDER_USER)));
       options.AddPolicy("HasAccessToSrtpRead",
           policy => policy.Requirements.Add(new
       SrtpReadRequirement()));
       … }
```

Joonis 35 – Poliitikapõhise autoriseerimiseteenuse konfiguratsioon

Kontrolleris on *SrtpReadRequirement* rakendatud läbi eelnevalt konfigureeritud „*HasAccessToSrtpRead*“ poliitika:

```
// Kontrolleri konfiguratsioon
public async Task<IActionResult> GetSrtp(Guid srtpId)
{
   // Kõigepealt päritakse ressurss andmebaasist
    var document = await
_bll.ResourceService.FindAsyncIncUserIds(srtpId);
    if (document is null)
        return NotFound();

 // Kontrollitakse õiguseid vastavalt ressursile ja poliitikale
    var authResult = await _resourceAuthService.AuthorizeAsync(User,
document, "HasAccessToSrtpRead");

    if (authResult.Succeeded)
    {
        return Ok(document.Content);
    }
    var errors = GetErrors(authResult);
    return Unauthorized(errors);

}
```

Joonis 36 – Ressursipõhise autoriseerimise poliitika rakendamine kontrolleris

5.6.3 Logimine

Logimise nõuded tulenevad rakendusele turvaraamistikust. Eeldatakse, et kõik HTTP päringud ning vastused salvestatakse kuueks kuuks andmebaasi.

Logimine on rakendatud läbi teada tuntud raamistike – *Serilog*, *Serilogi* adapteri *Elastic-Logstash-Kibana* (ELK) süsteemi ning ELK logimismootori enda abil. Valik *Serilogi* ja *ELK* süsteemi tarbeks langes seetõttu, et

1) *Serilog* on teada-tuntud lahendus .NET maailmas ning millele on mitmeid lisandusi, mida on võimalik vajadusel hõlpsasti rakendada.

2) ELK kasutamise puhul ei tule järgida kindlaks määratud logiskeemi (andmete struktuuri), on lihtsam teha tulevikus muudatusi logimise struktuuri. Võimalik salvestada nii rakenduse logisid, HTTP päringuid kui vastuseid ning dünaamiliselt määratleda struktuuri.

3) *Elasticsearch* on teada-tuntud kõrge võimekusega dokumendipõhine andmebaas ning võimaldab efektiivset otsingut tekstist. HTTP päringud ning vastused (XML/JSON) salvestatakse teksti kujul ning vajadusel on võimalik nendest lihtsasti andmeid leida.

4) *Logstash* võimaldab efektiivset andmete kogumist, transformatsiooni ja *Kibana* võimaldab andmeid lihtsasti visualiseerida.

5) ELK logimismootoril on suur ja aktiivne kasutajabaas, laialt erinevaid tööriistu ning teenuseid erinevateks juhtudeks ning toimub pidev arendustegevus.

6) ELK logimismootori võimekust on võimalik lihtsasti suurendada (*skaleerida*).

HTTP päringute logimiseks on loodud vahevara *HTTPLoggingMiddleware*, mis seob ennast ümber ülejäänud rakenduse töövoo ning salvestab nii päringu kui vastuse:

1) Ühe esimese vahevarana loetakse kontekstist HTTP päringu informatsioon.

2) Kutsutakse välja järgnev vahevara töövoos (`await next(context);`).

3) Kui programmivoog jõuab tagasi, siis loetakse HTTP kontekstist vastuse sisu.

4) Rikastatakse andmeid erinevate atribuutidega.

5) Logitakse/salvestatakse andmebaasi *Serilogiga.*

Logimise vahevara kood on leitav Lisast 9.

Veahaldus ja standardsed veateated

Teenusepakkujate vaheliste veateadete struktuur on ette määratud vastavalt teenusepakkujate vahelise rakendusliidese spetsifikatsioonile [11].

```
{
"timestamp": "2019-08-24T14:15:22Z",
"status": 0,
"error": "string",
"message": "string",
"path": "string",
"details": []
}
```

Joonis 37 – Teenusepakkujate vaheliste rakendusliideste veateate struktuur

Kõikidel veateadetel peab olema täidetud vähemalt staatus ning sõnum. Et sellist veateate struktuuri saaks kasutada ka *ASP.NET Model Validation* [33] vigade halduril tuleb luua veateate struktuuriga objektid, mis pärivad *ProblemDetails* ja *ValidationProblemDetails* klassidelt ning rakendada *ApiProblemDetailsFactory*, mille funktsionaalsus võimaldab sisseehitatud vigade halduril neid sõnumeid vastavalt vajadusele standardsel kujul luua.

Tuletatud *MyProblemDetails* ja *MyProblemValidationDetails* klassid kasutavad sarnast struktuuri varasemalt määratletud *Error (5.5.1)* klassi struktuuriga. Kõiksugune veateade, mis tagastatakse rakendusliidese poolt peab järgima ühte sama struktuuri. *ApiProblemDetailsFactory* loob vigade haldurile soovitud struktuuriga sõnumid: [43]

```
public class ApiProblemDetailsFactory : ProblemDetailsFactory
{
    public override MyProblemDetails CreateProblemDetails(HttpContext
httpContext, int? statusCode = null, string? title = null,string? type
= null, string? detail = null, string? instance = null)
    {
       …
    }

    public override MyValidationProblemDetails
CreateValidationProblemDetails(HttpContext httpContext,
        ModelStateDictionary modelStateDictionary, int? statusCode =
null, string? title = null, string? type = null,
        string? detail = null, string? instance = null)
    {
        var details = new List<ErrorDetails>();
        foreach (var kv in modelStateDictionary)
        {
            foreach (var error in kv.Value.Errors)
            {
                details.Add(new ErrorDetails()
                {
                    name = kv.Key,
                    message = error.Exception?.Message ??
error.ErrorMessage,
                    erronousValue = kv.Value.AttemptedValue
                });
            }
        }
        var myproblemDetails = new MyValidationProblemDetails()
        {
            Title = title ?? ErrorMessages.Default.ValidateTitle,
            Type = type ?? ErrorMessages.Default.ValidateType,
            Status = statusCode,
            message = title,
            path = httpContext.Request.Path,
            timestamp = DateTime.UtcNow,
            details = details
        };
        httpContext.Response.StatusCode = statusCode ??
StatusCodes.Status400BadRequest;
        return myproblemDetails;
    }
}
```

Joonis 38 – Rakendusliidese veahalduse sõnumite vabriku kood

Et rakenduses ei oleks maagilisi sõnesid (*Magic String*), siis veateated ning logimisel kasutatavad sõned on määratud kui staatilised sõned eraldi selle jaoks loodud klassis. See võimaldab vähendada võimalikke defekte tarkvaras, parandada veahaldust ning vigade avastamist (testimist) (*ErrorMessages.Default.ValidateTitle*).

5.6.4 HATEOAS rakendamine

Hypermedia As The Engine Of Application State ehk HATEOAS rakendamiseks kasutame selle jaoks eraldi loodud teeki – *RiskFirst.Hateoas*. Tegemist on teegiga, mis võimaldab lihtsasti luua ASP.NET Core rakendusliideste jaoks HATEOAS linke.

Iga sõnumi ümbrise mudeli (objekti) jaoks on konfigureeritud vastav HATEOAS poliitika, mida on võimalik hõlpsasti kas kontrolleris või mõnes teenuses rakendada läbi sinna süstitud *ILinksService* lepingut järgiva teenuse.

