TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Rommi Parman 184992IADB

Korteriühistu laenutaotluse otsuse automatiseerimine AS-i LHV Pank näitel

Bakalaureusetöö

Juhendaja: Meelis Antoi
MSc

Kaasjuhendaja: Artjom Pahhomov
BSc

Tallinn 2023

Autorideklaratsioon

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

Autor: Rommi Parman

15.10.2022

Annotatsioon

AS LHV Pangale on tähtis olla tehnoloogiliselt pädev ning seal kus võimalik, minimaliseerida füüsilisel kujul olevaid dokumente. Bakalaureusetöö eesmärgiks on luua korteriühistu laenutaotluse otsuse automatiseerimise lahendus, sealhulgas võimekus esitada korteriühistu laenutaotlust põhidomeenis.

Arendusprotsessi käigus luuakse korteriühistu laenutaotluse esitamise võimekus põhidomeenis ning seotakse laenu avaldus otsustuspuuga. Arendus on jagatud mitmesse ossa, mis käsitlevad *Proxy*-poolse lahenduse loomist, pärandrakenduse lahenduse loomist ning andmeaida kohandamist.

Lõputöö katab kasutatuid tehnoloogiaid ning arhitektuurilisi lähenemisi. Arendusprotsessi tulemuseks on töötav lahendus.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 31 leheküljel, 5 peatükki, 22 joonist, 3 tabelit.

Abstract

Automation of The Cooperative Loan Application Decision on Example of AS LHV PANK

It is important for AS LHV Bank to be technologically competent and if possible, minimize documents in physical form. The aim of the bachelor's thesis is to create a automation solution for the apartment association's loan application decision and ability to submit the apartment association's loan application in the main domain.

During the development process, the ability to submit a loan application of the apartment association is created in the main domain and the loan application is linked to the decision tree. The development is divided into several parts, which deal with the creation of the solution on the Proxy side, the creation of the legacy application solution, and the customization of the data warehouse.

The thesis covers the used technologies and architectural approaches. The development process results in a working solution.

The thesis is in Estonian and contains 31 pages of text, 5 chapters, 22 figures, 3 tables.

Lühendite ja mõistete sõnastik

bcp	*Bulk copy program*, hulgikopeerimisprogramm
CDS, *Credit Decision System*	Krediidiotsuse süsteem
end-to-end	Otsast otsani
Euribor	Euro üleeuroopaline pankadevaheline intressimäär
Getter-meetod	Meetod, mis tagastab välja väärtuse
HTML	Hypertext Transfer Protocol
Javax	Java standart pakett
JSON	JavaScript Object Notation
JUnit	Java testimisraamistik
Liquibase	Raamistik andmebaasi muutuste ajaloo jälgimiseks
Lombok	Java teek arenduse lihtsustamiseks
Mockito	Java testimisraamistik
null	Tühiväärtus
Numeric	Numbriväärtus
Proxy	Välisliiklust vahendav vaheserver
REST	Representational State Transfer
Setter-meetod	Meetod, mis sätestab välja väärtuse
Spring	Java rakenduse raamistik
Swagger	Tööriistade komplekt koodi dokumentatsiooni loomiseks
Thymeleaf	Java kuvamootor
WireMock	Testimisraamistik klasside imiteerimiseks

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

AS LHV Pangale on tähtis olla tehnoloogiliselt pädev ning seal kus võimalik, minimaliseerida füüsilisel kujul olevaid dokumente. Töö probleemiks on korteriühistu laen, kus panga klient peab laenu taotlemiseks täitma dokumendi ning dokumenti saab täita saates e-kirja või minnes kontorisse kohale. Lisaks tekitab füüsilise dokumendi olemasolu laenu haldamise ning taotlemise tsüklis palju muid erinevaid probleeme, milles põhilisemad on näiteks dokumentide haldus ja arhiveerimine, mis on ajakulukas ning majanduslikult ebamõistlik.

Töö eesmärgiks on luua võimalus esitada laenutaotlus ettevõtte avalikus domeenis, mille tagajärjel seotakse laenu taotlus automaatse riskihindamis- ning haldamissüsteemiga. Tulemusena säästab aega nii klient kui ettevõte, vähendades ajakulu ühele laenutaotlusele. Loodav lahendus peaks järgima ettevõttesiseseid turvanõudeid. Lahendusel peaks olema jälgitav ning vigu peaks saama hallata. Lahendus peaks olema kaetud nii ühik- kui *end-to-end*-testidega.

Bakalaureusetöö analüüsimise osas uuritakse eelnevalt täidetud dokumenti. Analüüsitakse funktsionaalseid ning mittefunktsionaalseid nõudeid. Võrreldakse olemasolevaid tehnoloogiaid ning võimalusi ning valitakse parim lahendus töö eesmärgi saavutamiseks. Analüüsi osa tulemusena valitakse tehnoloogiad, mida hakatakse kasutama praktilises osas.

Bakalaureusetöö praktilises osas proovitakse analüüsist lähtudes leitud tehnoloogiatega lahendada töö eesmärki. Vajadusel muudetakse arendusprotsessi või tehakse soovitusi protsessi muutmiseks analüüsist lähtuvalt. Praktiline osa nõuab rakendatavate tehnoloogiate täieliku tundmaõppimist.

2 Taust

Antud töö osas tutvustatakse lugejale ettevõtte ja projekti tausta, millega käesolev töö on seotud.

2.1 Korteriühistu laen

Korteriühistu laen on laenuliik kus korteriühistutele pakutakse võimalust laenata finantsvahendeid. Laenamise sihtotstarbeks võib olla näiteks renoveerimine, korrastamine või projekteerimine.

Miinimaalseks laenu summaks on 20000 eurot ning laenu maksimumperioodiks on kuni 30 aastat. Laenu väikseimaks intressiks on 3% millele lisandub kuue kuu *euribor*. Laenu saab taotleda maksehäireteta vähemalt 10 korteriga korteriühistu [1].

Laenutaotluse otsus tuleb kinnitada üldkoosolekul enamushääletusega. Laenu taotluse saab esitada korteri ühistu juhatuse liige [2].

2.2 Lahenduse majanduslik potentsiaal

Majandusliku potentsiaali saab mõõta kahes aspektis. Esimese aspektina on võimalik majanduslikku potentsiaali analüüsida ärilisest vaatenurgast, ehk kui palju kasu toob korteriühistu laen. Kahjuks ei too Finantsinspektsioon või Eesti Pank kogutud statistikas välja korteriühistu laenu mahte eraldi ärilaenudest, kuid lähtudes Eesti Vabariigi valitsuse strateegiast on potentsiaalne ärikasum väga suur [12].

Teiseks aspektiks on lahenduse automatiseerimisest tulenev ajavõit. Ajavõit tuleb nii kliendiga suhtlusel kui ka taotluse töötlemisel. Eelnevalt teostatud manuaalne protsess nagu avalduse täitmine dokumendis, e-kirja vahetamine, dokumendi sisestamine ning haldamine võttis ebamõistlikult palju aega. Autori hinnangul on ajasääst kaks kolmandiku eelnevalt kulunud ajast.

2.3 Ettevõtte taust

AS LHV Pank asutati 1999. aastal ettevõte, ning 2022. aasta seisuga on tegemist Eesti suurima kodumaise finantsasutusega. Algselt alustas ettevõte investeerimisühinguna kuid viimase 13 aasta jooksul on ettevõte laienenud nii era- kui äriklientidele suunatud turule, pakkudes erinevaid pangateenuseid ning -tooteid. 2022. aastal on ettevõttel olnud üle 530000 kliendi ning ettevõttes töötas üle 800 töötaja [3].

Lisaks pakutakse nii era- kui äriklientidele erinevaid finantstooteid nagu näiteks kodulaen, väikelaen, liising, jms. Töö autor on osa LHV finantstoodete arendusmeeskonnast ning omab pädevust uute toodete ja süsteemide väljaarendamises ning haldamises.

2.3.1 Pärandsüsteemi taust

Pärandrakendus, mida käesoleva töö autor plaanib siduda avaliku domeeniga, on CDS ehk *Credit Decision System*. CDS on pärandrakendus, mille arendamist alustati aastal 2012 ning on arendatud kasutades Java programmeerimiskeelt.

Süsteem on ettevõtte keskseks süsteemiks, kus teostatakse krediidiriskihindamist erinevatele finantstoodetele. Hindamist teostatakse nii era- kui ärikliendile. Süsteemiga on seotud otsustuspuu, mis teostab automaatset riskihindamist. Süsteemi jõuavad kõik laenutaotlused. Laenutaotluse süsteemi jõudmisel, läbitakse automaatselt otsustuspuu, pärast mida on võimalik taotlust kliendihalduril analüüsida. Pärast analüüsi on süsteemis võimalik pangaklienti otsusest teavitada ning sõlmida laenuleping.

3 Loodava lahenduse analüüs

Käesolevas peatükis analüüsitakse loodava lahenduse funktsionaalseid ning mittefunktsionaalseid nõudeid.

3.1 Nõuete määramine

Nõuete määramisel on peamiselt arvesse võetud ärilist vajadust ning ettevõtte siseseid reegleid. Lahendus peab olema kliendi jaoks piisavalt mugav ning panga turvanõuetele vastav. Nõuded, mida võiks arendada pärast praeguse skoobi valmimist, käsitletakse peatüki lõpus.

3.1.1 Funktsionaalsed nõuded

Esmased funktsionaalsed nõuded tulenevad eelnevalt täidetud PDF-taotlusest (vt Lisa 2). Põhidomeenis täidetav taotlus peab sisaldama elamu-, laenu-, taotleja informatsiooni, kavandatavaid tegevusi, finantsandmeid ja lisainfot (vt Tabel 1, lk 14-15).

Esitatud taotlus peab läbima automaatse eelhindamise, mis tähendab, et taotluse sihtpunkt peab olema seotud otsustuspuuga.

Taotluse sihtrakendus peab välja kuvama taotluse informatsiooni. Lisaks peab välja kuvama eelhindamise läbi kogutud informatsioon. Seetõttu on vajalik taotluse ning eelhindamise andmed salvestada andmebaasi.

Kliendi poolt sisestatud andmeid peab saama pärast esitamist muuta, kuid mitte kõiki andmeid. Muuta peaks saama näiteks kontaktandmeid ja laenutaotluse summat.

Laenutaotluse sihtrakenduses peab olema võimalalik teostada krediidiotsusega seotuid toiminguid nagu näiteks otsuse tegemine ning otsusest teavitamine.

