TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Maria Kaasik-Aaslav 194003IADB

Liisingusüsteemi integratsioonikihi prototüübi arendus

Bakalaureusetöö

Juhendaja: Aleksei Talisainen
Magistrikraad
Lauris Šmits
Bakalaureusekraad

Tallinn 2022

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: Maria Kaasik-Aaslav

16.05.2022

Annotatsioon

Käesoleva töö eesmärgiks on luua kõnealusele liisingusüsteemile uus integratsioonikiht. Töö teoreetilises osas antakse ülevaade integratsioonikihist tarkvara arhitektuuri kontekstis ja selle kasutusjuhtudest. Samuti kirjeldatakse kasutusel olevat integratsioonikihti, uuritakse asendusena sobivaid valmislahendusi ning seatakse nõuded loodava lahenduse jaoks. Töö praktilises osas luuakse rakenduse esialgne prototüüp, sealhulgas kirjeldatakse rakenduse olemust ja kasutatud tehnoloogiaid. Lõpus antakse hinnang loodud lahendusele ning sõnastatakse tuleviku arendustegevused.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 24 leheküljel, 7 peatükki, 7 joonist, 1 tabel.

Abstract

Prototype Development of Integration Layer for a Leasing System

The aim of this work is to create a new integration layer for given leasing system. In the theoretical part of the work, there’s is given an overview of the integration layer in the context of software architecture and its use cases. Mentioned part also includes the analysis of suitable third-party software. Subsequently the existing solution is described, based on which, also requirements for the new solution are set. In the practical part of the work, an initial prototype of the application is created, including an explanation of the application's structure and features as well as the technologies used. At the end, an evaluation is given of the created solution and activities for future development are formulated.

The thesis is in Estonian and contains 24 pages of text, 7 chapters, 7 figures, 1 table.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
CRUD	*Create-Read-Update-Delete*, andmebaasiga teostatavad baasoperatsioonid - kirje loomine, lugemine, muutmine ja kustutamine.
DRF	*Django REST Framework*, kolmanda osapoole rakendus RESTful API-de loomiseks Djangos
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastusprotokoll
JSON	*JavaScript Object Notation*, JavaScriptil põhinev andmevahetusvorming
MVC	*Model-View-Controller*, mudel-vaade-kontroller, tarkvara arhitektuurimuster, mis jagab tarkvararakenduse kolmeks omavahel seotud osaks
MVT	*Model-View-Template*, mudel-vaade-mall, sarnane MVC arhitektuurimustriga, kuid kus kontrolleri osa haldab raamistik ise.
PyPI	*Python Package Index*, Python paketiindeks on tarkvarahoidla, kus säilitatakse kogukonna hallatavate moodulite versioone
REST	*Representational state transfer*, tarkvaraarhitektuuri laad
SOAP	*Simple Object Access Protocol*, lihtne objektpöördusprotokol, arvutivõrkudes kasutatav protokoll millega veebiteenused vahetavad struktuurseid andmeid lihtne objektipöördusprotokoll
URL	*Uniform Resource Locator*, internetiaadress, mis vastab igale dokumendile või ressursile internetis
VPN	*Virtual Private Network*, virtuaalne privaatvõrk
W3C	*World Wide Web Consortium*, rahvusvaheline organisatsioon, mille eesmärgiks on arendada ja standardiseerida veebi
WSDL	*Web Services Description Language*, veebiteenuste kirjelduskeel
XML	*Extensible Markup Language*, W3C poolt väljatöödatud üldotstarbeline märgistuskeel, mille eesmärgiks on info jagamine erinevate infosüsteemide vahel
YAML	*YAML Ain't Markup Language*, andmete serialiseerimiskeel, mida tavaliselt kasutatakse konfiguratsioonifailides

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Viimastel aastatel on rakendusliidesed ehk API-d muutunud finantsteenuste sektoris parimaks tavaks kolmandate osapoolte teenuste ühendamisel finants- ja riigiasutustega andmete vahetamise eesmärgil. Selle töö kontekstis on veebiteenused vahendajaks tarbijaandmeid hoidvate asutuste nagu pangad, äriregister, krediidiühistud, ning sellele teabele juurdepääsu vajavate üksuste finantssüsteemide vahel. Nii on võimalik süsteemidel juurde pääseda finantsasutuse andmetele, ilma et nad peaksid oma taustasüsteemidega otse suhtlema. Finantsteenused kasutavad API-sid erinevate toimingute nagu krediidiotsuste automatiseerimiseks ja kiirendamiseks, tõstes seeläbi pakutava teenuse kvaliteeti.

Sageli äriprotsessides kasutatakse mitmeid eri teenuseid. Nende teenuste integreerimiseks luuakse integratsioonikiht, mis võimaldab süsteemidel ja kolmandatel pooltel vastastikku toimida kui ühendatud üksused. Integratsioonikihti on võimalik rakendad erineval moel ning võib varieeruda sõltuvalt süsteemist ja selle vajadustest.

Käesoleva lõputöö eesmärgiks on luua lahendus, mis teeks liisingusüsteemis kasutatavate veebiteenuste kasutamise ja haldamise tõhusamaks. Nimelt vajab liisingusüsteemi integratsioonikiht kaasajastamist. Peale uuemale tehnoloogiale üle viimist peaks loodav lahendus pakkuma ka korralikumat viisi kolmandate osapoolte veebiteenustega suhtlemiseks. Töö raames luuakse integratsioonikihi prototüüp.

Bakalaureuse töö on jaotatud kolme ossa: tausta ja olukorra tutvustus, analüüs ja praktiline osa. Töö esimeses osas antakse ülevaade olukorrast ning sõnastatakse lähteülesanne.

Analüüsi osas tutvustatakse veebiteenuste ja integratsioonikihi olemust ning uuritakse võimalikke lahendusvõimalusi, mis sobiks ettevõtte võimaluste ja vajadustega. Samuti pannakse paika funktsionaalsed nõuded uue lahenduse jaoks ja prioritiseeritakse need.

Töö praktilises osas kirjeldatakse tehnoloogia valikut ning valminud prototüüpi. Lisaks antakse selles osas ka hinnang loodud lahendusele ning kirjeldatakse edasised arendustegevused.

2 Probleemi püstitus

Käesolevas peatükis tutvustab autor organisatsiooni, kellele integratsioonikihti arendatakse, ning antakse ülevaade hetkeolukorrast ja projekti eesmärgist.

2.1 Taust

Ettevõtte, kus autor töötab, tegeleb liisingusüsteemi arendamisega. Liisingusüsteem võimaldab pangal jälgida liisinguandmeid, hallata päringuid ja taotlusi, salvestada finants- ja juriidilisi dokumente.

2021 aasta lõpus võeti kasutusele uuendatud liisingusüsteem, mis kasutab osi vanast liisingusüsteemist sealhulgas integratsioonikihti. Vanasse infosüsteemi sisseehitatud integratsioonikihi ülesandeks on pärida kolmandate osapoolte veebiteenuste kaudu ettevõtete ja eraisikute kohta finants- ja majandusandmeid liisingusüsteemi protsesside jaoks. Olemasolevat lahendust on aktiivselt kasutatud ja hooldatud ligikaudu 14 aastat.

Ettevõttel on plaan tulevikus täielikult uuele süsteemile üle minna mistõttu on vaja uut integratsioonikihti.

2.2 Eesmärk

Töö eesmärk on luua liisingusüsteemile uue integratsioonikihi prototüüp. Sobiva lahenduse loomiseks kogutakse lahendusele esitatavad nõuded ning prioritiseeritakse need.

Arendatava prototüübi puhul on tähtis, et see lahendaks vähemalt eksisteeriva lahendusega seotud probleemid kasutades kaasaegsemaid tehnoloogiaid.

2.3 Metoodika

Töös kirjeldatakse esmalt olemasolevat lahendust ning seejärel kogutakse nõuded. Arendusprotsess toimub inkrementaalselt, mille käigus on võimalus täiendavad

funktsionaalsused lisada või muuta, olenevalt kliendi tagasisidest juba tehtud arendustööle. Prototüübi loomisel keskendutakse eelkõige funktsionaalsetele nõuetele ja arendatakse osade kaupa, alustades kõrgema prioriteediga osadest. Nõuete prioritiseerimiseks kasutatakse MoSCoW meetodit.