```
builder.Services.AddLinks(config =>
{
    // Iga ümbrise jaoks eraldi poliitika
    config.AddPolicy<SRTP_InterSP_RTP_DS02_pain_013_001_10>(policy =>
    {
        policy.RequireRoutedLink("initialSepaRequestToPayUri",
"PostPayeeSrtp").RequireRoutedLink(
            "sepaRequestToPayCancellationRequestUri",
            "PostSrtpRequestForCancellation", wrapper => new { srtpId
= wrapper.resourceId });
    });
    config.ConfigureRelTransformation(transform =>
    {
    });
});
```

Joonis 39 – HATEOAS teenuse konfiguratsioon

Selleks, et poliitika rakenduks tuleb konfigureerida kontrolleri meetodile (tegevusele) nimi ning sellele loodud poliitikas viidata.

```
[HttpPost("{srtpId}/cancellation-requests/", Name =
"PostSrtpRequestForCancellation")]
public async Task<IActionResult> PostSrtpRequestForCancellation(Guid
srtpId,
    [FromBody] SRTP_InterSP_RfC_DS11_camt_055_001_08 model) { … }
```

Joonis 40 – HATEOAS teenus rakendusliidese kontrolleris

Järgnevalt on toodud näide tulemist mille on loonud *RiskFirst.Hateoas* teenus *SRTP_InterSP_RTP_DS02_pain_013_001_10* tüüpi ümbrisobjektile. Lisatud on kaks linki – viide esialgsele SRTP loomisel kasutatud lingile ning viide maksete küsimise skeemi sõnumi tühistamispalve lingile.

```
{
    "initialSepaRequestToPayUri": {
        "Href":"https://buddy-intersp.localhost:7217/v1/payee/srtp",
        "Method":"POST"
    },
    "sepaRequestToPayCancellationRequestUri":
    {
        "Href":"https://buddy-intersp.localhost:7217/v1/sepa-request-
to-pay-requests/8b517fc1-1612-4562-924b-cb597daf4c45/cancellation-
requests",
        "Method":"POST"
    }
}
```

Joonis 41 – HATEOAS sõnumi näide

Sõnumihalduri marsruuditava sõnumi loomise koodinäites lisas nr. 7 *(55)* on täpsemalt näha kuidas HATEOAS rakendatakse *SRTP_InterSP_RTP_DS02_pain_013_001_10* tüüpi ümbrisobjektile *outputModel*:

```
await _linksService.AddLinksAsync(outputModel);
```

5.6.5 Asünkroonsed vood. Tagasikuulde aadress (*Callback URL*)

Maksete küsimise protsess võib toimuda mitmete erinevate päringute läbi ning need päringud võivad toimuda erinevatel aegadel. Näiteks võib maksete küsimise tähtaeg olla kuni 3 kuud (RTP aegumistähtaeg). Kogu selle aja jooksul võib igal hetkel maksja vastata maksete küsimise sõnumile kas negatiivse või positiivse vastusega. Seetõttu peab meie süsteem toetama asünkroonseid voogusid. Asünkroonseid voogusid on kahte tüüpi:

1) Need millele peab teenusepakkuja vastama (sissetulevad). Siin hulgas on vood, mille puhul on teenusepakkuja andnud enda tagasikuulde ressursi aadressi (*Callback URL*).

2) Need mida peab teenusepakkuja ise kliendina alustama (väljaminevad). Siin hulgas on vood, mille puhul peab teenusepakkuja ise ühendama mõne tagasikuulde ressursi aadressi poole, et uuendada RTP olekut.

Sissetulev voog

Kui vahendatakse SRTP sõnumit teisele teenusepakkujale, siis tuleb luua ka tagasikuulde ressursi aadress. Tagasikuulde aadress luuakse ASP.NET Core *UrlHelper* klassi abil, mis võimaldab vastavalt veebiteenuse, kontrolleri ning meetodi informatsioonile luua uusi

aadresse. Tagasikuulde ressurssi aadressi loomisest on toodud näide meetodi Lisas 7 *(55)* koodis millal luuakse tagasikuulde ressursi aadress teenusepakkujate vahelise kontrolleri *PostRtpInterSpCallback* ehk SRTP tagasikuulde tarbeks.

```
            outputModel.callbackUrl = result.Data.urlHelper.RouteUrl(
                routeName: "PostSrtpInterSpCallback",
                values: new { controller = "InterSp", srtpId =
outputModel.resourceId },
                protocol: "https",
                host: _serviceProviderSettings.Value.InterSpHostUri
            );
```

Joonis 42 – Tagasikuulde aadressi (Callback URL) loomise koodinäide

Sissetulevad vood on lahendatud nimelt rakendusliideste kontrollerite meetoditena. Kui mõni RTP ressurss on loodud, siis sellest jäetakse maha andmebaasi vastavad andmed ning identifikaatorid, hilisemalt on lihtne juba loodud ressurss üles leida ning tegevust jätkata sealt kus pooleli on jäänud.

Teenusepakkujate vahelise kontrolleri (InterSP) *PostSrtpInterSpCallback* meetodi kood:

```
    [HttpPost("cb/srtp/{srtpId}", Name="PostSrtpInterSpCallback")]
    [ProducesResponseType(StatusCodes.Status200OK)]
    [ProducesResponseType(StatusCodes.Status400BadRequest)]
    public async Task<IActionResult> PostSrtpCallback(Guid srtpId)
      {
            // Leia tagastatud sõnumi tüüp, võib olla mitut tüüpi!
            var typeResult = await
validationService.SrtpCallbackIdentifyType(HttpContext);
      }
```

Joonis 43 – Tagasikuulde aadressi (Callback URL) kontrolleri koodinäide

Väljaminev voog

Süsteem salvestab andmebaasi uue kirje iga SRTP ressursi kohta, samuti iga ressursi kohta sissetulevate ning väljaminevate sõnumite kohta (näiteks olekuraporti soov). Igal sellisel sõnumil on määratud identifikaatorid, tüüp ja olenevalt tüübist võib olla ka tagasikuulde ressursi aadress. Need andmed salvestatakse andmebaasi. Kui peaks tekkima väljaminevat asünkroonset voogu rakendada, siis kasutatakse selleks Sõnumihalduri teenust, et luua nendele andmete põhinev spetsiifilist tüüpi sõnum, leida andmebaasist sõnumisse määratud tagasikuulde ressursi aadress, vajadusel marsruutimise teenuse abil valideerida sihtkoht, sertifikaadid ning seejärel kasutada konnektorit, et luua uus HTTPS päring. Nende teenuste toimimine on kirjeldatud varasemates peatükkides ning eraldi siin neid välja ei hakka tooma.

5.6.6 Rakendusliidese dokumentatsioon

API dokumenteerimiseks kasutatakse *Swashbuckle* teeki, mis võimaldab *OpenAPI* spetsifikatsiooni järgi dokumenteerida rakendusliideseid. *Swashbuckle* kasutab omakorda *Swaggeri* võimalusi, mis on kogum tööriistu, mille abil *OpenAPI* spetsifikatsiooni rakendada.

Rakendusliideste dokumenteerimine toimub vastavalt rakendusliideste kontrolleri koodi ning annotatsioonide järgi. *Swashbuckle* võimaldab lihtsasti luua dokumentatsiooni erinevate rakendusliideste versioonide jaoks, eristades rakendusliideseid vaid kontrolleri annotatsioonide järgi.

Järgnevalt on määratletud teenusepakkujate vahelise rakendusliidese kontrollerile annotatsioonid, mis määravad millist sisu kontrolleris tarbitakse ja väljastatakse (XML/JSON), millist versiooni (1.0) rakendusliidesest kontroller esindab, milliste põhimõtete järgi luua erinevate versioonide jaoks marsruudid (URL), milliseid veakoode võib kontrolleri meetod tagastada.

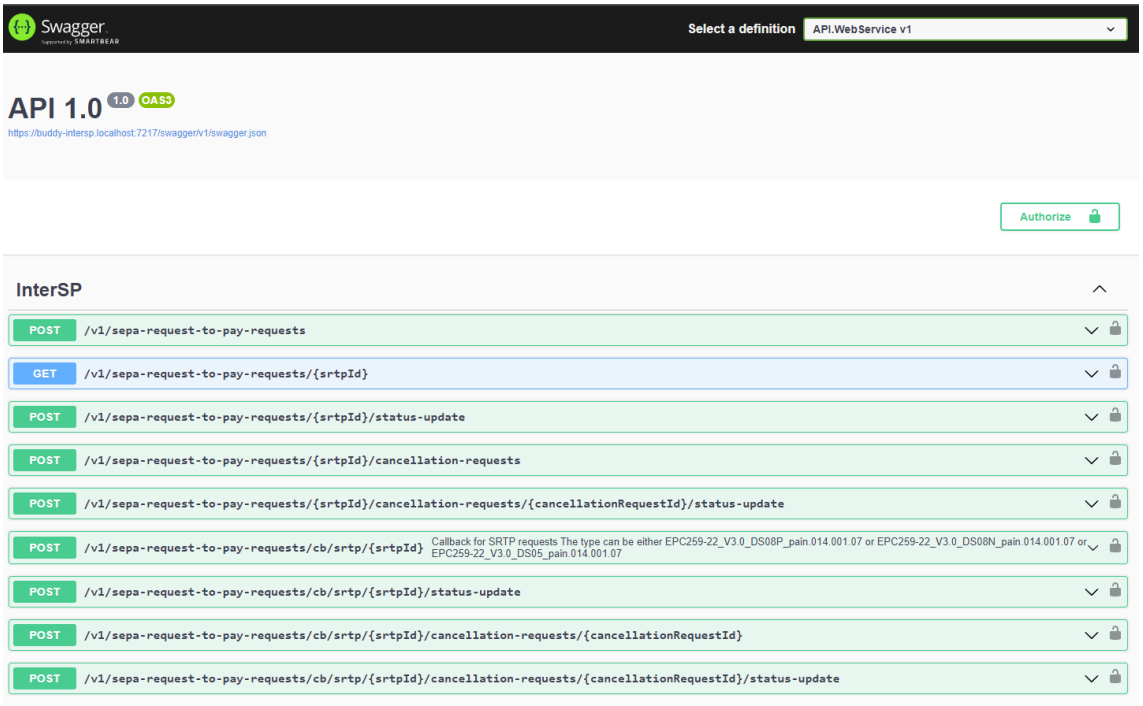
```
// Teenuspakkujate vahelise kontrolleri definitsioon
[ApiController]
[Consumes("application/json", "application/xml")]
[Produces("application/json", "application/xml")]
[ApiVersion("1.0")]
[Route("v{version:apiVersion}/sepa-request-to-pay-requests/")]
[Authorize(Policy = "RequireServiceProviderPolicy")]
[TypeFilter(typeof(ModelValidationFilter))]

public class InterSPController : ControllerBase {
…
// Kontrolleri meetod (action)
[HttpPost(Name = "PostNewSrtpMessage"),
 ProducesResponseType(StatusCodes.Status201Created),
 ProducesResponseType(StatusCodes.Status400BadRequest),
ProducesResponseType(StatusCodes.Status401Unauthorized),
 ProducesResponseType(StatusCodes.Status406NotAcceptable),
ProducesResponseType(StatusCodes.Status409Conflict),
 ProducesResponseType(StatusCodes.Status415UnsupportedMediaType),
 ProducesResponseType(StatusCodes.Status422UnprocessableEntity),
 ProducesResponseType(StatusCodes.Status429TooManyRequests),
 ServiceFilter(typeof(PostHeaderVerificationFilter))]