Tabel 1. PDF-taotluse väljad

Infogrupp	Väli
Elamuinformatsioon	Aadress
	Ehitusaasta
	Ehitusregistri kood
	Rekonstrueerimis aasta
	Hoone konstruktsioon
	Eluruumide pindala
	Äriruumide pindala
	Korteriomandite arv
	Remondifondi makse
Laenu informatsioon	Laenu sihtotstarve
	Omafinantseeringu summa
	KredEx-i toetuse summa
	Laenusumma
	Tööde maksumus
	Laenuperioodi
	Maksepäev
	Soovitav maksepuhkus
	Tagatis
Taotleja informatsioon	Esindaja isikukood
	Esindaja isikukoodi väljastaja riik
	Esindaja eesnimi
	Esindaja perenimi
	Esindaja e-posti aadress
	Esindaja kontakt telefon
	Korteriühistu nimi
	Korteriühistu registrikood
	Korteriühistu registrikoodi väljastaja riik
	Korteriühistu juhatuse liikmed
Kavandatavad tegevused	Töö teostaja ettevõtte nimi
	Töö teostaja ettevõtteregistrikood
	Omaniku-järelevalve ettevõtte nimi
	Omaniku-järelevalve ettevõtteregistrikood
	Üldkoosolekul osalenud liikmete arv
	Üldkoosolekul laenu poolt hääletanud liikmete arv
	Uus remondifondi makse

Tabel 1. PDF-taotluse väljad.

Finantsandmed	Korteriühistu reservkapitali summa
	Korteriühistu võlakohustused, sh:
	Kohustuse liik
	Finantseerija
	Jääk
	Kuumakse
	Tähtaeg
	Korteriühistu liikmete tasumata arved
	Võlgnike arv
	Võlgnevuste summa
Lisainfo	Planeeritavate tööde kirjeldus

3.2 Teostuse valik

Funktsionaalsest nõudest tulenev nõue siduda laenutaotluse avaldus otsustuspuuga, taotluse välja kuvamine ning taotluse väljade muutmine loob soodsa olukorra siduda lahendus pärandrakendusega. Pärandrakendus on juba seotud otsustuspuuga ning pärandrakenduses on olemas palju erinevaid lahenduseks vaja minevaid komponente nii teenuse välja arendamiseks kui taotluse välja kuvamiseks. Eraldi otsustuspuu, teadete saatmise välja arendamine nõuaks väga palju aega.

3.3 Turvalisus

Funktsionaalsetes nõuetes on välja toodud, et loodav lahendus peab vastama ettevõtte turvanõuetele. Üheks nõudeks on, et ettevõtte sisene rakendus ei tohi olla nähtav väljapoole. Üheks lahenduseks on *proxy*-serveri kasutamine.

Proxy-server on olemuselt eraldiseisev rakendus, mis on nähtav väljapoole ning töötab väravana siserakenduste ja välisrakenduste vahel. Rakendus võib ka töötada filtrina, et eemal hoida kuritarvitajad, spämmijad ning botid. Lisaks loob mainitud rakendus olukorra, kus siserakenduse interneti protokolli aadress ei ole nähtav kliendipoolsele rakendusele [13].

Teiseks turvalisuse aspektiks on isikutuvastus. Selle saab lahendada otsustuspuuga, kus kasutades eelnevalt loodud sõlmi, mis kasutavad ära X-tee teenuseid isikutuvastamiseks isikukoodi või ettevõtte registrinumbri järgi.

3.4 Tehnoloogia analüüs

3.4.1 Teenusepoolse programmeerimiskeele valikud

Serveripoolse programmerimis keele valikus on mitmeid erinevaid võimalusi. Tuntumateks lahendusteks on [8]:

- Ruby – avaliku lähtekoodiga programmeerimiskeel. Programmeerimiskeele loomisel keskenduti koodi lihtsusele ning produktiivsusele. Populaarne kasutada koos raamistikuga Ruby On Rails [5].

- PHP – laialdaselt kasutatav avatud lähtekoodiga üld-otstarbeline dünaamiline keel, mis sobib hästi veebiarenduseks. Kood genereeritakse serveri poolel ning seejärel serveeritakse HTML-i kujul kliendile [4].

- Javascript – dünaamiline, mitmekesine objektorienteeritud programmeerimiskeel mida on võimalik kasutada nii serveri- kui kliendipoolseks lahenduseks [6].

- C# – objektorienteeritud programmeerimiskeel mida kasutatakse peamiselt koos .NET raamistikuga. Väga paindlik, võimaldab erineva keerukusega rakendusi, suure jõudlus talumisega. Miinusena jookseb ainult platvormidel mis toetavad .NET keskkonda [7].

- Java – objektorienteeritud programmeerimiskeel. Interpreeritakse Java Virtual Machine'l mis võimaldab kasutada palju erinevaid platvorme [32]. Hästi dokumenteeritud kuna on üks populaarseimaid programmeerimiskeeli.

Järgnevalt tuuakse välja keelte võrdlus. Võrdluspunktideks on autori kogemus programmeerimiskeelega ning programmeerimiskeele õppimiskeerukus (vt Tabel 2).

Tabel 2. Programmeerimiskeelte võrdlus.

Keel	**Kogemus**	**Õppimiskeerukus**
Ruby	Puudub	Madal
PHP	Rahuldav	Madal
Javascript	Rahuldav	Madal
C#	Hea	Keskmine
Java	Väga hea	Keskmine

Lähtudes sellest, et uue programmeerimiskeele õppimine on väga aeganõudev ning lahenduse loomiseks mõeldud aeg on piiratud, ei näe autor võimalust uue keele selgeksõppimiseks. Seetõttu jääb sõelale antud tabelist Java või C#. Keerukus on mõlemal keelel keskmine kuid kuna autor töötab igapäeva elus Java programmeerimis keelega ning neid programmeerimis keeli võib pidada sarnaseks, ei tekita õppimiskeerukus raskusi.

Java eeliseks on lisaks ka see, et pärandrakendus on arendatud Java keeles. Seetõttu oleks loodava lahenduse sidumine palju kergem.

3.4.2 Teenusepoolse raamistiku valik

Teenusepoolse ehk *backend* tehnoloogia valikus on palju erinevaid valikuid.

Neist populaarsemad on:

- LAMP pinu – pinu sisaldab Linuxil põhinevat operatsioonisüsteemi, Apache HTTP serverit, MySQLI'i ning PHP programmeerimiskeelt. Antud pinu on väga populaarne ning põhineb vabavaral.
- MEAN pinu – pinu on sarnane LAMP pinule, kuna sisaldab vabavaralisi komponente. Traditsioonilise relatsioonilise andmebaasi asemel on kasutusel mitterelatsiooniline andmebaas. Sisaldab Express.js-i, Angular-i ja Node.js-i.

Pinu suureks plussiks on, et nii kliendipoolne kui ka teenuse poolne kood on kirjutatud Javascriptis.

- RORM pinu – sisaldab Ruby On Rails teeki mis põhineb Ruby programmeerimis keelel. Andmebaasina on populaarne valik MySQL või MongoDB.
- .NET pinu – Põhineb .NET raamistikul ning C# keelel. Pakub väga head skaleerimis võimalust ning palju tööriistu rakenduse arendamiseks.
- Java pinu – Põhineb Java programmeerimiskeelel ning põhiliselt on kasutusel raamistikud, nt Spring või Spring Boot. Serveri poolel on põhiliselt kasutusel WildFly (endine JBoss) või Apache Tomcat. Andmebaasi või kliendipoolse rakenduse valik ei ole oluline kuna sobib enamus.

Tuntumad teenusepoolsed raamistikud on:

- .NET – suletud lähtekoodiga raamistik. Raamistikuga kaasneb ka server.
- Flask – Kiiresti rakendatav ning populaarne valik kui aeg on piiratud.
- Express.js – Sobib hästi lihtsamate veebirakenduste loomiseks. Kiiresti rakendatav.
- Node.js – Keskkond mis lubab jooksutada Javascripti. Skaleerub hästi ning on võimalik luua ka keerulisemaid rakendusi.
- Ruby On Rails – Ressursinõudlik, kuid kiiresti rakendatav ja skaleeritav.
- Spring – Avatud lähtekoodiga Javal põhinev raamistik mis pakub erinevaid lahendusi veebirakenduse arendamiseks. Kõrval kasutatakse Spring Booti mis võimaldab Springi konfiguratsiooniga projekti luua.

Järgnevalt võrreldakse erinevaid populaarseid raamistike:

Tabel 3. Teenusepoolsete raamistike võrdlus.

	Paindlikus	**Kogemus**	**Turvalisus**
Django	Paindlik	Puudub	Piisav
.NET	Paindlik	Hea kogemus	Väga hea
Flask	Paindlik	Puudub	Vajab täiendavaid lahendusi [9]
Express.js	Vähem paindlik	Mõningane kogemus	Omab puudujääke [10]
Node.js	Paindlik	Mõningane kogemus	Omab puudujääke [10]
Ruby On Rails	Paindlik	Puudub	Vajab täiendavaid lahendusi [11]
Spring	Paindlik	Väga hea kogemus	Väga hea

Arvestades tabelit ning eelmises peatükis programmeerimiskeele põhjendusi oleks sobilik valik Spring. Lisaks on Java ka käesoleva töö autori eelistatud keeleks. Lisaks kasutab pärandrakendus Spring ja Spring Boot'i kooslust.

3.4.3 Kliendipoolsed tehnoloogiad pärandrakenduses

Tulenevalt ettevõtte nõuetest ei ole autoril võimalust välja arendada põhidomeeni kuva. Autori poolt kohandatakse pärandrakenduses olevaid elemente.

Pärandrakendus kasutab MVC disainimustrit koos *Thymeleaf* mallimustrit.

Thymeleaf on *Spring*-ga hõlpsasti töötav lahendus, kus on võimalik luua staatilisi *HTML* malle. Küll aga ei võimalda *Thymeleaf* keerulisi äriloogilisi lahendusi. Seetõttu tegi autor ettepaneku liikuda tulevikus modernsema lahenduse peale nagu näiteks React või Angular.

Sõelale jäi Angular, kuna Angular on ettevõtte siseselt juba suuresti kasutusel ning loodud juba palju erinevaid komponente, mida on võimalik taaskasutada.