Analüüsis välja toodud tehnoloogiad peavad olema kaasaegsed ning põhinema eelkõige sellel, mida ettevõte, kelle jaoks integratsioonikiht luuakse, kasutab oma igapäevatöös, kuna töö tulemust arendatakse edasi, mis eeldab arendajatelt nendes tehnoloogiates kompetentsi.

Lahenduse valmimisel sõnastatakse tulemused ja antakse hinnang loodud prototüübile.

3 Probleemi kirjeldus

Selles osas kirjeldatakse lähemalt kehtiva integratsioonikihi probleeme ja puudusi ning käsitletakse sellega seotud teemasid.

3.1 Probleemi kirjeldus

Kolmandate osapoolte arendatud rakendusliidesed (API-d) võimaldavad rakendustel juurde pääseda teiste ettevõtete väljatöötatud rakenduste pakutavatele teenustele ja andmetele. Nende rakenduste integreerimiseks luuakse integratsioonikiht, mis loob ühenduse andmete vahetamiseks, et süsteemisisesed protsessid saaksid toimida.

Integratsioonikiht rakendatakse ettevõtetes erinevalt vastavalt vajadustele. Käesolevas töös on integratsioonikiht osa liisingusüsteemist, mis põhineb monoliitsel arhitektuuril. Sellest tulenevalt on süsteemi komponendid omavahel tihedalt seotud ning seetõttu ka üksteisest väga sõltuvad. Monoliitse arhitektuuri puhul on üheks puuduseks on rakenduse hooldatavus, kus suure süsteemi puhul kujuneb muudatuste tegemine üsna keeruliseks ja nõuab terve süsteemi uuesti üles paigaldamist [1]. Need probleemid kanduvad üle ka integratsioonikihile. Kihti on aastate jooksul arendatud edasi vastavalt ärivajadusele, kus iga uue ühenduse loomiseks veebiteenusega luuakse uus kohandatud liides. Selline arendusmeetod on teinud kihi jäigaks ning edasiarenduste tegemise vaevarikkaks.

Sarnaselt süsteemiga kasutab integratsioonikiht samuti keelena Python 2, mida alates 1. jaanuarist 2020 ei toetata enam Python Software Foundationi poolt, mis muudab rakenduse turvaprobleemide suhtes haavatavaks [2]. Kuigi vana süsteemi jätkuvalt arendatakse, on ettevõttel plaanis minna täielikult üle uuele süsteemile mis kasutab Python 3-e.

Lisaks puudub kihil võimalus arendamise käigus veebiteenuste peal testida. Põhjus, miks arendamisel päris API-de peal testida ei saa on see, et kasutatavad teenused on peamiselt vahendajana finantseerimisasutuste, nagu pangad ja krediidiühistud, vahel, mis pakuvad tundlikke andmeid tarbijate kohta. Seepärast saab eksisteerivas lahenduses

hetkel silumiseks salvestada päringuid ja nende vastused, mis tähendab ikkagi seda, et ühenduste toimimist kontrollitakse alles peale tarnimist. Selline lahendus aga ei ole jätkusuutlik, kuna teeb integratsioonikihi ebausaldusväärseks ja seega liisingusüsteemi ebastabiilseks.

3.2 Kasutatud veebiteenused

Veebiteenus (ing k. *web service*) on tarkvarasüsteem, mis võimaldab rakendustel, sõltumata platvormist või programmeerimiskeelest, võrgu kaudu üksteisega suhelda. Igal teenusel on ka avalik liides, mis kirjeldab meetodid ning millised on selle on sisend- ja väljundparameetrid. [3]

Veebiteenuseid kasutakse mitmetes valdkondades erinevatel otstarvetel. Valdav osa veebiteenuseid on võimalik kasutada piiratult, ehk siis ettevõtte siseselt või eraldi juurdepääsu loal, kuid on ka teenuseid, mis on avalikkusele kasutuseks.

Laialdaselt on kasutusel kaks põhilist veebiteenuse tüüpi: „Suured“ või SOAP veebiteenused ja RESTful veebiteenused. „Suurte“ veebiteenuste all mõeldakse teenuseid, mis kasutavad XML-sõnumeid ning järgivad SOAP standardit [4]. SOAP-põhise teenuse kasutamiseks tuleb järgida veebiteenuse pakkuja poolt koostatud WSDL-dokumenti, mis kirjeldab, mis teenuseid saab kutsuda, milliseid parameetreid need ootavad ja milliseid andmestruktuure tagastatakse.

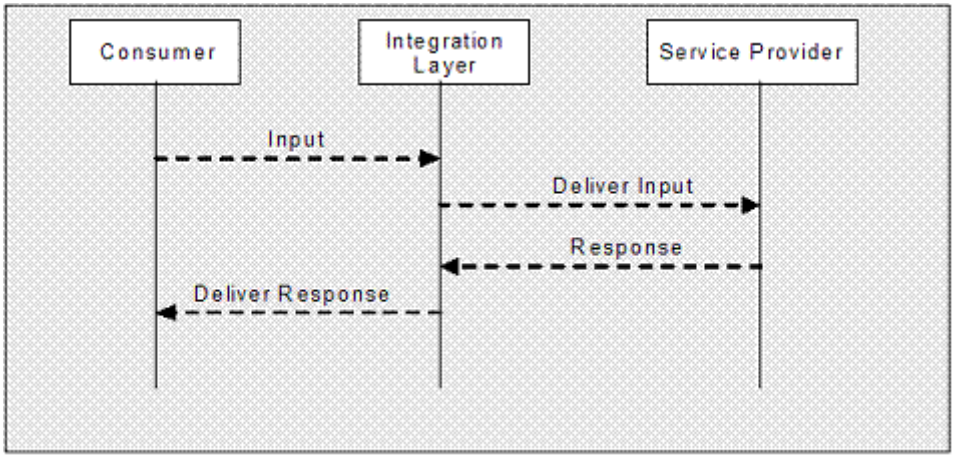
RESTful veebiteenused põhinevad aga REST tarkvara arhitektuuristiilil. Erinevalt SOAP veebiteenustest, mis põhineb funktsioonide või protseduuride kommunikatsioonil, on REST suunatud olekupõhiste ressurssidega suhtlemiseks [5]. RESTful veebiteenused toetuvad standardsetele meetoditele, mis määratletud HTTP-protokolliga: GET, POST, PUT ja DELETE. Samuti puudub RESTful veebiteenustel WSDL dokument ja sõnumiformaat ei ole piiratud XML-iga, vaid saab ka kasutada HTML, lihttekst (ing k. *plain text*), PDF, JPEG ja JSON-i põhist päringukäsitlust [4].

Nii REST kui SOAP vahetavad infot, kuid teevad seda väga erineval viisil. SOAP-i kasutatakse juhul, kui ettevõte nõuab ranget turvalisust ja selgelt määratletud reegleid, et toetada keerukamat andmevahetust ja protseduuride kutsumise võimalust. Seepärast kasutatakse sisemiste või partner API-de jaoks sageli SOAP-i. REST-i kasutatakse suhteliselt lihtsate andmete kiireks vahetamiseks. [6]

Liisingusüsteemi poolt kasutatavad API-d on peamiselt partner API-d, kelle pakkujateks on pangad või krediidiasutused.

3.3 Integratsioonikiht

Selleks, et süsteem saaks ühilduda teiste süsteemide teenuste või andmeallikatega luuakse integratsioonikiht, mis on vahendajaks teenusetarbija ja teenusepakkuja vahel (Joonis.1). Nii on tarbijad pakkujatest lahutatud, võimaldades integreerida erinevaid süsteeme.