public async Task<IActionResult> PostSrtp([FromBody]
SRTP_InterSP_RTP_DS02_pain_013_001_10 model, [FromQuery] EErrorType
err = EErrorType.EPC)
{
…
}
```

Joonis 44 – Rakendusliidese kontrolleri koodipõhine dokumenteerimine

Swashbuckle teegi abil loodud dokumentatsioon toob välja lihtsasti, sarusaadavalt nii veakoodid, vastavalt rakendusliidese lõpp-punktile oodatavad parameetrid ja

andmemudeli (SRTP_InterSP_RTP_DS02_pain_013_001_10) kui rakendusliidese poolt pakutavad vastuse informatsiooni (veakoodid, vastuste näited nii JSON kui XML kujul jpt.). Teek võimaldab teha dokumentatsiooni lihtsasti kättesaadavaks läbi *Swagger UI* (veebipõhise rakenduse).

Järgneval joonisel on toodud teenusepakkuja *buddy-intersp.localhost* automaatselt loodud veebipõhise dokumentatsiooni näide.



Joonis 45 – Automaatselt loodud rakendusliidese dokumentatsiooni näide

Sarnaselt on loodud ka makse küsija ning maksja rakendusliideste kontrollerite dokumentatsioon.

6 Kokkuvõte

Töö eesmärk oli maksete küsimise skeemi dokumentatsiooni (reeglite raamat, rakendusjuhised, turvajuhised, rakendusliideste spetsifikatsioon), rakendusliideste ning programmeerimismeetodite parimate praktikate analüüs, analüüsi baasil luua tarkvaralahenduse kontseptsioon ning vastavalt arendada maksete küsimise skeemi teenusepakkuja tarkvara prototüüp. Tulenevalt analüüsist määrati funktsionaalsed ja mittefunktsionaalsed nõuded, selle põhjal loodi kontseptsioon. Kontseptsioonis jagati teenusepakkuja tarkvara erinevateks komponentideks ning määrati igale komponendile tööülesanded. Vastavalt kontseptsioonis kirjeldatule arendati (ja dokumenteeriti) prototüübi komponendid.

Arendatud prototüüp toetab põhilist teenusepakkujate vahelist maksete küsimise skeemi sõnumivoogu, eksisteerib rakendusliides nii makse küsija kui maksja kui teiste teenusepakkujate poole, toetab XML ja JSON vormis sõnumeid (saatmist, vastuvõtmist, valideerimist), rakendusliidesed on arendatud Richardsoni III taseme küpsustasemel, maksete küsimise sõnumeid vahendatakse mööda turvalisi kanaleid (ainult HTTPS), spetsiifilised ressursid on kättesaadavad vaid autoriseeritud osapooltele. Arendatud lahendus on turvaline (autentimine, autoriseerimine, terviklus, konfidentsiaalsus on tagatud), võimaldab auditeerida kõiki sõnumeid (salvestatakse nii päring kui vastus), on paindlik ehk toetab erinevaid rakendusvorme ning dokumenteeritud (dokumentatsioon luuakse automaatselt vastavalt koodis kirjeldatule). Prototüübi näitel on enamik funktsionaalsetest ning mittefunktsionaalsetest nõuetest on täidetud.

Prototüübi arendamise käigus selgus, et Euroopa Maksenõukogu poolt puudub detailne kirjeldus sellest, mida tegevuskava halduri (OSM) roll täpselt teeb, kuidas sellega peaks teenusepakkuja liidese looma, seda informatsiooni ka ei ole saanud.

Prototüübi arendamisele järgnevad sammud ei ole teada. Kui prototüüpi on soov juurutada siis tuleks teha eraldi projekt kus vaadata üle prototüübi sobilikkus produktsioonikeskkonda – sh. analüüsida osade teenuste veebiserverist ära viimist, näiteks autentimise ja mahupiirangute jaoks eraldi rakendusliidese lüüsi (*API Gateway*) kasutamist, mõõta ja vajadusel parandada süsteemi jõudlust ning teha turvatestimine. Sellised tegevused ei olnud lõputöö skoobis.

Kasutatud kirjandus

[1] Eesti Pank, „Eesti maksekeskkond,“ [Võrgumaterjal]. Available: https://www.eestipank.ee/maksed-arveldused/eesti-maksekeskkond.

[2] European Payments Council, „SEPA Request-to-Pay (SRTP) Scheme - A new driver for innovation in European payments,“ [Võrgumaterjal]. Available: https://www.europeanpaymentscouncil.eu/news-insights/insight/sepa-request-pay-srtp-scheme-new-driver-innovation-european-payments.

[3] „Iberpay,“ [Võrgumaterjal]. Available: https://www.iberpay.com/.

[4] European Payments Council, „SEPA Request-To-Pay Payee-Payee Service Provider Implementation Guidelines (v3.0),“ [Võrgumaterjal]. Available: https://www.europeanpaymentscouncil.eu/sites/default/files/kb/file/2023-03/EPC258-22%20Payee-Payee%27s%20RTP%20Service%20Provider%20SRTP%20IG%20V3.0_0.pdf.

[5] European Payments Council, „SEPA Request-To-Pay Inter-RTP Service Provider Implementation Guidelines (v3.0),“ [Võrgumaterjal]. Available: https://www.europeanpaymentscouncil.eu/sites/default/files/kb/file/2023-03/EPC259-22%20Inter-RTP%20Service%20Provider%20SRTP%20IG%20V3.0.pdf.

[6] European Payments Council, „SEPA Request-To-Pay Payer-Payer Service Provider Implementation Guidelines (v3.0),“ [Võrgumaterjal]. Available: https://www.europeanpaymentscouncil.eu/sites/default/files/kb/file/2023-03/EPC260-22%20Payer-Payer%27s%20RTP%20Service%20Provider%20SRTP%20IG%20V3.0.pdf.

[7] European Payments Council, „SEPA RTP Scheme Rulebook (v3.1) (EPC014-20),“ [Võrgumaterjal]. Available: https://www.europeanpaymentscouncil.eu/sites/default/files/kb/file/2023-05/EPC014-20%20v3.1%20SEPA%20RTP%20Scheme%20Rulebook.pdf.

[8] European Commission, „What is euro area?,“ [Võrgumaterjal]. Available: https://economy-finance.ec.europa.eu/euro/what-euro-area_en.

[9] European Central Bank, „Single Euro Payments Area (SEPA),“ [Võrgumaterjal]. Available: https://www.ecb.europa.eu/paym/integration/retail/sepa/html/index.en.html.

[10] European Payments Council, „https://www.europeanpaymentscouncil.eu/what-we-do/sepa-instant-credit-transfer,“ [Võrgumaterjal]. Available: https://www.europeanpaymentscouncil.eu/what-we-do/sepa-instant-credit-transfer.

[11] European Payments Council, „Default SRTP related API specifications v3.1 (EPC133-22/EPC134-22),“ [Võrgumaterjal]. Available: https://www.europeanpaymentscouncil.eu/document-library/guidance-documents/default-srtp-related-api-specifications.

[12] European Payments Council, „API Security Framework v1.0 (EPC164-22),“ [Võrgumaterjal]. Available: https://www.europeanpaymentscouncil.eu/sites/default/files/kb/file/2023-03/EPC164-22%20v1.0%20API%20Security%20Framework.pdf.

[13] I. Spectrum, „The Top Programming Languages 2023," 2023. [Võrgumaterjal]. Available: https://spectrum.ieee.org/the-top-programming-languages-2023.

[14] W. F. Benchmark, „The Benchmaker - Web frameworks Benchmark tools," [Võrgumaterjal]. Available: https://web-frameworks-benchmark.netlify.app/compare?f=aspnet-minimal-api,gin,spring,express,django.

[15] Spring VMware Tanzu, „Spring Security Framework," [Võrgumaterjal]. Available: https://spring.io/projects/spring-security.

[16] Microsoft, „ASP.NEt Core Security and Identity," [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/aspnet/core/security/anti-request-forgery.

[17] M. Fowler, „Leonard Richardson Maturity Model," 2010. [Võrgumaterjal]. Available: https://martinfowler.com/articles/richardsonMaturityModel.html.

[18] Mozilla Developer Network, „HTTP Response status codes," [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status.

[19] Microsoft, „Microsoft REST API Guidelines," [Võrgumaterjal]. Available: https://github.com/microsoft/api-guidelines/blob/vNext/Guidelines.md.

[20] OWASP, „OWASP REST Security Cheat Sheet," 2023. [Võrgumaterjal]. Available: https://cheatsheetseries.owasp.org/cheatsheets/REST_Security_Cheat_Sheet.html.

[21] C4 model, „C4 model for visualising software architecture," [Võrgumaterjal]. Available: https://c4model.com/.

[22] Microsoft, „Azure Key Vault Basic Concepts," [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/azure/key-vault/general/basic-concepts.

[23] Microsoft, „.NET ecosystem fundamentals," [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/dotnet/core/introduction.

[24] Microsoft, „Kestrel web server in ASP.NET Core," [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel.

[25] Microsoft, „Dependency injection in ASP.Net Core," [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection.

[26] Microsoft, „Entity Framework Core ORM," [Võrgumaterjal]. Available: https://github.com/dotnet/efcore.

[27] .NET Foundation, „.NET Objects AutoMapper," [Võrgumaterjal]. Available: https://github.com/AutoMapper/AutoMapper.

[28] RiskFirst.Hateoas, „HATEOAS generator for ASP.NET Core Web Api," [Võrgumaterjal]. Available: https://github.com/riskfirst/riskfirst.hateoas.

[29] Serilog Contributers, „Serilog - flexible, structured events logger," [Võrgumaterjal]. Available: https://serilog.net/.

[30] Serilog open source contributors, „Serilog.Sinks.Elasticsearch," [Võrgumaterjal]. Available: https://github.com/serilog-contrib/serilog-sinks-elasticsearch.

[31] Elastic co., „Elasticsearch, Logstash, Kibana stack," [Võrgumaterjal]. Available: https://www.elastic.co/elastic-stack/.

[32] „Swashbuckle AspNetCore Swagger and OpenAPI generator," [Võrgumaterjal]. Available: https://github.com/domaindrivendev/Swashbuckle.AspNetCore.

[33] Microsoft, „Model validation in ASP.NET Core MVC and Razor Pages," [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/aspnet/core/mvc/models/validation.

[34] M. Ganss, „XML Schema Class Generator (xscgen),“ [Võrgumaterjal]. Available: https://github.com/mganss/XmlSchemaClassGenerator.

[35] XUnit.Net, „XUnit.Net Unit Testing Tool,“ [Võrgumaterjal]. Available: https://xunit.net/.

[36] Moq, „Moq object mocking tool,“ [Võrgumaterjal]. Available: https://github.com/devlooped/moq.

[37] Github, „Github Copilot,“ [Võrgumaterjal]. Available: https://github.com/features/copilot.

[38] A. Käver, „Distributed systems ICD0021,“ [Võrgumaterjal]. Available: https://courses.taltech.akaver.com/distributed/home/.

[39] Microsoft, „Model Binding ASP.NET Core,“ [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/aspnet/core/mvc/models/model-binding.

[40] Wikipedia, „Specification Pattern,“ [Võrgumaterjal]. Available: https://en.wikipedia.org/wiki/Specification_pattern.

[41] Microsoft, „Background tasks with hosted services in ASP.NET Core,“ [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/host/hosted-services.

[42] Microsoft, „Policy-based authorization in ASP.NET Core,“ [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/aspnet/core/security/authorization/policies.

[43] Microsoft, „Implement MVC Problem Details Factory,“ [Võrgumaterjal]. Available: https://learn.microsoft.com/en-us/aspnet/core/web-api/handle-errors#implement-problemdetailsfactory.

[44] European Payments Council, „SEPA Request-To-Pay,“ [Võrgumaterjal]. Available: https://www.europeanpaymentscouncil.eu/what-we-do/other-schemes/sepa-request-pay.

[45] European Payments Council, „SEPA Request-To-Pay Referenced Technical Solution Providers (RTSP),“ [Võrgumaterjal]. Available: https://www.europeanpaymentscouncil.eu/what-we-do/other-schemes/sepa-request-pay/sepa-request-pay-referenced-technical-solution-providers.

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

Mina, Kristjan Keskküla

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Ühtse euromaksete piirkonna maksete küsimise skeemi teenusepakkuja tarkvara“, mille juhendaja on Toomas Lepikult
   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.

04.01.2024