3.5 Andmebaasi valik

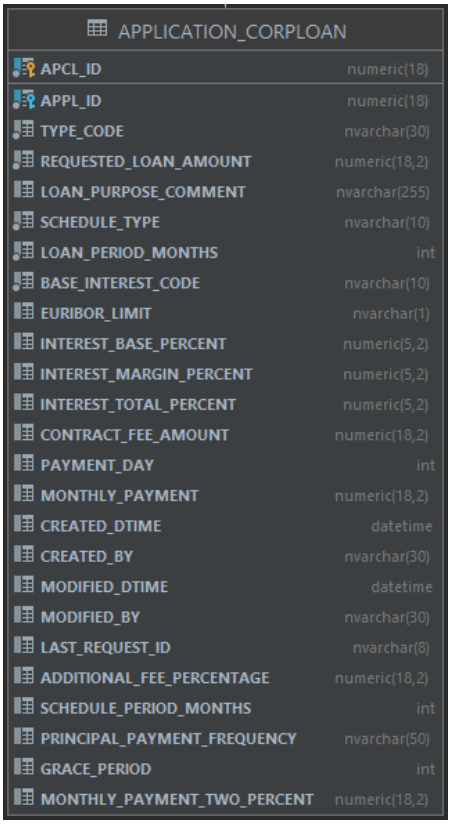
Eelnevatest peatükkidest väljatulnud nõuete ja analüüsi tulemusena kasutatakse ettevõtte poolt juba seadistatud andmebaasi. Andmebaasi süsteemiks on Microsofti SQL Server millega suheldakse Transact-SQL keeles.

3.5.1 Andmebaasi tabelite disain

Kuna kasutatakse eelnevalt ettevõttes loodud andmebaasi saame ka taaskasutada olemasolevat ärilaenu andmebaasi andmemudelit ning tänu sellele ka mitmeid kasutuses olevaid tabeleid.

3.5.1.1 Tabel APPLICATION_CORPLOAN

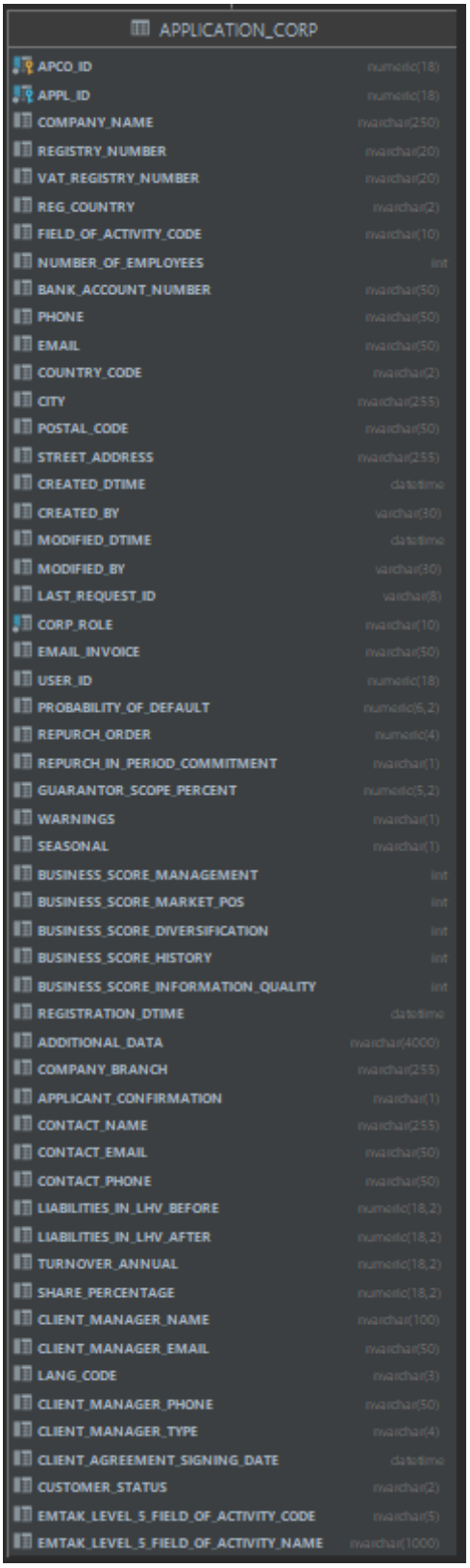
Tabeli APPLICATION_CORPLOAN alla saame salvestada funktsionaalsetes nõuetes välja tulnud laenuinformatsiooni (vt Joonis 1).



Joonis 1. Laenuandmete tabel

3.5.1.2 Tabel APPLICATION_CORP

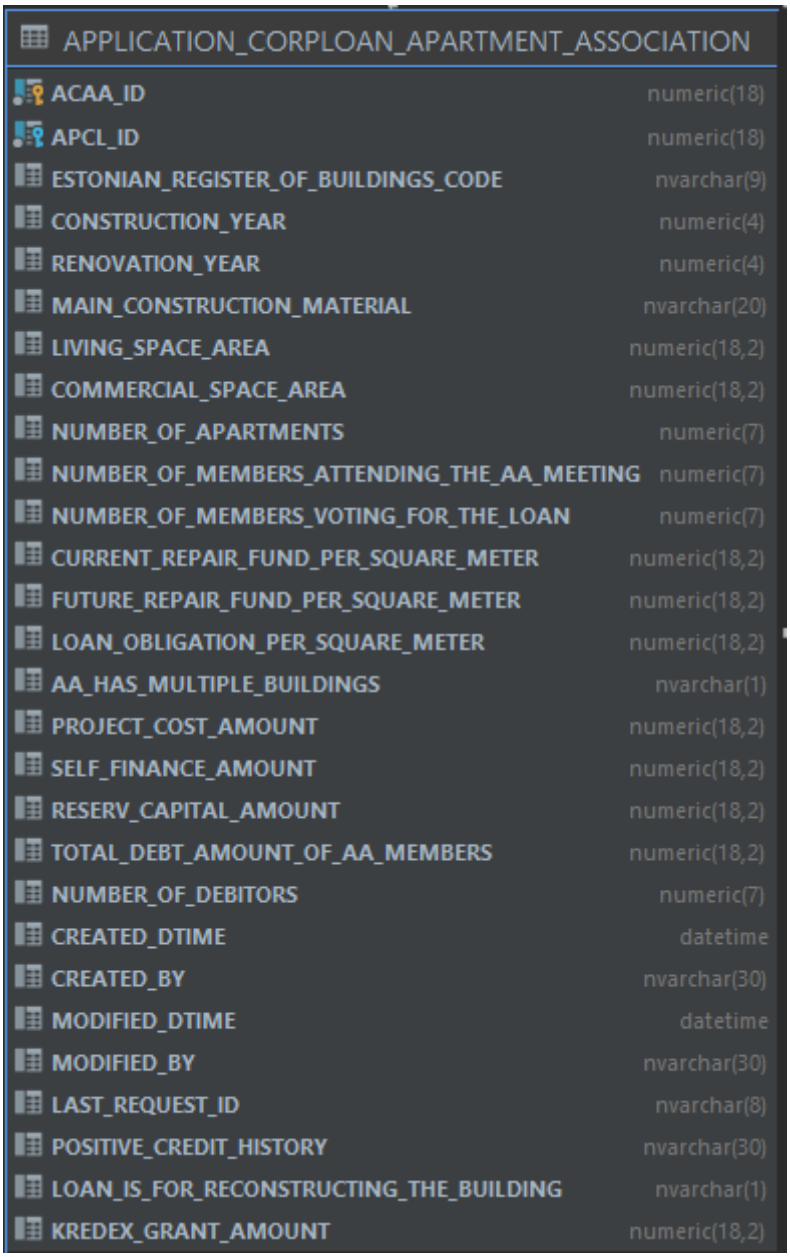
Tabel APPLICATION_CORP alla same salvestada taotleja andmed.



Joonis 2. Taotleja informatsiooni tabel

3.5.1.3 Tabel APPLICATION_CORP_APARTMENT_ASSOCIATION

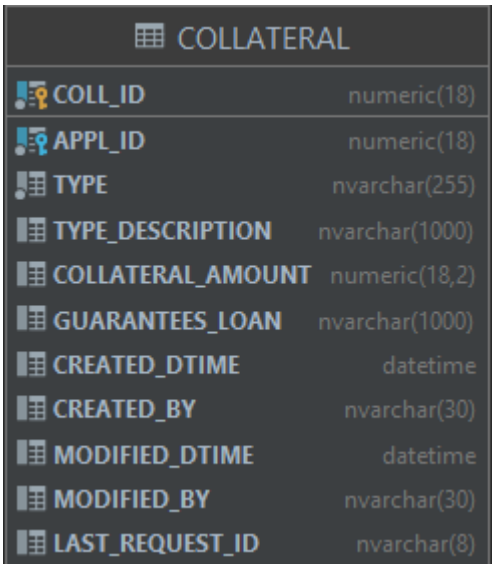
Tabel APPLICATION_CORP_APARTMENT_ASSOCIATION tabelisse saame salvestada korteriühistuga seotud andmed.



Joonis 3. Korteriühistu andmete tabel

3.5.1.4 Tabel APPLICATION_COLLATERAL

Tabelisse APPLICATION_COLLATERAL on võimalik salvestada taotluse tagatisega seotud andmeid.



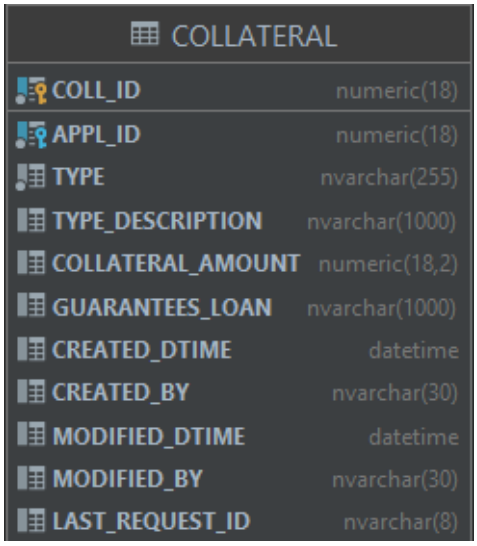
Joonis 4. Tagatisega seotud andmete tabel

3.5.1.5 Tabel APPLICATION_PERSON

Tabelit APPLICATION_PERSON kasutame taotlusega seotud isikute salvestamiseks, näiteks taotluse kontaktisik või korteriühistu esindaja.