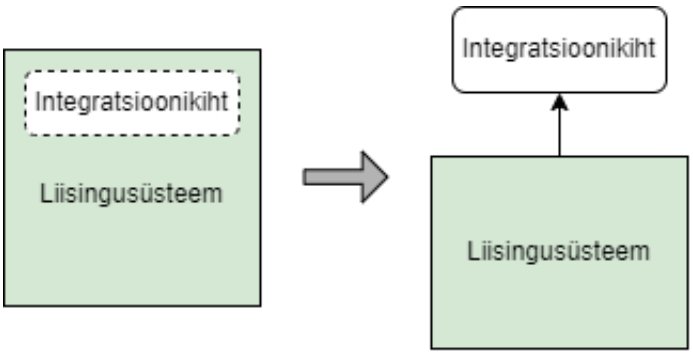
Joonis 1. Integratsioonikihi kasutus. [7]

Juurutamist saab teha kolmel erineval tasemel - rakenduse, esiliidese või andmete tasandil. Nendest kõige tavalisem on rakenduste juurutamine. See toimub liideste või teenuste kaudu ja ühendab omavahel kaks või enam süsteemi. Kõige levinumad andmete integreerimise protokollid on SOAP ja REST. [8]

Rakenduste integreerimise all võidakse mõista nii ettevõtte siseste kui väliste rakenduste integreerimist. Käesolevas lõputöös on tegemist väliste rakenduste integreerimisega olemasoleva süsteemiga. Ettevõtte väliste rakenduste puhul mõeldakse veebiteenuseid või API-sid, mille kaudu teenuseid või andmeid vahendatakse.

Liisingusüsteemis kasutusel olev integratsioonikiht toimib vahendajana süsteemi ja väliste veebiteenuste vahel, kuid kuna see toimib süsteemi siseselt, sõltub see liialt ülejäänud süsteemist. Selleks

Arvutiteaduses on otstarbe lahusus (ing k. *Separation of Concerns*) disainiprintsiip, kus tarkvarasüsteemi tükeldatakse võimalikult vähe kattuvate ülesannetega osadeks [9]. See tähendab ükski element ei tohiks jagada teiste kohustusi ega sisalda mitteseotud kohustusi. Otstarbe lahusus saavutatakse piiride kehtestamisega. Selleks võib olla mistahes loogiline või füüsiline piir, mis eraldab teatud kohustuste kogumi. [10]



Joonis 2. Integratsioonikihi eraldamine liisingusüsteemist.

Käesolevas lõputöös määratakse piir liisingusüsteemi ja integratsioonikihi vahel, eraldades kihi rakendusliideseks, mis hakkab suhtlema süsteemiga standardiseeritud sõnumite kaudu (Joonis 2). Tänu sellele on liidesel rohkem vabadust ja seda on lihtsam teistes süsteemides taaskasutada.

Selline lahendus sarnaneb mikroteenuste kontseptsioonile, kus süsteemisisesed osad on käsitletavad eraldi käitatavate teenustena, kuid mis, antud juhul, on ühe teenuse piires.

3.4 Eksisteerivad lahendused

Selles alampeatükis tutvustab autor põgusalt neid olemasolevaid lahendusi, mida uuriti sobivaks asenduseks kasutusel olevale integratsioonikihile.

Kirjeldatud integratsioonikiht sarnaneb ülesannete poolest paljuski API-haldusplatvormile. See tööriist hõlbustab ettevõttetel oma API-sid hallata, jälgida ja nende pealt raha teenida. API-haldustarkvara pakub ühtset platvormi, mis hõlmab nii ettevõtte siseste API-de haldamist kui ka kolmandate osapoolte API-de tarbimist [11].

Sellise lahenduse kasutamine, aitaks integratsioonikihis:

- normaliseerida suhtlust erinevate sõnumivormingute, nimelt XML või JSON vahel, teisendades standardvormingusse;

- API-sid katalogiseerida, mis kaudu saab parema ülevaate kasutatavatest API-dest;
- vältida punktist punkti integreerimist ja võimaldades kasutada sama kihti ka teistel süsteemidel;
- jälgida ja vajadusel piirata API-de tarbimist.

Leiti mitmeid valmistooteid, millest uuriti lähemalt kahte, mis tundusid kõige enam lubavamad. Esimesena uuriti Google'i toodet Apigee, mis on üks vanemaid ja populaarseimaid taoliste lahenduste seas. Apigee [12] sisaldab kõiki API-de haldamiseks vajaminevaid funktsionaalsusi ja veel palju lisa. Kõnealuse probleemi jaoks oleks see aga liiga suur ja kallis lahendus, mille õppimine ja seadistamine oleks ajakulukam kui uue lahenduse arendamine. Teisena uuriti samuti tuntud, kuid vabavaralist lahendust WSO2 [13]. WSO2 pakub paljuski samu funktsionaalsusi nagu Apigee, kuid võimaldab ka kohapeal majutamist. Mainitud API-haldustööriist on odavam aga seevastu raskesti kohandatav. Lisaks puudub sellel hetkel võimalus SOAP API-de testimiseks.

Valmistoodete kaalutlemisel oli põhiliseks küsimuseks selle seadistamiseks ja õppimiseks kuluv aeg ning hind. Leitud lahendused on läbivalt eelkõige mõeldud ettevõtte pakutavate API-de elutsükli haldamiseks, millest tarbitavate API-de haldamise osa on võrdlemisi väike. Seepärast otsustati algse arenduse loomisel ettevõttesisese lahenduse kasuks, mida saab luua ettevõttes kasutatavate tehnoloogiatega ning ei ole piiratud teenusepakkujate seatud piirangutest, võimaldades rohkem paindlikkust ja kohandamisvõimalusi.

4 Nõuete kogumine ja prioritiseerimine MoSCoW meetodi abil

Selleks, et uus rakendus oleks sobilik uuema liisingusüsteemi vajadustele, on tellijaga konsulteerimisel paika pandud funktsionaalsed nõuded loodava lahenduse jaoks. Üldplaanis on vaja luua uus terviklik rakendus kolmanda osapoolte API-de haldamiseks, mis:

- on eraldiseisev liisingusüsteemist, pakkudes võimalust pääseda ligi veebiteenustele tsentraliseeritud kohast;
- sisaldab kõiki CRUD-põhitoiminguid API-de lisamiseks ja haldamiseks;
- teisendab päritud andmed standardvormingusse, et pakkuda ühtset kogemust;
- võimaldab kasutada arendamiseks testservereid, kus API teenus on konfigureeritud;
- kogub statistikat teenuste kasutuse kohta;
- võimalus arendajatel saada ülevaadet olemasolevatest API-dest;
- logib rakenduse tööprotsesse, et jälgida API-de kättesaadavust;
- salvestab API-le saadetud päringud ja saadud vastused vahemällu;
- hoiab veebiteenustele ligipääsemiseks vajalikke paroole ja identsustõendeid.

Integratsioonikihi eraldamisega tekib liisingusüsteemi ja partner veebiteenuste vahele abstraktsioonikiht, mis vähendab sõltuvust kindlate kolmanda osapoolte pakkujatega.

Nii saab süsteem ühendustes esinenud rikete korral edasi toimida, ilma, et see mõjutaks ülejäänud süsteemi.. Selline lahendus aitab ka vältida kolmanda osapoolte teenuste eripärade kaasamist süsteemi loogikasse, lubades vajadusel teisele tehnoloogiale üle minna.

Lihtsamaks kasutamiseks peaks API päring olema kasutajatele ühtlustatud. See tähendab, et igale hallatavale kolmanda osapoole API-le pääseb juurde samamoodi ja ainsaks erinevuseks on kihi poolt määratud konkreetne marsruut (ing k. *route*), mis on

suunatud API-sse. Tagasi saadetud andmed teisendatakse kihis ühtsele kujule olenemata veebiteenuse tüübist, nimelt SOAP või REST.

Praegusel lahendusel puudub ühenduste testimise võimalus ehk võimalus veebiteenuste peal testida enne uue liidese ametlikult süsteemi integreerimist. Selleks on loodavas lahenduses võimalus lisaks põhiserverile ühenduda teenusepakkuja poolt pakutavale testserverile.