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 – Maksete küsimise protsessi töövoo viieteistkümne põhilise sammu kirjeldus

Tabel 6 – Maksete küsimise töövoo viieteistkümne põhilise sammu kirjeldus

Samm / funktsioon	**Nimetus**	**Kirjeldus**
PS-01.01	Loo ja saada SRTP	Makse küsija loob makse küsimise sõnumi standardiseeritud vormingus. Selles sisalduvad kõik kohustuslikud elemendid ja valitud elemendid. Makse küsija saadab sõnumi makse küsija SRTP teenusepakkujale.
PS-01.02	Valideeri SRTP	Makse küsija SRTP teenusepakkuja valideerib sõnumi. Siia juurde kuulub näiteks tehnilise kontrollid, turva ja vormingu valideerimine.
PS-01.02R	Keeldumine	Kui valideerimine ebaõnnestub, siis makse küsija SRTP teenusepakkuja lükkab SRTP sõnumi tagasi, loob negatiivse olekuraporti ja saadab selle tagasi makse küsijale.
PS-01.03	Täienda ja saada SRTP	Kui valideerimine sammus PS-01.02 õnnestub siis makse küsija SRTP teenusepakkuja täiendab SRTP sõnumit elementidega, mis on vajalikud marsruutimiseks teenusepakkujate vahelises ruumis ja lisab ajatempli.
PS-01.04	Marsruutimine	SRTP sõnum marsruuditakse SRTP teenusepakkujale vastavalt marsruutimismehhanismidele.
PS-01.05	Valideeri SRTP	Maksja SRTP teenusepakkuja valideerib SRTP, sealhulgas makse identifikaatori. Siin võib juhtuda ka maksjapõhine valideerimine, et kas maksja on teenusest välja arvatud või eksisteerib mõni teine lisakonfiguratsioon
PS-01.05R	Keeldumine	Kui valideerime sammus PS-01.05 ebaõnnestub, siis maksja SRTP teenusepakkuja lükkab SRTP sõnumi tagasi, loob negatiivse olekuraporti ja

Tabel 6 – Maksete küsimise töövoo viieteistkümne põhilise sammu kirjeldus

Samm / funktsioon	**Nimetus**	**Kirjeldus**
		saadab selle tagasi makse küsijale koos vastava koodiga (AT-R004 '*Reason code for non-acceptance of the RTP')*
PS-01.05C	Valikuline kinnitus	Kui valideerimine sammus PS-01.05 õnnestub, siis maksja SRTP teenusepakkuja võib saata kinnitussõnumi kui makse küsija on seda palunud
PS-01.06	Saada SRTP esitlemiseks	Kui valideerimine sammus PS-01.05 õnnestub, siis maksja SRTP teenusepakkuja saadab SRTP sõnumi esitlemiseks maksjale
PS-01.07	SRTP hindamine	Maksja otsustab kas võtta vastu ehk aktsepteerida makse küsimine või mitte (keelduda)
PS-01.07R	Negatiivne SRTP vastus	Kui maksja otsustab keelduda makse küsimisest, siis saadetakse negatiivne sõnum tagasi maksja SRTP teenusepakkujale.
PS-01.07R	Positiivne SRTP vastus	Kui maksja otsustab aktsepteerida makse küsimise, siis saadetakse positiivne vastus tagasi maksja SRTP teenusepakkujale.
PS-01.08	Loo/uuenda olekuraportit	Maksja SRTP teenusepakkuja loob olekuraporti olenevalt maksja käest saadud vastusest ja saadab selle tagasi makse küsija SRTP teenusepakkujale. Kui peaks juhtuma, et aegumise ajaks SRTP vastust pole maksja poolt saabunud, siis SRTP pole enam ajakohane. Sellel juhul peab maksja SRTP teenusepakkuja saatma negatiivse vastuse läbi makse küsija teenusepakkuja koos sobiva koodiga ning sellest teada andma makse küsijale.
PS-01.09	Marsruutimise olekuraport	Olekuraport (positiivne või negatiivne) marsruuditakse tagasi maksjani mööda varasemat tee läbi SRTP teenusepakkujate.

Tabel 6 – Maksete küsimise töövoo viieteistkümne põhilise sammu kirjeldus

Samm / funktsioon	**Nimetus**	**Kirjeldus**
PS-01.10	Olekuraporti toimingud ja edastamine	Makse küsija SRTP teenusepakkuja kontrollib olekuraportit (positiivne või negatiivne tagasiside), teavitab makse küsijat.
PS-01.11	Makse küsija toiming olekuraportiga	Makse küsija saanud kätte olekuraporti teeb viimaseid tegevusi (näiteks uuendab SRTP kirje olekut)
PS-0.12	Olekuraporti küsimine	Teenusepakkuja võivad küsida olekuraportit (kuni SRTP aegumisajani)
PS-0.13	Marsruudi olekuraport	Olekuraporti marsruutimine teenusepakkujate vahel
PS-0.14	Valideeri olekuraporti küsimine	Kui maksja SRTP teenusepakkuja on saanud kätte olekuraporti küsimise sõnumi, siis selle sisu kontrollitakse
PS-0.14R	Olekuraporti küsimise vastus	Maksja SRTP teenusepakkuja vastab makse küsija teenusepakkujale olekuraportiga
PS-01.15	Marsruutimine olekuraporti küsimisele	Kui maksja ei ole veel jõudnud vastata esialgsele SRTP sõnumile, siis võib maksja teenusepakkuja marsruutida olekuraporti uuenduse maksjale.

Lisa 3 – RTP tühistamispalve protsessi detailne kirjeldus

Tabel 7 – RTP tühistamispalve (RfC) protsessi detailne kirjeldus

Samm / funktsioon	**Nimetus**	**Kirjeldus**
PS-02.01	Loo ja saada RfC	Makse küsija loob tühistamispalve ning saadab selle makse küsija teenusepakkujale
PS-02.02	Valideeri RfC	Makse küsija teenusepakkuja valideerib tühistamispalve sõnumi
PS-02.02R	RfC keeldumine	Makse küsija teenusepakkuja keeldub tühistamispalve sõnumist (juba tühistatud, keeldutud, aegunud või tundmatu) tühistamispalve.
PS-02.03	Täienda ja saada RfC	Kui valideerimine õnnestub, siis makse küsija teenusepakkuja täiendab tühistamispalve sõnumit (marsruutimiseks vajaliku informatsiooniga) ning saadab tühistamispalve maksja teenusepakkujale.
PS-02.04	Marsruutimine	Tühistamispalve marsruuditakse maksja teenusepakkujani
PS-02.05	Valideeri RfC	Maksja teenusepakkuja valideerib tühistamispalve sõnumi, sh. kontrollib, et kas SRTP eksisteerib, seda pole varasemalt tühistatud või keeldutud, SRTP aegumistähtaeg pole kätte jõudnud.
PS-02.05R	Negatiivne RfC vastus	Kui PS-02.05 kirjeldatud kontroll on ebaõnnestunud, siis maksja teenusepakkuja keeldub tühistamispalvest ning annab sellest teada negatiivse vastusega makse küsija teenusepakkujale.
PS-02.05R	Positiivne RfC vastus	Kui PS-02.05 kirjeldatud kontroll on õnnestunud, siis maksja teenusepakkuja annab sellest teada positiivse vastusega makse küsija teenusepakkujale.
PS-02.06	RfC tühistamine	SRTP määratakse maksja teenusepakkuja kirjetes tühistatuks

Tabel 7 – RTP tühistamispalve (RfC) protsessi detailne kirjeldus

Samm / funktsioon	**Nimetus**	**Kirjeldus**
PS-02.07	Tühistamise informatsiooni edastamine	Maksja saab tühistamise kohta informatsiooni enda teenusepakkuja käest
PS-02.08	Olekuraporti küsimine	Kuni SRTP aegumistähtajani on võimalik makse küsija teenusepakkujal küsida tühistamispalve olekuraportit.
PS-02.09	Olekuraporti marsruutimine	Marsruuditakse olekuraportit teenusepakkujate vahel.
PS-02.10	Olekuraporti küsimise valideerimine	Maksja teenusepakkuja kontrollib olekuraporti küsimise sõnumit.
PS-02.10R	Olekuraporti küsimise vastus	Maksja teenusepakkuja saadab makse küsija teenusepakkujale olekuraporti.

Lisa 4 – Funktsionaalsed ja mittefunktsionaalsed nõuded detailides

Tabel 8 – Funktsionaalsete ja mittefunktsionaalsete nõuete detailid

Nõue	Madalam detailsus	Kõrgem detailsus
Mittefunktsionaalsed nõuded detailides		
Turvalisus	Tuvastus, autentimine, autoriseerimine, andmete terviklus, konfidentsiaalsus ja käideldavus vastavalt turvaraamistikus sätestatule.	TLS kasutamine on nõutud. Algoritmide kasutamine vastavalt EPC-P556 dokumentatsioonile krüptograafiliste algoritmide kasutamisest Klientide autentimine QWAC sertifikaatidega, allkirjastamine QSealC sertifikaatidega Turvatestimine on pidev protsess ning testimiseks on eraldi keskkond Kui sertifikaadid aeguvad on vaja sellest kiiresti tegevuskava haldurile (OSM) teada anda
Käideldavus	24x7 käideldavus koos liiasuse- ja tõrkekindlusmehhanismidega.	Andmed tuleb varundada Peab eksisteerima tõrkeotsingu klaster
Võimekus	Reaalajas sõnumite vahendamise võimekus, võimekus suurema mahu sõnumite puhul tegutseda kui väiksema mahu puhul.	
Auditeerimine	Sõnumite detailne logimine ja auditijälgede hoidmine	Auditeerimise logi peab olema alati lubatud Kõik sõnumid peavad olema logitud ja hoitud vähemalt kuus kuud Auditi logi peab sisaldama logi nime, ajatemplit, välja kutsuva teenuse nime, unikaalset identifikaatorit, HTTP päringu ja vastuse täielikku sisu
Paindlikkus	Võimekus ümber lülitada erinevate rakendusvormide	

Tabel 8 – Funktsionaalsete ja mittefunktsionaalsete nõuete detailid

Nõue	Madalam detailsus	Kõrgem detailsus
	vahel nagu 3 nurga mudel (üks teenusepakkuja) ja 4 nurga mudel (kaks teenusepakkujat).	
Dokumenteeritus	Rakendusliidese toimimine peaks olema dokumenteeritud ning avalikult kättesaadav	
Funktsionaalsed nõuded detailides		
RTP teenusepakkujal peab olema liides teiste teenusepakkujate poole	Standardiseeritud vastavalt rakendusliideste juhendile	
RTP teenusepakkujal peab olema liides makse küsija või maksja poole	Võib kasutada standardiseeritud sõnumeid või mittestandardiseeritud sõnumeid	
RTP teenusepakkujal peab olema liides tegevuskava halduri (OSM) poole		
RTP teenusepakkuja peab toetama põhilisi Request-to-Pay sõnumivooge	Valideerimine vastavalt reeglitele Edastamine vastavalt nõuetele Veahaldus vastavalt nõuetele	Toetama peab kindlasti protsess samme PS-01.01, PS-01.02, PS-01.02R, PS-01.03, PS-01.04, PS-01.05, PS-01.05C, PS-01.06, PS-01.07, PS-01.07R, PS-01.08, PS-01.09, PS-01.10, PS-01.11, PS-01.12, PS-01.13, PS-01.14, PS-01.14R, PS-01.15, PS-02.01, PS-02.02, PS-02.03, PS-02.04, PS-02.05, PS-02.05R, PS-02.06, PS-02.07, PS-02.08, PS-02.09, PS-02.10, PS-02.10R
RTP teenusepakkuja peab toetama põhiliste sõnumivoogude edastamiseks vajalikke andmestruktuure (DS-) ja atribuute (AT-)	Vastavalt reeglite raamatus ning rakendusjuhistes väljatoodule	Baseerub ISO20022 finantsteenuste sõnumitel
RTP teenusepakkuja peab toetama registreerimise, aktiveerimise / desaktiveerimisse protsesse	Registreerimine - kasutaja ehk makse küsija või maksja identifitseerimine, rahapesutõkestamise meetmed ja tunne oma klienti protsessid, IBAN omaniku kontrollid.	Registreerimise väljund on SRTP aadress kujul AT-P009 + AT-N001/AT-N002.

Tabel 8 – Funktsionaalsete ja mittefunktsionaalsete nõuete detailid

Nõue	Madalam detailsus	Kõrgem detailsus
	Aktiveerimine ja desaktiveerimine – luba makse küsija ja maksja vahel edastada SRTP sõnumeid.	
RTP teenusepakkuja peab toetama RTP sõnumite saatmist läbi rakendusliides(t)e	Kohustuslikud sõnumid peavad olema toetatud rakendusliideses ISO20022 sõnumistruktuur Asünkroonsed ja sünkroonsed vood	
RTP XML sõnumeid peab saama konverteerida rakendusliideses JSON vormi	Kasutusel teenusepakkujate standardiseeritud rakendusliideses	
Iga RTP sõnumitüübi jaoks peab olema rakendusliidese puhul ressurss ja lõpp-punkti aadress		
RTP teenusepakkuja peab toetama manuste saatmist (näiteks arve)		
Oleku, veateateid, näiteks keeldumissõnumeid tuleb hallata vastavalt standardile		
RTP sõnumid peavad olema edastatud mööda turvalisi kanaleid ehk toetus krüptograafilistele meetoditele	Osaliste identifitseerimine läbi TLS raamistiku ning QWAC sertifikaatide Kõik sõnumid peavad olema krüpteeritud	
RTP ressursid peavad olema kättesaadavad vaid autoriseeritud osapooltele	HTTP päringud peavad andma sobivad vastuse koodid	Autoriseerimata päringud saavad vastuseks rakendusliidese HTTP koodi 403
Teenusepakkuja funktsionaalsus peab võimaldama RTP skeemi toimimist vastavalt Euroopa Maksenõukogu RTP reeglite raamatus, rakendusjuhistes, turvaraamistikus või teistes väljastatud lisadokumentides määratletud tingimustele		

Lisa 5 – EFBaseRepository klassi rakendamise kood