3.5.1.6 Tabel APPLICATION_CORP_LIABILITY

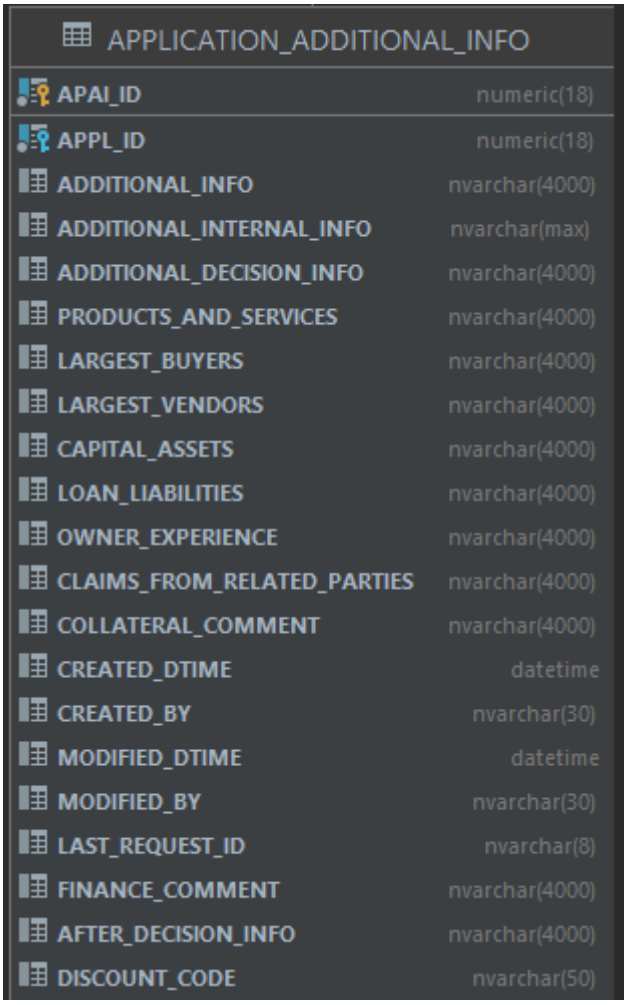
Tabelisse APPLICATION_CORP_LIABILITY saame salvestada korteriühistuga seotud kohustused.



Joonis 5. Korteriühistuga seotud kohustuste tabel

3.5.1.7 Tabel APPLICATION_ADDITIONAL_INFO

Tabelisse APPLICATION_ADDITIONAL_INFO saame salvestada taotluse lisaandmed mis ei ole otsustusprotsessis olulised.



Joonis 6. Taotluse lisainformatsioon tabel

3.6 Otsustuspuu valik

Otsustuspuu on puu struktuuri meenutav diagramm või protsess, mis aitab automatiseerida otsuse tegemist. Antud kontekstis lahendatakse otsustuspuuga küsimust, kas pakkuda kliendile laenulepingut või mitte.

On olemas ka populaarseid otsustuspuid nagu Drools või OpenL Tablets, kuid nende ülesseadmine ning mugavdamine ettevõtte vajadustele kohaseks on väga ajakulukas.

Kuna uue otsustuspuu välja arendamine on väga ajakulukas, kasutatakse antud lahenduses juba olemas olevat otsustuspuud.

Otsustuspuu on lahendatud tavalise puu struktuuri põhimõttel. Otsustuspuu on lahendatud andmebaasi põhiselt kus sisendpunktiks on alliksõlm mis kutsub välja järgmise sõlme. Tulenevalt sõlme tulemusest kutsutakse välja järgmine sõlm või katkestatakse protsess. Sõlmede tulemused salvestatakse eraldi tabelisse ning kuvatakse pärandrakenduses vastavas otsustuspuu tulemuse vaates.

Tuleviku lahendusena ja arhitektuurilise parandusena on plaan tõsta otsustuspuu eraldi rakenduseks, kus uus rakendus võtaks sisendiks JSON-formaadis andmed ning tagastaks tulemuse. See vähendaks pärandrakenduse koormust, lisaks oleks rakendus eraldiseisva meeskonna vastutusala mis vähendaks segadust ning suurendaks ekspertiisi.

Lisaks võiks otsustuspuu sõlmed, seal kus võimalik, joosta paralleelselt, kus ühe sõlme tulemus ei sõltu teise sõlme tulemusest. See vähendaks otsusele kuluvat aega ning parandaks rakenduse võimekust.

3.7 Analüüsi kokkuvõte

Analüüsis sai käsitletud funktsionaalseid nõudeid. Tulenevalt turvalisuse nõuetest ning funktsionaalsetest nõuetest peaks lahendus sisaldama veebiteenuseid. Veebiteenus mis loob ühenduse põhidomeeni ning *proxy*-rakenduse vahel, teine veebiteenus mis ühendab Proxy ning pärandrakenduse. *Proxy*-rakenduse kasutamine hoiab sisedomeeni avalikuse eest nähtamatuna.

Pärandrakenduse eelistamine täielikult uue rakenduse loomise ees annab võimaluse kasutada eelnevalt loodud andmebaasi ning otsustuspuud, mis säästab aega.

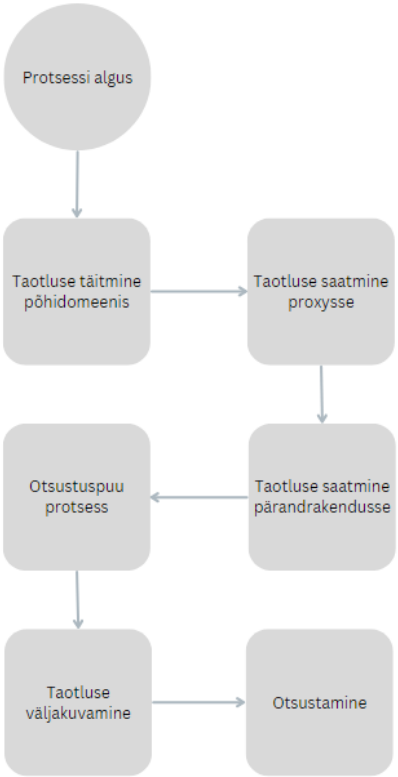
Serveripoolseks programmeerimiskeeleks osutus valituks Java. Autoril on juba eelnev põhjalik kogemus ning tugi arenduskeskkondadele. Teenusepoolse raamistiku valikuks osutus seetõttu Spring Boot. Pärandrakenduse kliendipoolse lahenduse loomiseks kasutatakse *Thymeleaf*-i, kuid tehti meeskonnale soovitus võtta aeg mõtlemaks modernsema lahenduse peale.

Koodi kirjutamiseks võetakse kasutusele IntelliJ IDEA. Andmebaasiks on ettevõte Microsoft poolt arendatav süsteem SQL Server. Koodihaldus toimub ettevõtte GitLab-is.

4 Lahenduse arendus

Lahenduse arendus on jagatud kolme suuremasse peatükki:

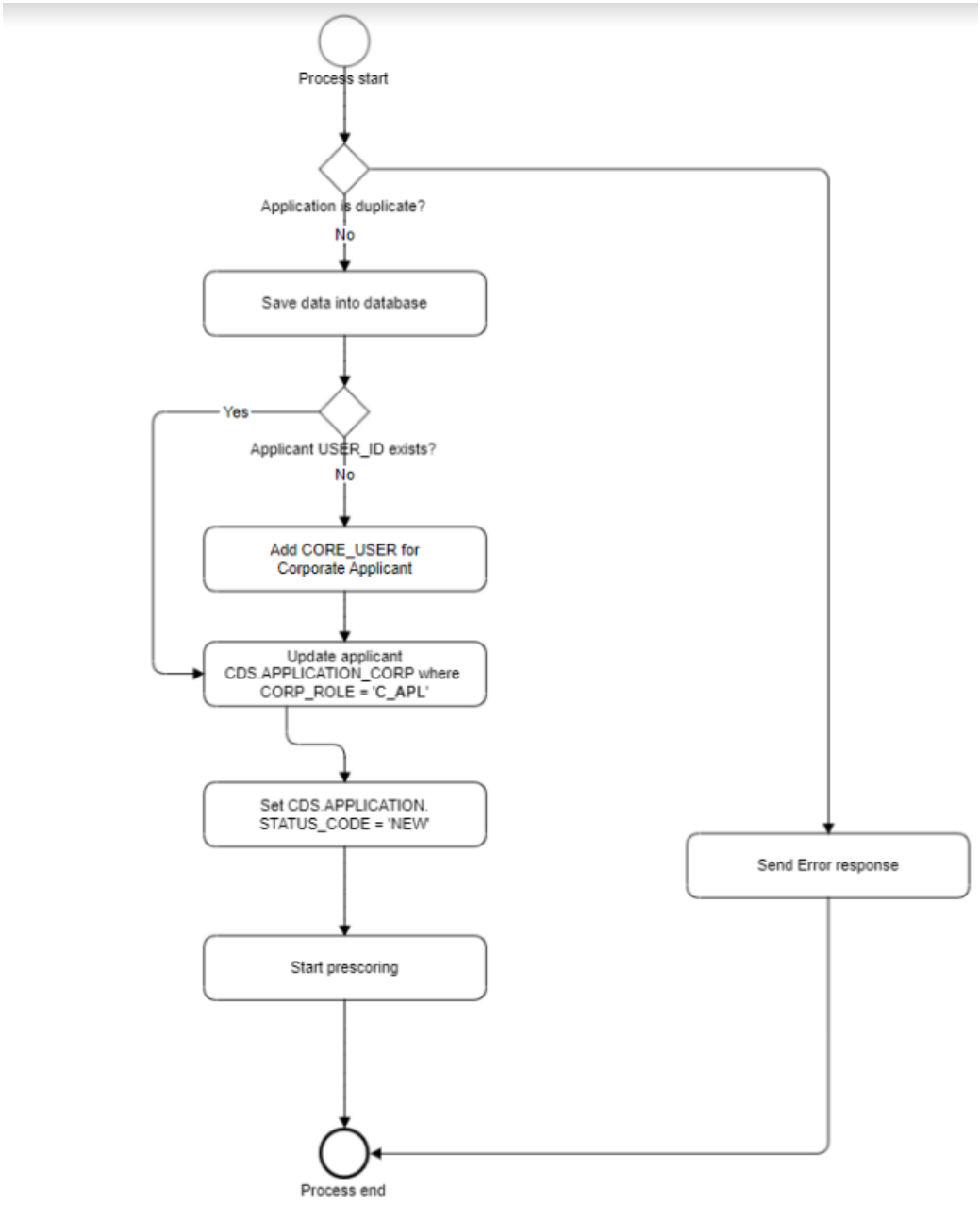
- *Proxy*-süsteemi lahendus, arendus ja testimine.
- Pärandrakenduse poolse lahenduse arendamine, mis sisaldab vastuvõtva veebiteenuse arendamist, pärandrakenduse kliendipoolse kuva arendamist, andmebaasi kohandamist ning lahenduse testimist.
- Andmeaida täiendamine.