Lõputöö eesmärgiks on luua uus integratsioonikiht eelkõige silmas pidades liisingusüsteemi nõudeid. Samas on loodav lahendus arendatud selliselt, et seda saavad kasutada ettevõte siseselt ka teised infosüsteemid, mis kasutavad samu veebiteenuseid. Sellega kaasnev kasu seisneb selles, et edaspidi ei ole vaja teistel süsteemidel kulutada ressursse uue integratsioonikihi arendamisele ja paigaldamisele.

Selleks, et loodav prototüüp sisaldaks neid funktsionaalsusi, mida on vaja rakenduse põhiülesannete täitmiseks ja mis tooks kliendile enam väärtust, on oluline nõuded prioritiseerida. Selle põhjal jäetakse madalama prioriteediga funktsioonid käesoleva töö skoobist välja. Need funktsionaalsused plaanitakse lisada tulevikus, sest nende olemasolu ei ole kriitiline.

Prioritiseerimiseks kasutatakse MoSCoW meetodit, mis pakub lihtsat ja intuitiivset viisi ärinõuete järjestamiseks väärtuse maksimeerimise eesmärgil jagades need nelja kategooriasse [14]:

- Peab omama (ing k. *Must have*) - esmaolulised nõuded, mis tuleb täita, et lahendus toimiks.
- Peaks omama (ing k. *Should have*) - olulised nõuded, mida on vaja, kuid ilma milleta lahendus töötab.
- Võiks omada (ing k. *Could have*) - lisaväärtust andvad nõuded, mis võiksid olla lahenduses, kuid ei ole vajalikud.
- Ei pea omama (ing k. *Won't have*) - teisejärgulised nõuded, mis ei pea olema lahenduses.

Nõuded prioritiseeriti ja jaotati algse arenduse jaoks MoSCoW tabelisse (Tabel 1) samuti tellijaga konsulteerides.

Tabel 1. MoSCoW meetodi abil prioritiseeritud nõuded.

Peab omama	- Rakendus peab võimaldama kasutada kolmanda osapoolte API-de teenuseid. - Rakendus peab sisaldab kõiki CRUD-põhitoiminguid API-de lisamiseks ja haldamiseks. - Rakendus peab olema eraldiseisev liisingusüsteemist. - Rakendus peab teisendama päritud andmed standardvormingusse, et pakkuda ühtset kogemust.
Peaks omama	- Rakendus peaks võimaldama arendajatel saada ülevaadet olemasolevatest API-dest. - Rakendus peaks võimaldab kasutada arendamiseks testservereid, kus API teenus on konfigureeritud. - Rakendus peaks logima rakenduse tööprotsesse, et jälgida API-de kättesaadavust.
Võiks omada	- Rakendus võiks kogub statistikat teenuste kasutuse kohta. - Rakendus võiks salvestada API-le saadetud päringu ja vastuse info vahemällu.
Ei pea oma	- Rakendus ei pea hoidma veebiteenustele ligipääsemiseks vajalikke paroole ja identsustõendeid (token).

5 Lahenduse realiseerimine

Selles peatükis kirjeldatakse täpsemalt lõputöö fookuses olevat rakendust. Tutvustatakse tehnoloogia valikut, valminud prototüübi ülesehitust ning funktsionaalsusi.

5.1 Tehnoloogia valikud

Lõputöös käsitletav probleem keskendub põhiliselt integratsioonikihi kaasajastamisele või teisisõnu olemasoleva lahenduse ümber kirjutamisele kasutades uuemaid tehnoloogiaid. Seetõttu ei ole uue lahenduse eesmärk muuta olemasoleva integratsioonikihi loogikat, vaid kuidas seda on teostatud. Samuti lähtuti tehnoloogia valikul kokkusobivusest ettevõttes kasutatavate tehnoloogiatega ning eelistati tööriistu, mis oleks sobivad kiireks arenduseks ning lihtsasti seadistatavad.

Uue lahenduse programmeerimiskeeleks jääb Python, kuid see asendatakse uuema versiooniga, milleks on Python 3. Otsus kasutada Pythonit ei tulene sõltuvusest niivõrd liisingusüsteemist kuivõrd asjaolust, et seda hooldama hakkaval arendajal on selles keeles rohkem kogemust. Lisaks on Python kõrgetasemeline programmeerimiskeel, millel on kergesti loetav süntaks, mis teeb omakorda selle õppimise lihtsamaks [15], pakkudes rohkem funktsionaalsust vähesema koodiga. Täiendavalt on Python TIOBE Programming Community indeksi, programmeerimiskeelte populaarsuse näitaja, põhjal on üks populaarsemaid ja kiiremini kasvavaid keeli maailmas [16].

5.1.1 Raamistik

Erinevalt praegusest integratsioonilahendusest, mis töötab ainult Pythonil, kasutatakse uue lahenduse loomiseks ka veebiraamistiku. Veebiraamistik on tarkvara, mis muudab veebiarenduse kiiremaks ja lihtsamaks, pakkudes levinud mustreid usaldusväärsete, skaleeritavate ja hooldatavate veebirakenduste loomiseks [17]. Mitmed veebiraamistikud on spetsiaalselt loodud või pakuvad lisa teeke RESTful rakendusliideste loomiseks.

Loodava lahenduse jaoks uuriti kahte kasutatavamat Pythoni põhist veebiraamistikku RESTful veebiteenuste loomiseks, milleks on Python Software Foundationi ja JetBrainsi koostöös korraldatud Python Developers Survey küsitluse järgi viimase nelja aasta jooksul olnud Flask ja Django.

Flask on mikroraamistik, mis on mõeldud peamiselt väikestele rakendustele, millel on lihtsad nõuded [18]. Selle külge saab lisada erinevaid mooduleid lubades paindlikust töötada erinevate mallide ja ORM standarditega. See on kerge, kiire ning sobib hästi RESTful teenuste ehitamiseks.

Django on populaarne Pythonil põhinev raamistik, mis järgib MVC või täpsemini MVT arhitektuuri. See muudab rakenduse arendusprotsessi lihtsamaks ja sisaldab peaaegu kõike, mida on vaja täistoimiva veebirakenduse loomiseks nagu näiteks põhjalikku andmebaasi kasutamise tuge, programmi osade eraldamise ja taaskasutamise võimalust ning sisseehitatud administraatoriliidest [19]. Rakendusliideste loomiseks on Djangol olemas kolmanda osapoole moodul Django REST Framework (DRF), mida kasutatakse Django mudelite eksponeerimiseks RESTfuli liidese kaudu. See sisaldab vaateid, serialisaatoreid, valideerimist, autentimist ja palju muud nagu versioonimist ja vahemällu salvestamist, et API-sid kiiresti ja lihtsalt luua [20].

Mõlemad raamistikud keskenduvad kiiresti alustamisele ja lihtsusele, mis osas Flask võib olla isegi parem, kuid otsustati lõpuks ikka DRF kasuks. Seda põhjusel, et prototüübi loomisel otsustati jääda ettevõtte tehnoloogiapinu raamidesse, arvestades, et nii töö autoril kui ettevõtte arendajatel on selles rohkem kompetentsi, sest liisingusüsteem põhineb Djangol, ning Flaski kasutamine eeldaks uute tehnoloogiate õppimist.

5.1.2 Andmebaas

Valitud raamistik toetab ametlikult viit andmebaasi - PostgreSQL, MariaDB, MySQL, Oracle ja SQLite [21]. Nende seast otsustati esialgse arenduse jaoks kasutada SQLite andmebaasi. See on vaikimisi juba Djangosse installitud ning piisab prototüübi tegemiseks.

SQLite on aga serverivaba SQL-andmebaasimootor, kus andmeid hoitakse samas rakenduses ühes binaarfailis. Kuigi arendatav integratsioonilahendus on suunatud

peamiselt andmete edastamisele kui salvestamisele, ei ole SQLite'i kasutamine pikas perspektiivis mõistlik. Ideena hakkavad sama integratsioonikihti kasutama mitmed süsteemid, mispuhul isegi relatiivselt väikse andmemahuga, peab andmebaasisüsteem suutma tõhusalt teenindada korraga mitut päringut erinevatelt kasutajatelt.