```
public class EFBaseRepository<TEntity, TKey, TDbContext> :
IBaseRepository<TEntity, TKey>
    where TEntity : class, IDomainEntityId<TKey>
    where TKey : struct, IEquatable<TKey>
    where TDbContext : DbContext
{
    protected TDbContext RepositoryDbContext;
    protected DbSet<TEntity> RepositoryDbSet;

    public EFBaseRepository(TDbContext dataContext)
    {
        RepositoryDbContext = dataContext ?? throw new
ArgumentNullException(nameof(dataContext));
        RepositoryDbSet = RepositoryDbContext.Set<TEntity>();
    }
    public virtual async Task<IEnumerable<TEntity>> AllAsync()
    {
        return await RepositoryDbSet.ToListAsync();
    }
    public TEntity? Find(TKey id)
    {
        return RepositoryDbSet.Find(id);
    }
    public virtual async Task<TEntity?> FindAsync(TKey id)
    {
        return await RepositoryDbSet.FindAsync(id);
    }
    public virtual TEntity Add(TEntity entity)
    {
        return RepositoryDbSet.Add(entity).Entity;
    }
    public virtual TEntity Update(TEntity entity)
    {
        return RepositoryDbSet.Update(entity).Entity;
    }
    public virtual TEntity Remove(TEntity entity)
    {
        return RepositoryDbSet.Remove(entity).Entity;
    }
    public virtual async Task<TEntity?> RemoveAsync(TKey id)
    {
        var entity = await FindAsync(id);
        return entity != null ? Remove(entity) : null;
    }
}
public class EFBaseRepository<TEntity, TDbContext> :
EFBaseRepository<TEntity, Guid, TDbContext>,
    IBaseRepository<TEntity>
    where TEntity : class, IDomainEntityId
    where TDbContext : DbContext
{
    public EFBaseRepository(TDbContext dataContext) :
base(dataContext)
    {    }
}
```

Lisa 6 – CertAuthService autoriseerimisteenuse kood