Joonis 7. Kergendatud protsessi diagramm

4.1 Pärandrakenduse poolne lahendus

Pärandrakenduse poolse vastuvõtva teenuse protsess on toodud allpool (vt Joonis 8). Joonises kuvatud CORE_USER tähendab ettevõtte keskse süsteemi klienti. Kuna korteriühistu laenutaotlus on ärilaen, saame mugavalt ära kasutada eelnevalt loodud andmebaasi mudelid ning väikeseid muudatusi tehes tagada minimaalse muudatuse teistes taotlustes.



Joonis 8. Vastuvõtva teenuse protsess

4.1.1 Vastuvõttev teenus

Vastuvõtva teenuse jaoks on vaja luua viis erinevat klassi: vastuvõetavad andmed, tagastatavad andmed, kontroller, andmete konverter, ning teenuse klass. Lisaks luuakse abiklass valideerimise jaoks. Valideerimises kasutatav abiklass sisaldab andmete valideerimis meetodeid, mida ei saa annotatsioonidega lahendada.

4.1.1.1 Vastuvõetavad- ning tagastatavad andmed

Vastuvõetavad andmed on kuvatud Tabelis 1. Tagastatavateks andmeteks on laenutaotluse number. Vastuvõetavate andmete klassi loomist täiendatakse *@Data* annotatsiooniga, mis tagab mugava *getter* ning *setter*-meetodite kasutamise[17]. Lisaks kasutatakse *@JsonIgnoreProperties* annotatsiooni, mis ignoreerib päringuga saadud välju, mida ei ole klassis kirjeldatud [16].

Sisendi valideerimiseks kasutatakse vastuvõtvate andmete klassis *Javax* valideerimise teegi annotatsioone, nt *@NotBlank*, *@NotNull* või *@Size* [18].

```
@Data
@JsonIgnoreProperties(ignoreUnknown = true)
public class ApartmentAssociationLoanJsonRequest {

    @NotBlank(message = "cds.mobile_api.validator.constraints.not_blank.message")
    private String clientRequestId;

    @Valid
    @NotNull(message = "cds.mobile_api.validator.constraints.not_blank.message")
    private ApartmentAssociationLoanApplication apartmentAssociationLoanApplication;

    @Data
    @JsonIgnoreProperties(ignoreUnknown = true)
    public static class ApartmentAssociationLoanApplication {

        @Size(max = 1, message = "cds.mobile_api.validator.constraints.size.message")
        @NotBlank(message = "cds.mobile_api.validator.constraints.not_blank.message")
        private String channelCode;

        @Valid
        @NotNull(message = "cds.mobile_api.validator.constraints.not_blank.message")
        private ApartmentAssociationLoanJsonRequest.ApplicationApartmentAssociationLoan
applicationApartmentAssociationLoan;

        @Valid
        @NotNull(message = "cds.mobile_api.validator.constraints.not_blank.message")
        private ApartmentAssociationLoanJsonRequest.ApplicantApartmentAssociation
applicantApartmentAssociation;

        @Valid
        @NotNull(message = "cds.mobile_api.validator.constraints.not_blank.message")
        private ApartmentAssociationLoanJsonRequest.ContactPerson contactPerson;

        @Valid
        @NotNull(message = "cds.mobile_api.validator.constraints.not_blank.message")
        private ApartmentAssociationLoanJsonRequest.ApplicationSubmitter applicationSubmitter;
```

Joonis 9. Päringu andmete klassi näide

4.1.1.2 Kontrolleri klass

Kontrolleri klassi loomisel kasutatakse kahte annotatsiooni. Esimeseks on *@PreAuthorize* koos õiguste täiendusega, mis tagab olukorra, kus ainult kindlad rakendused saavad otspunkti poole pöörduda [21]. Teisalt täiendatakse klassi Spring raamistiku *@RestController* annotatsiooniga, mis märgib kontroller klasse [19].

Klassi sõltuvusi täiendatakse *@Resource* annotatsiooniga, mis võimaldab sõltuvuste automaatse tuvastuse ja süstimise [14].

Klassi luuakse kolm meetodit, millest kaks on abimeetodid. Peamiseks meetodiks on otspunkti meetod. Otspunkti meetodit täiendatakse *@PostMapping* annotatasiooniga, mis märgib, et meetod võtab vastu POST-päringuid märgitud aadressil ning märgitud formaadis [20]. Meetodi sisendiks on vastuvõetavate andmete klass ning meetod tagastab tagastatavate andmete klassi.

Meetodi sisendit täiendatakse *@Valid* ning *@RequestBody* annotatsiooniga. *@Valid* märgib, et sisend tuleb Spring raamistiku poolt ära valideerida ning *@RequestBody* märgib, et meetodi parameeter tuleb väärtustada vastavalt päringu keha sisule [19].

Meetodi kehas olev kood peegeldab eelnevalt mainitud protsessi (vt Joonis 8). Esmalt kasutatakse loodud valideerimise abiklassi, tagamaks korduvtaotluste eemaldamise ning lisaks valideeritakse, kas autentimis andmed on taotlusel olemas. Seejärel kasutatakse konverteerimise klassi, et sissetulevad andmed teisendada andmebaasi mudelile vastavale kujule. Lõpuks lisatakse konverteeritud andmed andmebaasi ning tagastatakse andmeklass, mis sisaldab taotluse numbrit.

```
@PreAuthorize("hasAnyRole(this.roles)")
@RestController(value = "ApartmentAssociationLoanNew")
public class ApartmentAssociationLoanJsonWs {

    @Resource
    private ApartmentAssociationLoanJsonConverter converter;

    @Resource
    private ApartmentAssociationLoanJsonService loanApplicationJsonService;

    @Resource
    private CorploanApplicationService corploanApplicationService;

    @Resource
    private ApartmentAssociationLoanValidatorUtil apartmentAssociationLoanValidatorUtil;

    @PostMapping(value = "/services/application/apartment_association/new",
            consumes = {MediaTypeConstants.APPLICATION_JSON_UTF8_VALUE},
            produces = {MediaTypeConstants.APPLICATION_JSON_UTF8_VALUE}
    )
    public ApartmentAssociationLoanJsonResponse postApplication(
            @Valid @RequestBody ApartmentAssociationLoanJsonRequest request
    ) throws CustomLhvException {
        apartmentAssociationLoanValidatorUtil.handleDuplicate(request);
        ApartmentAssociationLoanValidatorUtil.validateAuthData(request);
        BuloApplicationModel corploanApplicationModel = converter.convert(request);
        corploanApplicationService.setDefaultValuesToCorploanModel(corploanApplicationModel);
        apartmentAssociationLoanValidatorUtil.validateRequest(corploanApplicationModel);
        Long applId = loanApplicationJsonService.addNewApplication(corploanApplicationModel);

        return createApplicationJsonResponse(applId);
    }
```

Joonis 10. Kontrolleri näide

Abimeetoditeks luuakse *@PreAuthorize* annotatsiooni kasutatav meetod, mis tagastab nimekirja rakendustest, mis otspunkti poole pöörduda võivad [21].

Teiseks abimeetodiks on tagastatavate andmete klassi sisu loomise abimeetod.

4.1.1.3 Andmete konverter klass

Andmete konverter klass on abiklass, mis konverteerib päringust tulenevad andmed andmebaasi mudelile kohaseks. Klassi loomisel kasutatakse komponenti *@Component*, mis tagab klassi automaatse tuvastuse [14].

```
@Component
public class ApartmentAssociationLoanJsonConverter implements
Converter<ApartmentAssociationLoanJsonRequest, CorpLoanApplicationModel> {

    @Resource
    private ProductService productService;

    @Resource
    private LhvMessageSourceAccessor messageSourceAccessor;

    @Override
    @NonNull
    public BuloApplicationModel convert(ApartmentAssociationLoanJsonRequest request) {
        final ApartmentAssociationLoanJsonRequest.ApartmentAssociationLoanApplication
loanApplication = request
                .getApartmentAssociationLoanApplication();
        BuloApplicationModel model = new BuloApplicationModel();
        model.setApplication(getApplication(request));
        model.setCorpLoans(Collections.singletonList(getApplicationCorploan(loanApplication)));
        model.setApplicant(getCorporateApplicant(loanApplication));
        model.setApplicationAdditionalInfo(getApplicationAdditionalInfo(loanApplication));

model.setCollaterals(getApplicationCollaterals(loanApplication.getApplicationApartmentAssociati
onLoan()));

model.setManagementBoardMemberList(getApplicationManagementBoardList(loanApplication.getApplica
ntApartmentAssociation()));
        return model;
    }
```

Joonis 11. Konverter klassi näide

4.1.1.4 Teenuse klass

Teenuse klassi kasutatakse andmebaasiga suhtlemiseks. Klassi loomisel kasutatakse annotatsioone *@Log4j2* ning *@Service*. Log4j2 on teek tagamaks rakenduse logimise [22]. *@Service* annotatsioon on vajalik klassi automaatseks tuvastamiseks ja sõltuvuste süstimiseks, mis on teostatav *@Resource* annotatsiooni abil [14].

Klassis sisendmeetod on uue taotluse lisamine, abimeetodid loevad päringust tuleva info ning konverteerivad selle ümber andmebaasi mudelile kohaseks andmeteks.

4.1.1.5 Valideerimise klass

Valideerimise abiklassi kasutatakse valideerimismeetodite jaoks mida annotatsioonidega lahendada ei saa. Klassi loomisel kasutatakse *@Log4j2* ning *@Service* annotatsioone[14].

Sõltuvused täiendatakse @Resource annotatsiooniga [14]. Mitmekordse taotluse esitamise mure lahendatakse ExpirableObjectLocks klassi abil, mis salvestab kindlaks ajaks rakenduse mällu taotluse tunnuse. Sama tunnuse saabumisel otspunkti tagastatakse taotlejale viga. Lisaks valideeritakse klassis ka autentimisandmete olemasolu.

4.1.2 Vastuvõtva teenuse automaattestid

Kõikidele klassidele luuakse JUnit ühiktestid, milles kasutatakse Mockito teeki. Testimisklassi loomisel täiendatakse klassi *@ExtendWith* annotatsiooni MockitoExtension sisendiga, mis laiendab loodavat klassi sisendis oleva teegiga [23].

Klassi sõltuvused täiendatakse *@Mock* annotatsiooniga, mis tuleneb Mockito teegist ning imiteerib sõltuvusklasse. Testitav klass täiendatakse *@InjectMocks* annotatsiooniga mis sisendab sõltuvused automaatselt [24].