Seepärast plaanitakse edasiseks arenduseks kasutada ühte klient-server arhitektuuriga andmebaasisüsteemi, mis on kõnealuse ettevõtte süsteemides enim kasutusel - Oracle või PostgreSQL. Mõlemad andmebaasisüsteemid toetavad enamasti samu funktsionaalsusi, kuid otsustatud sai PostgreSQL kasuks. Kuigi Oracle on võimsam, olles kiirem ja pakkudes kõrgemat turvalisuse taset [22], tooks see rakendusse liialt keerukust.

5.1.3 Vajalike teekide valik

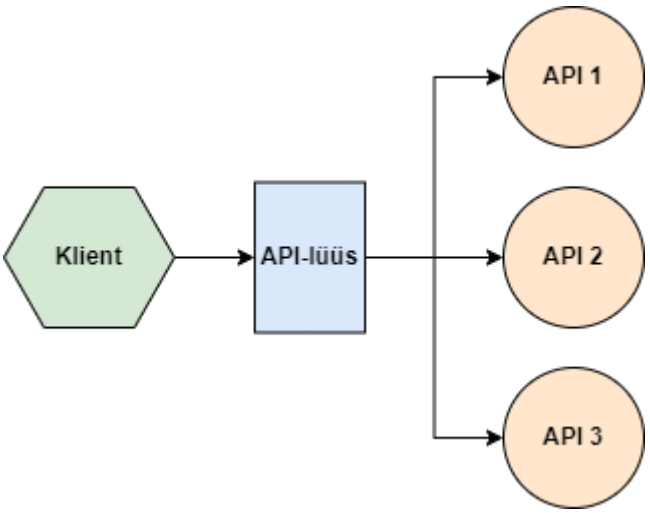
Pythoni üheks eeliseks on ka selle kolmandate osapoolte moodulite ja tugiteekide olemasolu sisaldades Pythoni paketiindeksis (PyPI) üle 300 000 paketi [23].

Pythonis kõige otsem viis SOAP veebiteenustega suhtlemiseks oleks kasutada requests teeki [24], mis võimaldab saata HTTP päringuid. Nii tuleks iga päringu jaoks koostada vastav XML-i keha vastavalt WSDL-is esitatud struktuurile. Selline lähenemine aga ei oleks tõhus, kuna nõuab igale SOAP veebiteenusele saadetava päringu jaoks eraldi XML-i kirjutamist. Siinjuures on Python moodulite seas samuti erinevaid liideseid, mis genereerivad XML-e automaatselt andes kaasa veel teisi abifunktsioone, mis tõstavad kasutusmugavusust ja korduvkasutatavust. Olemasolev integratsioonilahendus kasutab selleks SOAP-klienti Suds, mis toimib veebiteenuste puhverserverina [25]. Algsest Suds teegist ei ole aga avaldatud uut versiooni 2010. aastast. Kuigi originaalist on tehtud edasiarendusi, mida aktiivselt hooldatakse, otsustati kasutusele võtta teine kõrgetasemeline liides Zeep. Zeep on kaasaegne SOAP-klient, mis on ülesehitatud lxml ja requests moodulite põhjal [26].

Rakenduse jälgimiseks või monitoorimiseks rakendatakse rakendusse ka logimisfunktsionaalsus, mis logib rakendusse tehtud päringuid ja saadud vastusi. Selleks oli vaja seadistada ka logide haldur, mis salvestaks logisid andmebaasi. Esialgse arenduse jaoks leiti Pythoni moodul django-db-logger [27], mida oli võimalik lihtsasti rakendusse integreerida ja modifitseerida vastavalt vajadusele.

5.2 Rakendusliidese struktuur

Praegu suhtleb liisingusüsteemi integratsioonikiht erinevate välisteenuste ja andmeallikatega otse rakendusest. Uuem integratsioonilahendus on aga rakendusest eraldatud, mis otsustati realiseerida REST API-na. Loodav rakendusliides kujutab endast API-lüüsi välistele veebiteenusetele.



Joonis 3. API-lüüsi arhitektuur.

Lüüs on tarkvara, mis võimaldab tsentraliseeritud andmevahetust erinevate veebiteenuste vahel üle võrgu (Joonis 3). Läbi lüüsi vahendatakse päringuid teenuse kasutaja ehk kliendi ja pakkuja vahel. Lisaks andmete edastamisele, toimub kihis ka sõnumite teisendamine, vastavalt protokollile, sobivasse formaati. Nõnda säilitades ühtset kogemust erinevate süsteemide vahel.

Lüüsi rakendamine pakub mitmeid eeliseid, sealhulgas

- teenuste kasutajate tsentraalset haldamist igale projektile määratud API-võtme kaudu;
- juurdepääsu piiramist tundlikele kolmanda osapoole API-dele;
- ühenduse tsentraliseerimist;
- saada kokkuvõtet API igakuise kasutamise kohta.

Integratsioonikihi eraldamine teeb selle omavahel vähem sõltuvamaks liisingusüsteemist. Liidest, mida algselt kasutati ühe konkreetse süsteemi poolt, on nüüd võimalik kasutada ka teiste analoogsete süsteemide poolt. Samuti piisab vaid ühest serverist VPN ühenduse konfigureerimiseks.

Kuna ühendusi käsitletakse ühte moodi teeb see API-integratsioonide haldamise lihtsamaks. Salvestades ühenduste konfiguratsioonid tsentraliseeritud andmebaasi ja kataloogides kolmandate osapoolte API-de dokumentatsiooni saab hõlbustada nende kasutamist erinevates projektides.

Mitmete väliste API-de tarbimisel tuleb kasuks rakendada ka meetod, mis jälgib, kuidas igat API-t kasutatakse, eriti kui neid kasutavad mitmed süsteemid. Nii saab veenduda, et iga projekt rakendab samu turbestandardeid ja järgib ettevõttesiseseid eeskirju. Kolmanda osapoole API-d kasutavad samuti mitmesuguseid sõnumivorminguid, milleks on vaja rakendada lahendus nende muutmiseks sobivasse sõnumivormingusse [28]. Oluline on jälgida üldist API-de kasutust, kas need toimivad ootuspäraselt ja kui tihti neid on erineva süsteemi poolt kasutatud. Viimane tuleb eriti kasuks juhul, kui veebiteenuse eest makstakse päringute põhjal.

5.3 Rakenduse funktsionaalsus

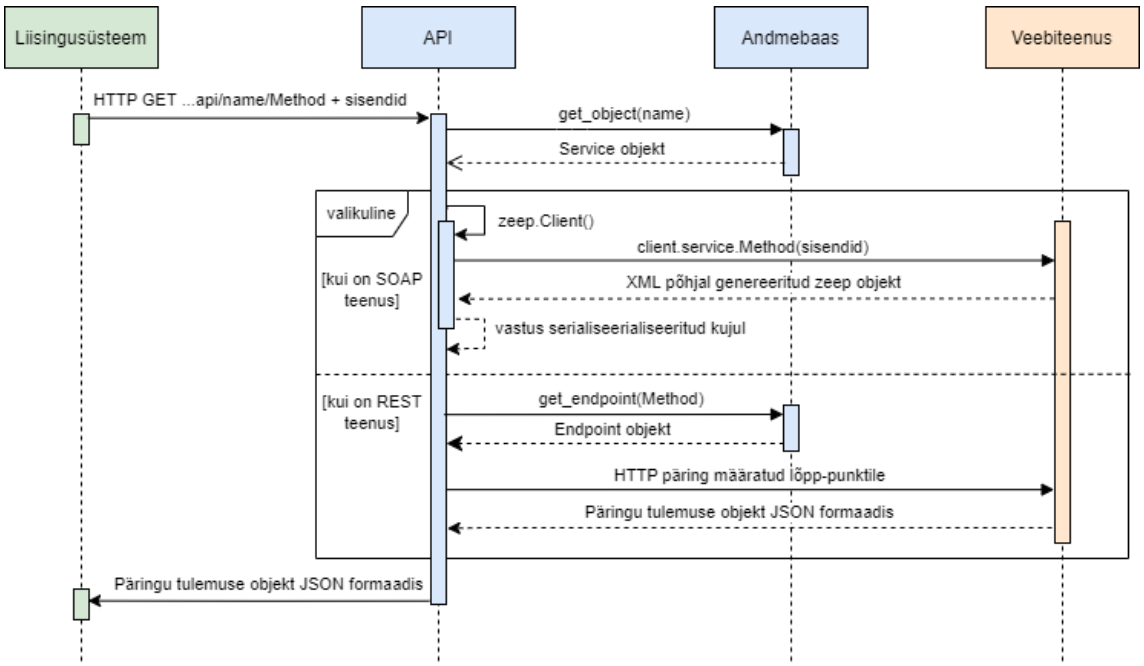
Rakendusliides saab olema ühtseks suhtluspunktiks liisingusüsteemi ja välismaailma vahel. See vastutab päringu marsruutimise, koostamise ja ka muude funktsioonide nagu näiteks autentimise eest. Kõik liisingusüsteemilt saadetud päringud lähevad esmalt lüüsi, kus suunatakse see vastava teenuse juurde. API-de jälgimiseks sai rakendatud vigade logimine ning API-dest parema ülevaate saamiseks ka nimekiri ja detailvaade.