```
public class CertAuthService : ICertAuthService
{
    private readonly SignInManager<User> _signInManager;
    private readonly IBllUow _bllUow;
    private readonly IRepoUow _uow;
    private readonly ILogger<CertAuthService> _logger;

    public CertAuthService(SignInManager<User> signInManager, IBllUow
bllUow, IRepoUow uow, ILogger<CertAuthService> logger)
    {
        _signInManager = signInManager;
        _bllUow = bllUow;
        _uow = uow;
        _logger = logger;
    }

    public async Task<ServiceResult<ClaimsPrincipal>>
ValidateAuthorize(HttpContext context)
    {

        X509Certificate2? clientCertificate =
context.Connection.ClientCertificate;

        var result = new ServiceResult<ClaimsPrincipal>();
        // vaata kas sertifikaat eksisteerib
        if (clientCertificate is null)
        {

result.AddErrorSetFail(SRTP.Errors.CertificateAuth.NoClientCertificate
Attached, HttpStatusCode.BadRequest.ToString());
            return result;
        }
        // valideeri sertifikaadi sõrmejälg
        var validateCert = await
ValidateCertificateAsync(clientCertificate);
        if (!validateCert)
        {

result.AddErrorSetFail(SRTP.Errors.CertificateAuth.NoSuchCertificateIn
Database,
                HttpStatusCode.Unauthorized.ToString());
            return result;
        }

        // leia kasutaja õigused (claims)
        var claimsPrincipal = await
GetClaimsBasedOnDatabaseInfo(clientCertificate);
        // Set the user as authorized, if needed.
        if (claimsPrincipal is null)
        {

result.AddErrorSetFail(SRTP.Errors.CertificateAuth.NoCertificateAttach
edToUser,
                HttpStatusCode.Unauthorized.ToString());
        }
```

```
        return
ServiceResult<ClaimsPrincipal>.Success(claimsPrincipal!,
HttpStatusCode.OK);
    }
    // Meetod,mis leiab vastavalt sertifikaadile õigused andmebaasist
    public async Task<ClaimsPrincipal?>
GetClaimsBasedOnDatabaseInfo(X509Certificate2 certificate)
    {

_logger.LogInformation(SRTP.Status.CertificateAuth.FindingClaims,
certificate.Thumbprint);
        User? user = await
_uow.UserRepository.FindUserByCertificateThumbprint(certificate.Thumbp
rint);
        if (user != null)
        {
            var principal = await
_signInManager.CreateUserPrincipalAsync(user);
            return principal;
        }

_logger.LogError(SRTP.Errors.CertificateAuth.NoUserNorClaimsFound,
certificate.Thumbprint);
        return null;
    }
    public async Task<bool> ValidateCertificateAsync(X509Certificate2
clientCertificate)
    {
        var cert = await
_bllUow.CertificateService.FindCertificateByThumbprint(clientCertifica
te.Thumbprint);
        if (cert != null) // cert must exist
        {

_logger.LogInformation(SRTP.Status.CertificateAuth.FoundCertificate,
cert.PublicKey, cert.FingerPrint);
            if (cert.ServiceEnabled) // service must be enabled on it
            {
                return true;
            }

_logger.LogInformation(SRTP.Errors.CertificateAuth.CertificateDisabled
, clientCertificate.Thumbprint);
        }
        else
        {

_logger.LogInformation(SRTP.Errors.CertificateAuth.CertificateNotFound
, clientCertificate.Thumbprint);
        }
        return false;
    }
}
```

Lisa 7 – Marsruuditava sõnumi loomise teenuse kood

Järgnevast koodist on välja jäetud sammud nagu logimine ja veahaldus, et lihtsustada vähendada mahtu ning lihtsustada koodi kirjeldamist.

```
public async Task<ServiceResult<RequestMessage>>
Create(SRTP_Payee_RTP_DS01_pain_013_001_10 model,
ServiceResult<RequestMessage> result)
{
        // Loo uus sõnum
        var httpReqResult = ServiceResult<RequestMessage>.Success();
        httpReqResult.Data = result.Data;

        // Loome uue objekti
        var outputModel = new SRTP_InterSP_RTP_DS02_pain_013_001_10();
        // Loome identifikaatorid, kanname ühest mudelist teise
(Automapper)
        var oldMsgId = model.Document.CdtrPmtActvtnReq.GrpHdr.MsgId;
        var msgId = Guid.NewGuid().ShortGuidString();
        outputModel.Document.CdtrPmtActvtnReq.GrpHdr.MsgId = msgId;

        for (var i = 0; i < result.Data!.Resource.PaymentInfo.Count;
i++)
        {
            var pmtInfo = result.Data.Resource.PaymentInfo[i];
            for (var j = 0; j < pmtInfo.CreditTransferTxs?.Count; j++)
            {
                var cdtTrfTx = pmtInfo.CreditTransferTxs[j];

outputModel.Document.CdtrPmtActvtnReq.PmtInf[i].CdtTrfTx[j].PmtId.Inst
rId = cdtTrfTx.InstrId;
            }
        }
         …
        // Määra ressursi Id
        outputModel.resourceId = result.Data.Resource.Id.ToString();
        // Loo CallBack Url
        outputModel.callbackUrl = result.Data.urlHelper.RouteUrl(
            routeName: "PostSrtpInterSpCallback",
            values: new { controller = "InterSp", srtpId =
outputModel.resourceId },
            protocol: "https",
            host: _serviceProviderSettings.Value.InterSpHostUri
        );
        // Lisa HATEOAS lingid
        await _linksService.AddLinksAsync(outputModel,
"InterSPPolicy");