Testimisstsenaariumites kasutatakse veel *when* ja *thenReturn* meetodeid imiteerimaks sõltuvuste tagastatavaid andmeid [25].

```
@ExtendWith(MockitoExtension.class)
class ApartmentAssociationLoanJsonWsUnitTest extends BaseCdsUnitTest {

    @Mock
    private ExpirableObjectLocks<String> byApplicantIdCodeLock;

    @Mock
    private ApartmentAssociationLoanJsonConverter converter;

    @Mock
    private LhvValidatorFactoryBean validator;

    @Mock
    private ApartmentAssociationLoanJsonService apartmentAssociationLoanJsonService;

    @Mock
    private CorploanApplicationService corploanApplicationService;

    @Spy
    @InjectMocks
    private ApartmentAssociationLoanValidatorUtil apartmentAssociationLoanValidatorUtil;

    @InjectMocks
    private ApartmentAssociationLoanJsonWs apartmentAssociationLoanJsonWs;

    @Test
    void should_process_validated_request() throws Exception {
        mockConvertedApplicationModel();
        when(apartmentAssociationLoanJsonService.addNewApplication(any())).thenReturn(123L);
        when(byApplicantIdCodeLock.tryLock(anyString())).thenReturn(true);

        ApartmentAssociationLoanJsonResponse response =
apartmentAssociationLoanJsonWs.postApplication(
                getApartmentAssociationJsonRequest());

        verify(converter).convert(any());
        verify(apartmentAssociationLoanJsonService).addNewApplication(any());
        assertEquals(Long.valueOf(123L), response.getApplicationId());
    }
```

Joonis 12. Pärandrakenduse otspunkti testimine

4.1.3 Andmebaasi kohandamine

Kuna lahenduses kasutatakse juba loodud tabeleid saab mugavalt, vähese mõjutusega lisada lahendusest tulenevad muudatused.

Pärandrakendus kasutab Liquibase andmebaasi haldamise lahendust, luues vastavad andmebaasi muutmiskomplektid, mida loetakse rakenduse käivitamisel.

Lahenduse jaoks on vaja teha kaks muudatust (vt Joonis 13):

- lisada CDS.APPLICATION_CORPLOAN_APPARTMENT_ASSOCIATION tabelisse KREDEX_GRANT_AMOUNT väli mis ei tohi olla *null* ning on *numeric*(18,2) tüüpi;
- muuta CDS.APPLICATION_PERSON tabelis PERSON_ROLE väärtuse lubatud pikkust. Esimene muutatus salvestab KredEx-i käendust, teine väli salvestab juhataja rolli korteriühistus.

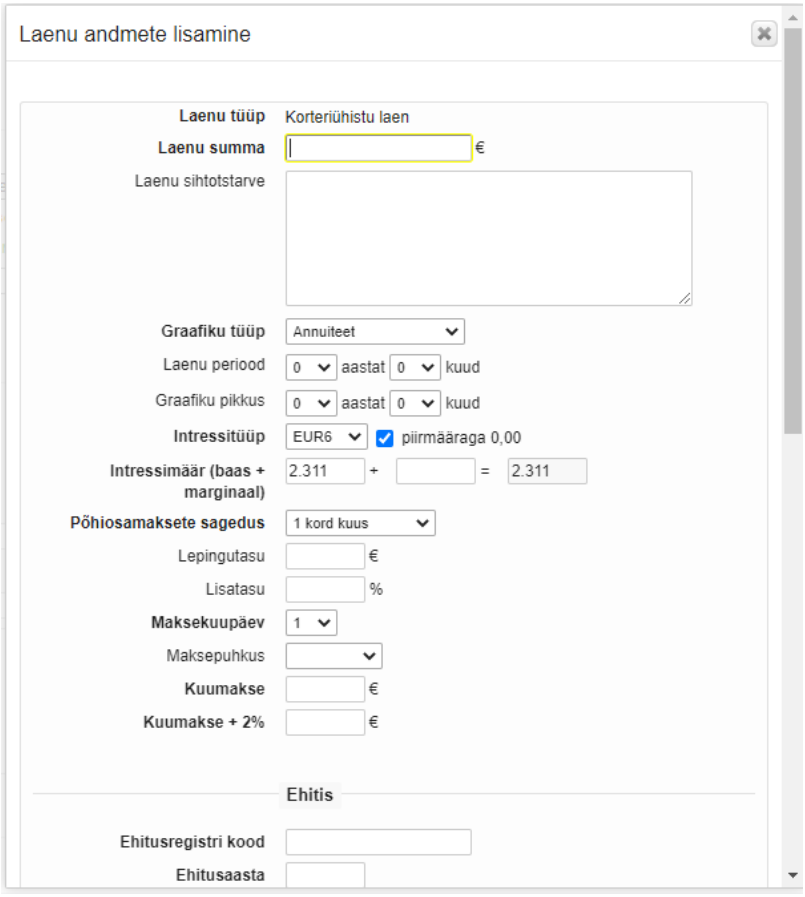
Andmebaasis on ka ajaloo säilitamiseks mõeldud tabelid, mida täiendatakse samuti vastavalt muudatustele. Andmebaasi ajalugu laetakse automaatselt, seega peab ajaloo skripte täiendama uue loodud väljaga.

```
+ --liquibase formatted sql
+
+ --changeset rommi:CDS_4647_1 runInTransaction:true splitStatements:false
+ ALTER TABLE CDS.APPLICATION_CORPLOAN_APARTMENT_ASSOCIATION ADD KREDEX_GRANT_AMOUNT numeric(18, 2) NULL;
+ ALTER TABLE CDS_BAK.APPLICATION_CORPLOAN_APARTMENT_ASSOCIATION_BAK ADD KREDEX_GRANT_AMOUNT numeric(18, 2) NULL;
+
+ --changeset rommi:CDS_4647_2 runInTransaction:true splitStatements:false
+
+ ALTER TABLE CDS.APPLICATION_PERSON ALTER COLUMN PERSON_ROLE nvarchar(25) NOT NULL;
+ ALTER TABLE CDS_BAK.APPLICATION_PERSON_BAK ALTER COLUMN PERSON_ROLE nvarchar(25) NOT NULL;
```

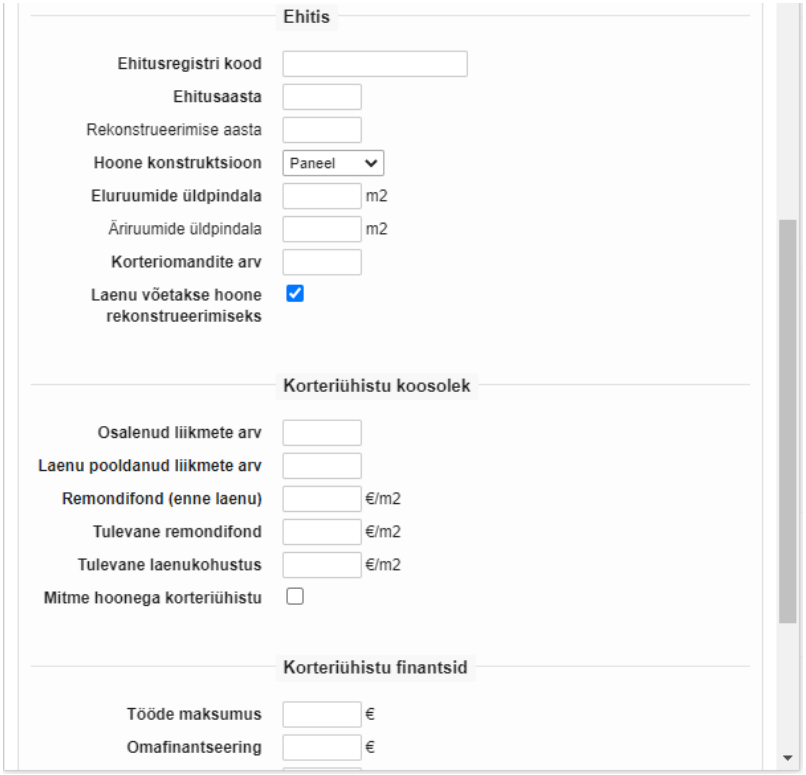
Joonis 13. Andmebaasi muudatused

4.1.4 Kliendipoolse vaate kohandamine

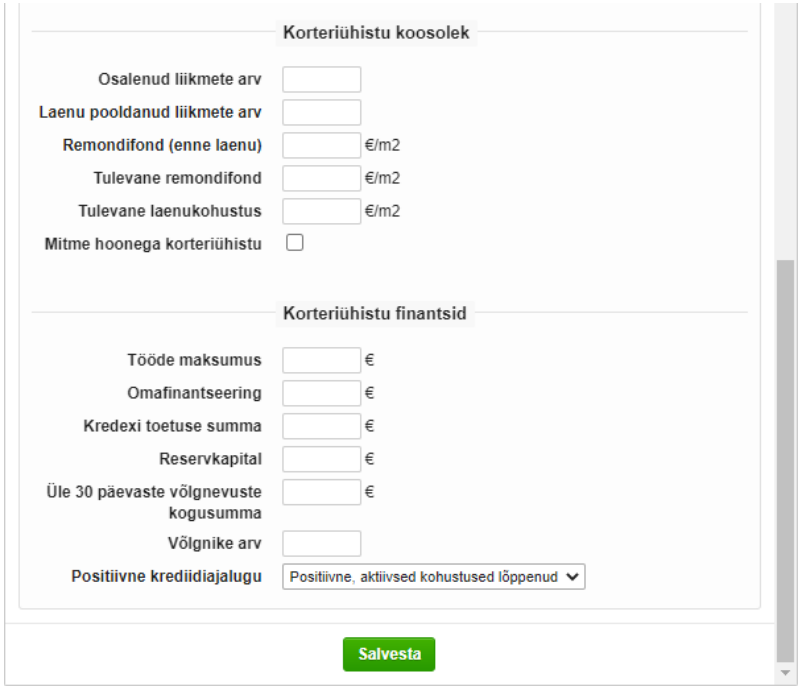
Pärandrakenduse kliendipoolse vaatel saame ära kasutada juba eelnevalt loodud ärilaenu kuva ning kuva mudelit, täiendades seda korteriühistu laenu spetsiifiliste väljadega.