5.3.1 Marsruutimine

Integratsioonikihi üks põhifunktsioone on päringu edastamine, mispärast oli oluline leida võimalikult lihtne viis teenuste pärimiseks ilma, et see nõuaks uute ühenduste lisamisel rakenduses lähtekoodi muutmist.

Vaja oli välja mõelda marsruutimislahendus, mis ühtlustaks API-de pärimist olenemata selle tüübist või spetsifikatsioonist. Algselt mõeldi luua lahendus, kus igale integratsioonikihti lisatud API-le antakse API-de halduris konkreetne lõpp-punkt. Selline lähenemine oleks olnud aga liiga jäik ning oleks tähendanud iga uue ühenduse lisamise korral rakendusse uue lõpp-punkti lisamist. Selle vältimiseks tuldi ideele määrata üks kindel viis kuidas teenust välja kutsuda suunates päringud ühele lõpp-punktile.

Selleks lisatakse URL-ile parameetritena päritava teenuse nimetus ja meetodi nimi, nagu “api/<nimi>/<meetod>“. Päring saadetakse HTTP GET protokolliga, kus meetodi sisendid edastatakse kehas JSON-i kujul. Sealt edasi suunatakse antud parameetrite põhjal päring teenusele, kus olenevalt veebiteenuse protokollist vormindatakse sõnum nõutud kujule, ning saadetakse edasi. (Joonis 4)



Joonis 44. Päringu edastamise jadadiagramm.

Kui REST päringute korral edastatakse päring samal kujul edasi, on SOAP päringute puhul on see veidi keerulisem. Siin lihtsustab paljuski kasutusele võetud Zeep teek. Zeep Client klass on peamine liides SOAP-serveriga suhtlemiseks, mille initsialiseerimiseks tuleb sisendiks anda WSDL-i URL (Joonis 5). Selle tulemusel luuakse ServiceProxy objekt *service* atribuutina.

```
client = Client('http://my-endpoint.com/production.svc?wsdl')
client.service.X()
```

Joonis 55. Zeep Client initsialiseerimine SOAP-serveriga suhtlemiseks.

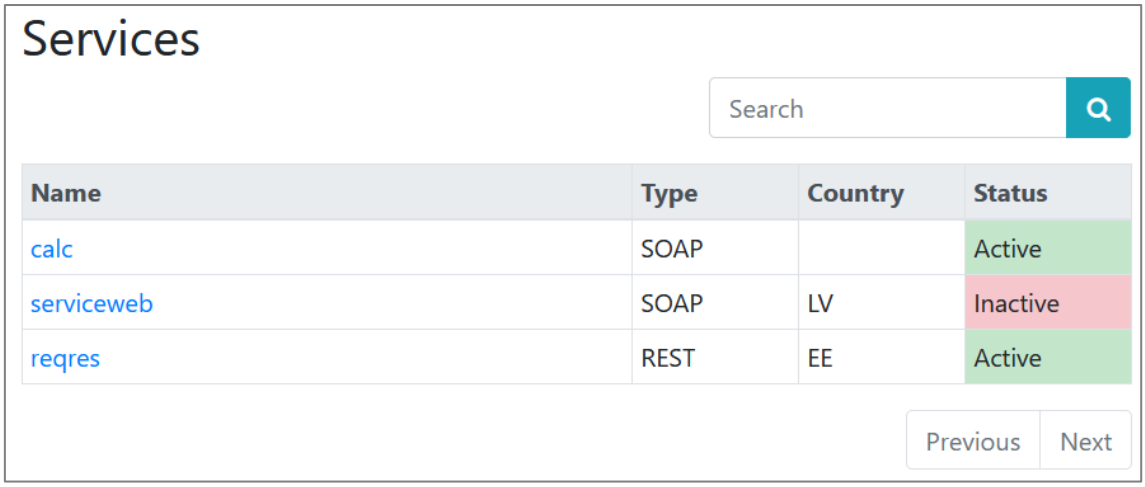
Sealt edasi saab ServiceProxy objekti kaudu kutsuda SOAP operatsioone nagu tavalisi funktsioone. ServiceProxy kontrollib, kas antud atribuudi jaoks on vastav operatsioon olemas. Kui operatsioon on olemas tagastatakse OperationProxy objekt, mis vastutab operatsiooni kutsumise eest. [26]

5.3.2 Vaated

Ühenduste konfiguratsioonide haldamiseks kasutatakse Django projektiga kaasnevat administraatori portaali. Administraatori portaalis on ülevaade seadistatud ühendustest.

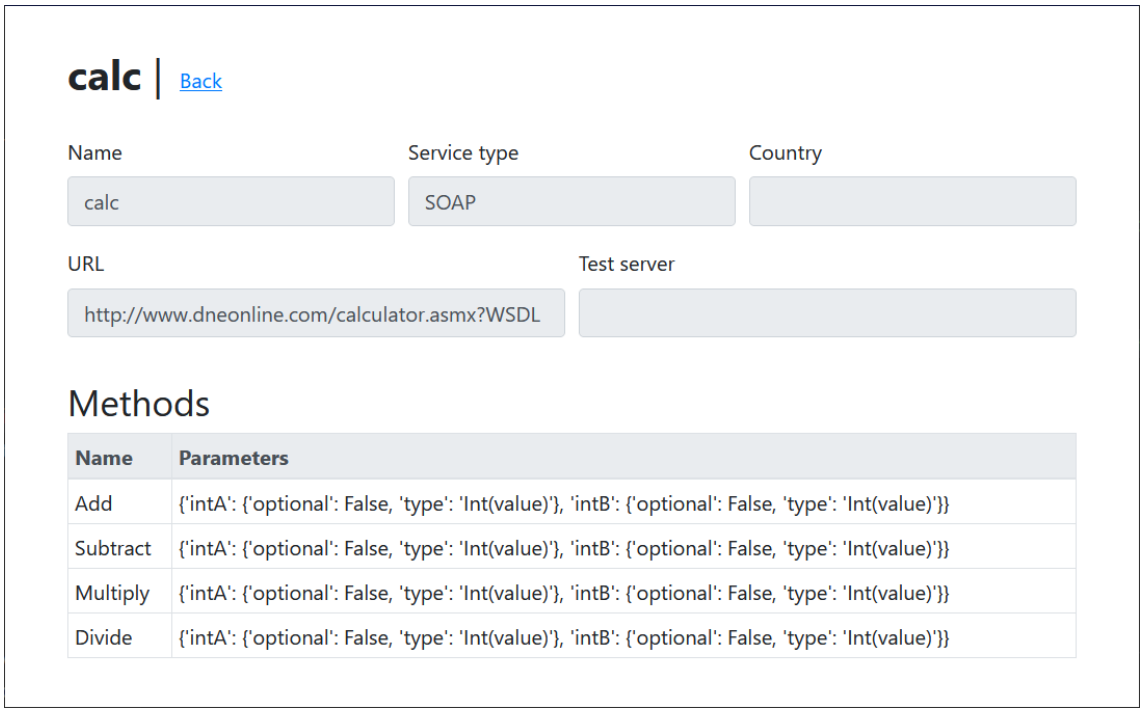
Selleks et muuta, kustutada ja vaadata mudeliga seotud tabeli kirjeid, võib kasutada Django sisseehitatud administraatori paneeli. Peale loodud mudelite saab administraatori paneelis näha ka Django poolt loodud kasutajate ja gruppide tabelit.