        // serialiseeri sisu
        string serializedContent;
        switch (result.Data.Resource.ContentType?.Code)
        {
            case nameof(EContentType.CONTENT_JSON):
                serializedContent =
_jsonXmlService.Json.Serialize(outputModel);
                break;
```

```
            case nameof(EContentType.CONTENT_XML):
                serializedContent =
_jsonXmlService.Xml.Serialize(outputModel);
                break;
            default:
                httpReqResult.AddErrorSetFail(…);
                return httpReqResult;
        }
        // Loo uus Http sõnum
        var httpReq = new HttpRequestMessage
        {
            Content = new StringContent(serializedContent),
            RequestUri = null, // TBD later during routing phase
        };
        // Määra meetod
        httpReq.Method = HttpMethod.Post;

        // Määra päised
        var idempotencyKey = Guid.NewGuid().ShortGuidString();
        var xRequestId = Guid.NewGuid().ShortGuidString();
        httpReq.Headers.Add(SRTP.Constants.HeaderTypes.IdempotencyKey,
idempotencyKey);
        httpReq.Headers.Add(SRTP.Constants.HeaderTypes.XRequestId,
xRequestId);
        httpReq.Headers.Add(SRTP.Constants.HeaderTypes.Accept,
contentString);


_logger.LogInformation(StatusMessages.CreatedHttpRoutableRequest,
serializedContent);
        httpReqResult.Data!.Request = httpReq;
        httpReqResult.Data.AddressType = EAddressType.SP_RTP_URL;

        await _messageDatabaseSaver.SaveMessageToDatabase(outputModel,
result, EMessageType.INTERSP_OUT_SRTP_REQ);

        return httpReqResult;
    }
    catch (Exception e)
    {
        _logger.LogError(e, …);
        return ServiceResult<RequestMessage>.Fail(..);
    }
}
```

Lisa 8 – Teenusepakkuja konnektori kood

```
public async Task<ServiceResult<HttpResponseMessage>>
MakeApiRequest(HttpRequestMessage request, X509Certificate2?
clientCertificate = null, int timeout = 10, string?
serverCertificateThumbprint = null)
{
    var result = ServiceResult<HttpResponseMessage>.Success();
    try
    {
        // Kui serveri näpujälg on, siis loo HTTP klient mis kontrollib
        using var handler = serverCertificateThumbprint is null
            ? new HttpClientHandler()
            :
HTTPClientHandlerService.CreateHttpClientHandlerWithServerCertificateC
heck(serverCertificateThumbprint);

         // Kui kliendi sertifikaat on määratud, siis kasutada seda
        if (clientCertificate != null)
handler.ClientCertificates.Add(clientCertificate);
        // Ühenda ning anna vastus
        using var client = new HttpClient(handler);
        client.Timeout = TimeSpan.FromSeconds(timeout);
        result.Data = await client.SendAsync(request);
        result.StatusCode = result.Data.StatusCode;
    } // Kui viga on ühendusega
    catch (HttpRequestException ex) when (ex.InnerException is
SocketException socketException)
    { // Halda vead
        result.AddErrorSetFail(
            string.Format(ErrorMessages.ApiServerRefusedConnection,
request.RequestUri, ex.Message,
                socketException.Message),
ErrorMessages.ConnectorServiceFailed, HttpStatusCode.GatewayTimeout);
    }
    // Kui viga on autentimisega
    catch (HttpRequestException ex) when (ex.InnerException is
AuthenticationException authenticationException)
    {
result.AddErrorSetFail(string.Format(ErrorMessages.InvalidCertificateE
rror, request.RequestUri, ex.Message,
                authenticationException.Message),
ErrorMessages.ConnectorServiceFailed,
            HttpStatusCode.Unauthorized);
    } // Kui on mõni teine viga
    catch (Exception ex)
    {

result.AddErrorSetFail(string.Format(ErrorMessages.ErrorMakingApiReque
st, request.RequestUri, ex.Message, ex.InnerException?.Message),
ErrorMessages.ConnectorServiceFailed,
HttpStatusCode.InternalServerError);
    }
    return result;
}
```

```
  public override async Task<ServiceResult<T1>>
MakeApiRequestResponseAsync<T1>(ServiceResult<T1> request)
{
    if (request.Data?.Request is null)
        return
ServiceResult<T1>.Fail(ErrorMessages.MissingRequestDataRoutableMessage
,ErrorMessages.ServiceProviderConnectorServiceFailed);

   if (request.Data is { UseClientCertificate: true,
ClientCertificate: null })
        return
ServiceResult<T1>.Fail(ErrorMessages.NoClientCertificateAttached,
ErrorMessages.ServiceProviderConnectorServiceFailed);

   if(request.Data is { ValidateServerCertificate: true,
ServerCertificateThumbprint: null })
        return
ServiceResult<T1>.Fail(ErrorMessages.NoServerCertificateThumbprintAtta
ched, ErrorMessages.ServiceProviderConnectorServiceFailed);

   var result = await
MakeApiRequestResponseAsync(request.Data.Request,
request.Data.ClientCertificate,
request.Data.ServerCertificateThumbprint);

   if (result.HasErrors()) return
ServiceResult<T1>.Fail(result.Errors, result.StatusCode);
    request.Data.Response = result.Data;
    return request;
}
// Overload
public override async Task MakeApiRequestAsync(HttpRequestMessage
request, X509Certificate2? certificate = null)
{
    await MakeApiRequestResponseAsync(request, certificate);
}
```

Lisa 9 – Logimise vahevara kood

```
public class HTTPLoggingMiddleware : IMiddleware
{
    private readonly ILogger<HTTPLoggingMiddleware> _logger;

    public HTTPLoggingMiddleware(ILogger<HTTPLoggingMiddleware> logger)
    {
        _logger = logger;
    }

    public async Task InvokeAsync(HttpContext context, RequestDelegate next)
    {
            try
            {
                // Loe päringu sisu ja vorminda vastavalt
                var requestBody = await context.ReadRequest();
                var request = FormatRequest(context.Request, requestBody);
                // Vaheta tavaline Stream objekt mäluga Streami vastu
                var originalResponseStream = context.Response.Body;
                var responseMemoryStream = new MemoryStream();
                context.Response.Body = responseMemoryStream;

                // Oota kuni kõik ülejäänud teenused töötlevad päringu
                await next(context);

                // Loe vastus ning töötle see sobivale kujule
                var responseBody = await context.ReadResponse();
                var response = FormatResponse(context.Response,
responseBody);
                // Taasta esialgsed Stream objektid
                await
responseMemoryStream.CopyToAsync(originalResponseStream);
                context.Response.Body = originalResponseStream;

            // Rikasta andmeid ning logi andmebaasi päring-vastus
                            using (LogContext.PushProperty("UserName",
            context.User?.Identity?.Name ?? "Not set"))
                            {
                                using (LogContext.PushProperty("UserId",
            context.User.GetUserId()))
                                {
                                    _logger.LogInformation("{@Request}",
            request);
                                    _logger.LogInformation("{@Response}",
            response);
                                }
                            }
                        }
                        catch (Exception e)
                        {
            _logger.LogError(e,ErrorMessages.HTTPLoggingMiddleware.ErrorLogg
            ingRequestResponse, e.Message);
                        }
                    }
            }
}
```