Joonis 14. Korteriühistu laenu lisamine 1



Joonis 15. Korteriühistu laenu lisamine 2



Joonis 16. Korteriühistu laenu lisamine 3

4.2 *Proxy*-poolne lahendus

Proxy-poolseks lahenduseks kasutatakse eraldiseisvat avaliku domeeni *backend* rakendust. Rakendusse luuakse REST-otspunkt, mis suunab päringu pärandrakenduse poole. Seeläbi on võimalik peita eelmises peatükis loodud siserakenduse otspunkti. Avaliku domeeni suhtlus käib ainult läbi kindla otspunkti.

Seda saavutamaks luuakse kõigepealt veebiteenusekliendi teenuse klass, mida on täiendatud *@Service* annotatsiooniga. Annotatsioon on vajalik klassi automaatseks tuvastamiseks [14].

Vajaliku funktsionaalsuse loomiseks võetakse kasutusele Spring raamistikus olev RestTemplate abiklass, mida on võimalik konfigureerida pärandrakenduse jaoks. Konfiguratsioon on eelnevalt täpsustatud klassis, mis on täiendatud *@Configuration* annotatsiooniga. See tähendab, et antud töö raames ei ole vaja luua eraldi liidestust, kuna see on juba eelnevalt tehtud. Liidestus tagab klasside kaardistamise ning veahalduse kui pärandrakendus ei vasta. Samuti tagastatakse pärandrakenduse poolt tulevad veateated.

Klassis kasutame veel *@Autowired* annotatsiooni mis võimaldab automaatselt tuvastatud sõltuvusi klassidesse automaatselt süstida [15].

Klassi luuakse kaks meetodit. Esimeses meetodis pannakse paika pärandrakenduse meetodi väljakutsumise parameetrid, teine meetod teostab HTTP POST-päringu pärandrakenduses loodud meetodi pihta. Teises meetodis kasutame ka geneerilisi andmetüüpe, et tulevikus loodavad lahendused ei peaks meetodit muutma.

```
@Service
public class CorploanJsonWsClient {

    public final RestTemplate cdsJsonRestTemplate;

    @Autowired
    public CorploanJsonWsClient(RestTemplate cdsJsonRestTemplate) {
        this.cdsJsonRestTemplate = cdsJsonRestTemplate;
    }

    public ApartmentAssociationLoanResponse
sendApplicationToCds(ApartmentAssociationLoanRequest request, HttpHeaders headers) {
        return doPost("new", request, headers, ApartmentAssociationLoanResponse.class);
    }

    private <R, T> T doPost(String endpoint, R request, HttpHeaders headers, Class<T>
respClass) {
        URI uri = cdsJsonRestTemplate.getUriTemplateHandler().expand("/endpoint/url/" +
endpoint);
        return cdsJsonRestTemplate.postForObject(uri, new HttpEntity<>(request, headers),
respClass);
    }
}
```

Joonis 17. Veebiteenusekliendi teenuse klass

Järgmisena luuakse vastus ja päring klassid, mis sisaldavad tagastatavaid ja saadetavaid andmeid. Saadetavateks andmeteks on analüüsis täpsustatud andmed ning vastuse klass sisaldab üht andmevälja milleks on taotluse number. Antud klassides kasutatakse kahte annotatsiooni. Esimene on pärit Lombok teegist, mis loob hõlpsaks klassi väljade *getter*-ning *setter*-meetodid. Teine annotatsioon on *@JsonIgnoreProperties*, mis ignoreerib andmevälju, kui need ei ole antud klassis täpsustatud [16].

Pärast veebiteenusekliendi teenuse klassi loomist saab luua otspunkti kontroller klassi, mis kasutab eelnevalt loodud veebiteenusekliendi teenuse klassi. Kontroller klassi loomisel kasutatakse viit annotatsiooni. *@Log4j2* annotatsiooni kasutatakse rakenduse logimiseks. *@PublicApi* on ettevõtte kohandatud annotatsioon, millega märgitakse avalikke kontrollerklasse. *@RestController* annotatsioon on Spring raamistiku annotatsioon, mis märgib kontroller klasse [19].

@RequestMapping on annotatsioon, mida antud juhul kasutatakse alamaadressi märkimiseks ning vastus- ja päringformaadi täpsustamiseks [27]. Lisaks kasutatakse ka *Swagger*-i annotatsiooni *@Api* otspunkti dokumentatsiooni jaoks.

Luuakse üks meetod, mis on annoteeritud *@PostMapping* annotatsiooniga ning selle kaudu on võimalik sätestada otspunkti aadress [20]. Meetodi sisendiks on avaliku domeeni kliendipoolsest vaatest tulenev JSON-formaadis päring ning HTTP päised. Meetodi väljund on pärandrakendusest tulenev vastus. Meetodi kehaks on kasutaja autentimisandmete sätestamine ning juhul kui kasutajal puudub sessioon ka saneerimine. Sessiooni puudumine tähendab, et kui päring otspunkti vastu teostati, ei olnud kasutaja sisse logitud. Saneerimise käigus eemaldatakse autentimisandmed, mille järel tagastab pärandrakendus vea.

Seejärel teostatakse HTTP POST-päring kasutades eelnevalt loodud veebiteenusekliendi klassi POST-meetodit.

```
@Log4j2
@PublicApi
@RestController
@RequestMapping(value = "/laen", consumes = MediaType.APPLICATION_JSON_VALUE, produces =
MediaType.APPLICATION_JSON_VALUE)
@Api(tags = SwaggerTagName.CDS_CORPLOAN)
public class CorploanJsonProxyController {

    private final CorploanJsonWsClient corploanJsonWsClient;
    private final ExternalAuthSessionManagementService authSessionManagementService;
    private final UserDataProviderWs userDataService;

    @Autowired
    public CorploanJsonProxyController(
        CorploanJsonWsClient corploanJsonWsClient,
        ExternalAuthSessionManagementService authSessionManagementService,
        UserDataProviderWs userDataService) {
        this.corploanJsonWsClient = corploanJsonWsClient;
        this.authSessionManagementService = authSessionManagementService;
        this.userDataService = userDataService;
    }

    @LogoutExternalSession
    @PostMapping("/endpoint/url")
    public ApartmentAssociationLoanResponse postApplication(
        @RequestBody ApartmentAssociationLoanRequest application,
        @RequestHeader HttpHeaders headers) {
        var sessionUser = this.authSessionManagementService.getAuthUser();;
        if (sessionUser != null) {
            CoreUserBasic privateUser =
userDataService.getPrivateUserDataBasic(sessionUser.getAuthUserId());
            fillAuthAndSubmitterData(sessionUser.getAuthUserId(),
application.getApartmentAssociationLoanApplication(), privateUser);
        } else {
            sanitizeAnonApplication(application.getApartmentAssociationLoanApplication());
        }
        return corploanJsonWsClient.sendApplicationToCds(application, headers);
    }
```

Joonis 18. Kontroller klassi näide

Klassis luuakse ka veahalduseks abimeetod. Abimeetodis kasutatakse *@ExceptionHandler* annotatsiooni koos *RestClientException* sisendiga. Annotatsioon võimaldab suunata veahalduse antud klassi ning garanteerib vastuse ka juhul kui tekib viga [26].

```
@ExceptionHandler(RestClientException.class)
public RestError restClientExceptionHandler(RestClientException restClientException,
HttpServletResponse response) {
    Throwable cause = restClientException.getCause();
    RestError restError = new RestError();
    if (cause instanceof ResourceAccessException) {
        return resourceAccessExceptionHandler(response);
    } else if (cause instanceof CdsRestException) {
        response.setStatus(HttpStatus.BAD_REQUEST.value());
        restError = ((CdsRestException) cause).getCdsErrorResponse();
    } else {
        log.error(restClientException);
        response.setStatus(HttpStatus.INTERNAL_SERVER_ERROR.value());
        restError.setStatus(HttpStatus.INTERNAL_SERVER_ERROR);
        restError.setMessage("Technical error");
        restError.setCode("UNKNOWN_ERROR");
    }
    return restError;
}
```

Joonis 19. Veahalduri näide

4.2.1 *Proxy*-poolse lahenduse automaattestid

Proxy-poolsele lahendusele luuakse ühiktestid kasutades JUnit raamistikku ning Spring raamistiku *MockHttpSession* klassi funktsionaalsust, nt imiteerimaks avaliku domeeni sessiooni [28]. Lisaks kasutatakse ka WireMock teegi võimalusi, mis loob võimaluse imiteerida pärandrakenduse käitumist ja vastuseid.

Testides kasutatakse lisaks ka *@Resource* annotatsiooni, mis märgib testide jooksmiseks vajalikud ressursid [14].

@BeforeEach annotatsiooniga märgitud setup meetod loob testimiseks vajaliku keskkonna [29]; vajaliku keskkonna loomiseks imiteeritakse sessioon ning luuakse stsenaarium testide jaoks. Stsenaarium loob reegli, kus kindla päringu puhul peaks tagastatama kindel vastus.

```
class CorploanJsonProxyControllerTest extends WireMockTestBase {

    private static final String APARTMENT_ASSOCIATION_LOAN_APPLICATION_PATH =
"/corploan/apartment_association/application";
    private static final String APARTMENT_ASSOCIATION_LOAN_BUILDING_PATH =
"/corploan/apartment_association/building";

    private static final String BUILDING_DATA_REQUEST_PATH =
"service/cds/corploan/apartment_association_building_request.json";
    private static final String BUILDING_DATA_RESPONSE_PATH =
"service/cds/corploan/apartment_association_building_response.json";
    private static final String BUILDING_DATA_MISSING_REQUEST_PATH =
"service/cds/corploan/apartment_association_building_missing_request.json";
    private static final String BUILDING_DATA_MISSING_RESPONSE_PATH =
"service/cds/corploan/apartment_association_building_missing_response.json";

    @Resource
    private CorploanJsonProxyController corploanJsonProxyController;

    private UserDataService userDataService;

    private MockHttpSession mockSession;

    @BeforeEach
    void setup() throws IOException {
        mockSession = login("testuser", "1", "user.data.modify");
        ThreadContextUtil.addToContext(ThreadContextAttribute.SESSION_ID_HASH, "ABCDEFGH");

        userDataService = mock(UserDataService.class);

        String body =
TestResourceLoader.getFile("service/cds/corploan/apartment_association_loan_response.json");
        stubJson(body, "/cds/services/application/apartment_association/new",
containing("3sdfsa"));

        body = TestResourceLoader.getFile(BUILDING_DATA_RESPONSE_PATH);
        stubJson(body, "/cds/services/application/apartment_association/building",
containing("Estonia pst 4, Tallinn, Harjumaa"));
        body = TestResourceLoader.getFile(BUILDING_DATA_MISSING_RESPONSE_PATH);
        stubJson(body, "/cds/services/application/apartment_association/building",
containing("Olematu tn 26, Tallinn, Harjumaa"));

        ReflectionTestUtils.setField(corploanJsonProxyController, "userDataService",
userDataService);
        setupErrorMocks("/cds/services/application/apartment_association/new");
    }
```

Joonis 20. Koodinäide testimis stsenaariumi loomisest

Seejärel luuakse JUnit-i *@Test* annotatsiooni abil testimisjuhtumid [30]. Testimisjuhtumis luuakse autentimisandmete imitatsioon. Seejärel kasutatakse Mockito teegist tulenevaid *when* ja *thenReturn* meetodeid, et tagastada loodud autentimisandmete imitatsioon, kui automaattest autentimisandmeid pärib.