Rakendusliidesele loodi ka kaks vaadet saadaval olevate ühenduste nimekirjast (Joonis 6) ja nende detailvaatest (Joonis 7), mis annab arendajatele täpsemat informatsiooni veebiteenusest ja ülevaate veebiteenuse meetoditest.



Joonis 66. Veebiteenuste nimekiri vaade.

Veebiteenuste detailvaates laetakse SOAP APIde meetodid WSDL faililt koos vajaminevate parameetritega.



Joonis 77. Veebiteenuse detailvaade.

Näite jaoks on kasutatud kalkulaatorit jäljendavat SOAP veebiteenust [29].

5.3.3 Logimine

Integratsioonikiht vastutab oluliste protsesside eest, mistõttu on vaja võimalikult kiiresti tuvastada, mis on süsteemis toimunud ja pidada järge, mis API-sid kasutatakse. Logisid kasutatakse süsteemist toimivate protsesside jälgimiseks, statistika kogumiseks ja probleemide lahendamiseks [30]. Rakenduse usaldusväärsuse tagamiseks on tarvis, et logitaks kõik olulised sündmused rakenduse töös [31].

Logisid salvestatakse kohalikku andmebaasi kasutades django-db-loggerit, mille lähtekood asub versioonihaldusrakenduse GitHub salve aadressil https://github.com/CiCiUi/django-db-logger. Valitud teek pakub valmis lahendust logide haldamiseks ja visualiseerimiseks, mida oli võimalik lihtsasti rakendusse juurutada ja kohandada vastavalt vajadustele.

6 Tulemused

Järgnevas peatükis annab lõputöö autor hinnangu loodud prototüübile ja tuuakse välja tuleviku arendustegevused.

Lõputöö tulemusena valmis rakendusliides, millele jõuti arendada prioriteedilt esimesed ja sekundaarsed määratud funktsioonid. Rakenduse edasised arendused on seotud tarkvara muutmisega, funktsionaalsuse täiendamisega ja automaattestide välja töötlemisega.

6.1 Hinnang loodud integratsioonikihile

Loodud integratsioonikihti saab hinnata kaheti. Ühest küljest täidab tulemus kõiki põhilisi funktsionaalsusi, mis nõuetena määratud sai. Integratsioonikiht on eraldiseisev ja andmete vahendamine toimub määratud andmevahetusvormingu abil, mis võimaldab teistel ettevõtte sisestel süsteemidel seda kasutada. Ühenduste loomisel ei ole vaja luua eraldi liidest, vaid piisab API-ga suhtlemiseks vajalike andmete sisestamisest. Kood on struktureeritud loogilisteks osadeks ja iga funktsioon on dokumenteeritud. Nii on kood loetavam ja tagab võimaluse edasiarenduseks. Integratsioonikihil on ka kasutajaliides administraatori ja teiste arendajate jaoks, kust saab infot veebiteenuste kasutamise kohta. Administraatori vaates on võimalik näha ka logisid API-de pihta tehtud päringutest, mis on andmebaasi salvestatud.

Teisest küljest tuldi loodud prototüübi hindamisel arvamusele, et kuigi prototüüp on võrreldes olemasolevaga hallatavam ja hooldatavam, ei ole uus lahendus kõige optimaalsem. Uus integratsioonikiht vastutab mitte ainult päringute saatmise, vaid ka logi- ja dokumendihalduse eest. Lisa keerukust toob omakorda ka andmebaas, mille kasutamine on rohkem õigustatud teostatud lahenduse puhul, kuid kui ära jätta ära logide, dokumentide hoidmise, siis ainuüksi ühenduste konfiguratsioonide jaoks on seda liiga palju. Sellisel juhul oleks mõistlikum minna hoopis täiesti andmebaasi vaba lahenduse teed, kus ühenduste konfiguratsioone hoitakse rakenduses olevas YAML failis. API-sid oleks võimalik lisada nii otse faili kirjutades kui läbi kasutajaliidese,

mille kaudu saaks genereerida konfiguratsiooni esitus YAML-is. Selline lahendus oleks kergem, lihtsam ning teeks üles seadmise kiiremaks.

Mis puudutab logide haldamist saaks rakendusse integreerida logide haldamise tarkvara nagu ELK Stack või Graylog. Need tööriistad pakuvad tasuta funktsionaalsusi logide indekseerimiseks, talletamiseks ja visualiseerimiseks ning on juba ettevõtte siseselt osades projektides kasutusel. API-de kasutamiseks vajalikku dokumentatsiooni saaks hoida Confluence'is.

Sellise lahenduse peale ei tuldud varem, sest lahendusvõimaluste otsimisel võrreldi peamiselt kahte äärmist lähenemist, nimelt kas majasisene või sisseostetud lahendus. Sobivat lahendust otsides vaadati vaid terviklahendusi, mis enamasti sisaldasid palju mittevajalikke funktsionaalsusi, mille tõttu eelistati teha majasisene lahendus. Analüüsis jäi aga arvesse võtmata hübriidlahenduse võimalus, millega oleks saanud liita nii majasisese kui sisseostetud lahenduste plussid ning eemaldada nende miinused. Nende kahe meetodi kokku sobitamisel saab luua optimaalsem lahendus, kus integratsioonikiht tegeleb ainult päringute edastamisega ning ei pea sisaldama lisa funktsionaalsusi, mida oleks võimalik asendada valmis osadega.

6.2 Tuleviku arendustegevused

Edasiarendustena on järgnevalt plaanis sisse viia eelmises alampeatükis mainitud ja nendest tulenevad muudatused. See tähendab:

- kaotada andmebaas ja võtta asemele YAML failide kasutamine;
- parema kasutajakogemuse tagamiseks lisatakse kasutajaliidesesse ka funktsionaalsus mallist YAML faili genereerimiseks.;
- integreeritakse rakendusse logide haldamise moodul, mis pakub logide põhjal statistikat API-de kasutamise kohta;
- nimekiri API-dest ja nende dokumentatsioon viiakse üle Confluence'i, mis eemaldab vajaduse omada loetelu ja detail vaadet.

Nende muudatustega kaetakse ka nõuded, mis käesoleva töö skoopi ei mahtunud. Logide jälgimisele lisaks on mõeldud lisada ka moodul vigade jälgimiseks. Sentry on vigade jälgimise ja jõudluse monitoorimise platvorm, mis annab reaalajas ülevaate

rakenduse probleemidest. Lisaks saab Sentry’s konfigureerida vigade korral hoiatusi ja teateid SMS-ile või e-postile.

Veendumaks, et koodimuutus ei põhjustanud vigu, kirjutakse regressioonitestid, mis jooksutatakse automaatselt pärast igat kompilatsiooni.

Järgmise sammuna on rakenduse üles seadmine kohapealsesse serverisse, sest kõnealune ettevõte pakub ka serverite hooldus ja haldus teenuseid, ning integreerimine liisingusüsteemi koos vajaminevate ühenduste lisamisega.

7 Kokkuvõte

Lõputöö eesmärgiks oli arendada liisingusüsteemile uus integratsioonikiht, mis kasutaks kaasaegsemaid tehnoloogiad ning lihtsustaks sellega seotud hooldustöid. Lisaks võimaldab uus lahendus ka tõhusamat viisi väliste teenuste kasutamiseks.

Töö esimeses osas tutvustas autor põgusalt ettevõtte ja liisingusüsteemi tausta ning kirjeldas probleemi olemust. Töö teises osas analüüsiti tarkvarasid, mis sobiksid probleemi lahendamiseks ning leiti, et mõistlikum oleks arendada majasisene lahendus, mida saab kohandada vastavalt vajadustele. Selle järgi sõnastati ja prioritiseeriti funktsionaalsed nõuded soovitud lahendusele. Töö kolmandas osas kirjeldati tehnoloogia valikut ning loodava prototüübi ülesehitust ja funktsionaalsusi. Samas osas anti ka hinnang loodud integratsioonikihile ning sõnastati tuleviku arendustegevused.

Autori hinnangul vastab bakalaureusetöö tulemus küll püstitatud nõuetele ja ettevõtte kriteeriumitele, kuid ei ole lahendusena kõige optimaalsem. Töö käigus arendatud prototüüp lahendab probleemi liiga keerukalt, see tähendab, rakenduses on osi, mida saaks asendada paremate valmislahendustega. Edasiste arendustegevuste tulemusena, mis sai esitatud kuuendas peatükis, oleks integratsioonikiht lihtsam, kergem ja jätkusuutlikum.

Kasutatud kirjandus

[1] „Monolithic Architecture Pattern,“ [Võrgumaterjal]. Available: https://microservices.io/patterns/monolithic.html. [Kasutatud 22.aprill 2022]

[2] „Sunsetting Python 2 | Python.org,“ [Võrgumaterjal]. Available: https://www.python.org/doc/sunset-python-2/. [Kasutatud 22.aprill 2022]

[3] „Mis asi on veebiteenus? - services.krediidiinfo.ee,“ [Võrgumaterjal]. Available: http://services.krediidiinfo.ee/wiki/index.php/Mis_asi_on_veebiteenus%3F. [Kasutatud 7.mai 2022]

[4] „What Are RESTful Web Services? - The Java EE 6 Tutorial,“ [Võrgumaterjal]. Available: https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html. [Kasutatud 7.mai 2022]

[5] „What is a Web Service? | Webopedia,“ [Võrgumaterjal] Available: https://www.webopedia.com/definitions/web-services/. [Kasutatud 7.mai 2022]

[6] „What are the types of APIs and their differences? – TechTarget,“ [Võrgumaterjal] Available: https://www.techtarget.com/searchapparchitecture/tip/What-are-the-types-of-APIs-and-their-differences. [Kasutatud 8.mai 2022]

[7] „SOA Reference Architecture – Integration Layer,“ [Võrgumaterjal]. Available: https://www.opengroup.org/soa/source-book/soa_refarch/p13.htm. [Kasutatud 8.mai 2022]

[8] „IT Architecture: Integration Layer,“ [Võrgumaterjal]. Available: https://www.itarch.info/2020/05/technology-architecture-integration.html. [Kasutatud 7.mai 2022]

[9] „STATS - Standardipõhine tarkvaratehnika sõnastik,“ [Võrgumaterjal]. Available: https://stats.cyber.ee/term/3252-separation-of-concerns. [Kasutatud 10.mai 2022]

[10] „The Art of Separation of Concerns · Aspiring Craftsman,“ [Võrgumaterjal]. Available: http://aspiringcraftsman.com/2008/01/03/art-of-separation-of-concerns/. [Kasutatud 10.mai 2022]

[11] „Third-party APIs | 5 reasons API Management can manage 3rd-party APIs,“ [Võrgumaterjal]. Available: https://blog.axway.com/amplify-products/api-management/third-party-apis . [Kasutatud 9.mai 2022]

[12] „Apigee API Management | Google Cloud,“ [Võrgumaterjal]. Available: https://cloud.google.com/apigee . [Kasutatud 11.mai 2022]

[13] „WSO2 API Manager Documentation 4.1.0,“ [Võrgumaterjal]. Available: https://apim.docs.wso2.com/en/latest/ . [Kasutatud 11.mai 2022]

[14] „What is MoSCoW Prioritization? | Overview of the MoSCoW Method,“ [Võrgumaterjal]. Available: https://www.productplan.com/glossary/moscow-prioritization/. [Kasutatud 24.aprill 2022]

[15] „Python Advantages and Disadvantages - Step in the right direction - TechVidvan,“ [Võrgumaterjal]. Available: https://techvidvan.com/tutorials/python-advantages-and-disadvantages/. [Kasutatud 8.mai 2022]

[16] „index | TIOBE - The Software Quality Company,“ [Võrgumaterjal]. Available: https://www.tiobe.com/tiobe-index/. [Kasutatud 8.mai 2022]

[17] „Web Frameworks - Full Stack Python,“ [Võrgumaterjal]. Available: https://www.fullstackpython.com/web-frameworks.html. [Kasutatud 8.mai 2022]

[18] „Django vs Flask vs Pyramid: Choosing a Python Web Framework,“ [Võrgumaterjal]. Available: https://www.airpair.com/python/posts/django-flask-pyramid. [Kasutatud 8.mai 2022]

[19] „Flask vs Django: Comparing REST API Creation - ActiveState,“ [Võrgumaterjal]. Available: https://www.activestate.com/blog/flask-vs-django/. [Kasutatud Mai 2022]

[20] „Django vs. Flask in 2022: Which Framework to Choose | TestDriven.io,“ [Võrgumaterjal]. Available: https://testdriven.io/blog/django-vs-flask/. [Kasutatud 8.mai 2022]

[21] „Databases | Django documentation | Django,“ [Võrgumaterjal]. Available: https://docs.djangoproject.com/en/4.0/ref/databases/. [Kasutatud 8.mai 2022]

[22] „PostgreSQL vs Oracle: The Battle of the Titans,“ [Võrgumaterjal]. Available: https://www3.dbmaestro.com/blog/postgresql-vs-oracle-the-battle-of-the-titans. [Kasutatud 8.mai 2022]

[23] „PyPI · The Python Package Index,“ [Võrgumaterjal]. Available: https://pypi.org/. [Kasutatud 8.mai 2022]

[24] „requests · PyPI,“ [Võrgumaterjal]. Available: https://pypi.org/project/requests/. [Kasutatud 8.mai 2022]

[25] „WebServices - Python Wiki,“ [Võrgumaterjal]. Available: https://wiki.python.org/moin/WebServices#SOAP. [Kasutatud 9.mai 2022]

[26] „Zeep Documentation Release 4.1.0,“ [Võrgumaterjal]. Available: https://readthedocs.org/projects/python-zeep/downloads/pdf/master/. [Kasutatud 11.mai 2022]

[27] „django-db-logger · PyPI,“ [Võrgumaterjal]. Available: https://pypi.org/project/django-db-logger/. [Kasutatud 11.mai 2022]

[28] „Managing Third-Party API Consumption | 3Pillar Global,“ [Võrgumaterjal]. Available: https://www.3pillarglobal.com/insights/managing-third-party-api-consumption/. [Kasutatud 10.mai 2022]

[29] „Calculator Web Service,“ [Võrgumaterjal]. Available: http://www.dneonline.com/calculator.asmx. [Kasutatud 10.mai 2022]

[30] „Logid ja logimine - ITuudised,“ [Võrgumaterjal]. Available: https://www.ituudised.ee/uudised/2019/02/17/logid-ja-logimine. [Kasutatud 9.mai 2022]

[31] „Logimine - RIHA,“ [Võrgumaterjal]. Available: https://e-gov.github.io/RIHA-Index/Logimine. [Kasutatud 9.mai 2022]

[32] „Home - Django REST framework,“ [Võrgumaterjal]. Available: https://www.django-rest-framework.org/. [Kasutatud 12.mai 2022]

[33] „Python 3.10.4 Documentation,“ [Võrgumaterjal]. Available: https://docs.python.org/3/. [Kasutatud 12.mai 2022]

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

Mina, Maria Kaasik-Aaslav

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Liisingusüsteemi integratsioonikihi prototüübi arendus“, mille juhendajateks on Aleksei Talisainen ning Lauris Šmits.
   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.

16.05.2022

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.