Seejärel teostatakse POST-päring, andes kaasa päringu sisu ning täpsustatakse päringu formaat. Lisaks täpsustatakse sessioon, mis eelnevalt *@BeforeEach* annotatsiooniga märgitud meetodis loodi. Lõpuks kirjutatakse testi läbimiseks vajalikud eeldused.

```
@Test
void testCds5xx() throws Exception {
    CoreUserBasic privateUser = getPrivateUser();
    when(userDataService.getPrivateUserDataBasic(any(Long.class))).thenReturn(privateUser);
    String requestPayload =
TestResourceLoader.getFile("service/cds/corploan/apartment_association_loan_request_bad_gateway
.json");
    String expectedJsonResponse =
TestResourceLoader.getFile("service/cds/bad_gateway_error_response.json");
    mvc.perform(post(APARTMENT_ASSOCIATION_LOAN_APPLICATION_PATH).session(mockSession)
                    .contentType(MediaType.APPLICATION_JSON_VALUE)
                    .content(requestPayload)
                    .accept(MediaType.APPLICATION_JSON))
        .andExpect(status().isBadGateway())
        .andExpect(content().contentType("application/json"))
        .andExpect(content().json(expectedJsonResponse));
}
```

Joonis 21. Koodinäide testimisjuhtumist

4.3 Andmeaida täiendamine

Andmeait on ettevõttesisene andmebaas, mille andmete järgi tehakse statistikat ning muid sarnaseid tegevusi. Andmeaita laetakse andmeid perioodiliselt. Andmeaita tuleb täiendada andmebaasimuudatustega ning lisada väli *bcp* import- ning eksportskripti [31].

```
USE [DWH_CDS]
GO

ALTER TABLE CDS.APPLICATION_CORPLOAN_APARTMENT_ASSOCIATION ADD KREDEX_GRANT_AMOUNT numeric(18, 2) NULL;
ALTER TABLE CDS.APPLICATION_PERSON ALTER COLUMN PERSON_ROLE nvarchar(25) NOT NULL;
ALTER TABLE CDS_BAK.APPLICATION_PERSON_BAK ALTER COLUMN PERSON_ROLE nvarchar(25) NOT NULL;

USE [DWH_TEMP]
GO
ALTER TABLE CDS_BAK.APPLICATION_PERSON_BAK_TEMP ALTER COLUMN PERSON_ROLE nvarchar(25) NOT NULL;
```

Joonis 22. Andmeaida täiendamine

5 Kokkuvõte

Bakalareusetöö eesmärgiks oli luua lahendus korteriühistu laenutaotluse otsuse automatiseerimiseks. Lahenduse skoobiks oli laenutaotluse esitamise võimekus põhidomeenis ning selle automaathindamine. Lisaks oli vajalik taotluse sidumine otsustuspuuga. Lahendus pidi olema turvaline ning ettevõtte nõuetele vastav.

Töö analüüs andis hea ülevaate tehnoloogiatest ning planeeritavast funktsionaalsusest. Analüüsi käigus selgitati välja kõige efektiivsem viis skoobi saavutamiseks. Lõputöö arenduskäiku kirjeldav peatükk annab ülevaate erinevatest arhitektuurikihtidest, veebiteenuste loomisest, rakenduste liidestamisest, andmete konverteerimisest ning lahenduse testimisest.

Projekti võib lugeda õnnestunuks. Korteriühistu laenu on võimalik taotleda läbi avaliku domeeni. Taotluse eelhindamist on võimalik teostada läbi pärandrakenduse ning taotlus on mugavalt hallatav läbi pärandrakenduse liidese.

Projekti järel aitamis ülesandeid on olnud minimaalselt ning äriosakond on lahendusega rahul. Tänu loodud projektile saavutati märkimisväärne ajavõit võrreldes eelneva protsessiga.

Teostatud lahendusega on esitatud 1. Jaanuari 2023 seisuga 360 taotlust. Taotluse otsustus kiirus on 43% vähenedes 30 päevalt 17le päevale.

Kasutatud kirjandus

[1] AS LHV Group, „Korteriühistu laen“ [Võrgumaterjal]. Loetud aadressil: https://www.lhv.ee/et/korteriyhistu-laen. [Kasutatud 8. oktoober 2022].

[2] „Korteriühistuseadus“ [Võrgumaterjal] Loetud aadressil: https://www.riigiteataja.ee/akt/125052012017. [Kasutatud 8. oktoober 2022].

[3] AS LHV Group, „Ettevõttest,“ [Võrgumaterjal]. Loetud aadressil: https://www.lhv.ee/et/ettevottest. [Kasutatud 8. oktoober 2022].

[4] The PHP Group, „What is PHP?“, [Võrgumaterjal]. Loetud aadressil: https://www.php.net/manual/en/intro-whatis.php [Kasutatud 31.10.2022].

[5] Ruby Community, „About Ruby“, Loetud aadressil: https://www.ruby-lang.org/en/about/ [Kasutatud 31.10.2022].

[6] Mozilla, „MDN Web docs“, Loetud aadressil: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript [Kasutatud 31.10.2022].

[7] Sharpened Productions, „TechTerms – C#“, Loetud aadressil: https://techterms.com/definition/c_sharp [Kasutatud 31.10.2022].

[8] Stackscale, „Most popular programming languages 2022“, Loetud aadressil: https://www.stackscale.com/blog/most-popular-programming-languages/ [Kasutatud 31.10.2022].

[9] Pallets Team, „Flask – Security Considerations“, Loetud aadressil: https://flask.palletsprojects.com/en/2.2.x/security/ [Kasutatud 02.11.2022].

[10] Vaadata, „Node.js: Common vulnerabilities and security best practices“, Loetud aadressil: https://www.vaadata.com/blog/node-js-common-vulnerabilities-security-best-practices/ [Kasutatud 02.11.2022].

[11] Honeybadger Industries, „Security Risks On Rails“, Loetud aadressil: https://www.honeybadger.io/blog/rails-security-risks-part-3/ [Kasutatud 02.11.2022].

[12] Eesti Vabariigi Valitsus, „Aastaks 2050 tuleb renoveerida suur osa hoonetest“, Loetud aadressil: https://www.valitsus.ee/uudised/aastaks-2050-tuleb-renoveerida-suur-osa-elamutest-ja-hoonetest [Kasutatud 02.11.2022].

[13] John R. Vacca, „Network and System Security“, Loetud aadressil: https://www.sciencedirect.com/topics/computer-science/proxy-server [Kasutatud 04.11.2022].

[14] [Võrgumaterjal]. Loetud aadressil: https://www.baeldung.com/spring-component-repository-service [Kasutatud 19.11.2022].

[15] Tarnum Javam, „@Component vs @Repository and @Service in Spring“, Loetud aadressil: https://www.baeldung.com/spring-autowire [Kasutatud 19.11.2022].

[16] FasterXML, „Annotation Type JsonIgnoreProperties“, Loetud aadressil: https://fasterxml.github.io/jackson-annotations/javadoc/2.6/com/fasterxml/jackson/annotation/JsonIgnoreProperties.html [Kasutatud 20.11.2022].

[17] Lombok, „All together now: A shortcut for @ToString, @EqualsAndHashCode, @Getter on all fields, @Setter on all non-final fields, and @RequiredArgsConstructor“, Loetud aadressil: https://projectlombok.org/features/Data [Kasutatud 20.11.2022].
[18] Tarnum Javam,“Difference Between @NotNull, @NotEmpty, and @NotBlank Constraints in Bean Validation“, Loetud aadressil: https://www.baeldung.com/java-bean-validation-not-null-empty-blank [Kasutatud 20.11.2022].
[19] Tarnum Javam, „The Spring @Controller and @RestController Annotations“, Loetud aadressil: https://www.baeldung.com/spring-controller-vs-restcontroller [Kasutatud 20.11.2022].
[20] VMWare, „Annotation Interface PostMapping“, Loetud aadressil: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/bind/annotation/PostMapping.html [Kasutatud 20.11.2022].
[21] Tarnum Javam,“Introduction to Spring Method Security“, Loetud aadressil: https://www.baeldung.com/spring-security-method-security [Kasutatud 20.11.2022].
[22] The Apache Software Foundation, „Apache Log4j™ 2“, Loetud aadressil: https://logging.apache.org/log4j/2.x/ [Kasutatud 20.11.2022].
[23] JUnit Team, „Annotation Interface ExtendWith“, Loetud aadressil: https://junit.org/junit5/docs/5.8.0/api/org.junit.jupiter.api/org/junit/jupiter/api/extension/ExtendWith.html [Kasutatud 20.11.2022].
[24] Tarnum Javam, „Getting Started with Mockito @Mock, @Spy, @Captor and @InjectMocks“, Loetud aadressil: https://www.baeldung.com/mockito-annotations [Kasutatud 20.11.2022].
[25] Mockito, Loetud aadressil: https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html [Kasutatud 20.11.2022].
[26] VMware, „Exception Handling in Spring MVC“, Loetud aadressil: https://spring.io/blog/2013/11/01/exception-handling-in-spring-mvc [Kasutatud 20.11.2022].
[27] Tarnum Javam, „Spring RequestMapping“, Loetud aadressil: https://www.baeldung.com/spring-requestmapping [Kasutatud 20.11.2022].
[28] VMware, „Class MockHttpSession“,Loetud aadressil: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/mock/web/MockHttpSession.html [Kasutatud 20.11.2022].
[29] Tarnum Javam, „@Before vs @BeforeClass vs @BeforeEach vs @BeforeAll“, Loetud aadressil: https://www.baeldung.com/junit-before-beforeclass-beforeeach-beforeall [Kasutatud 20.11.2022].
[30] JUnit, „Annotation Type Test“, Loetud aadressil: https://junit.org/junit4/javadoc/4.12/org/junit/Test.html [Kasutatud 20.11.2022
[31] Microsoft, „bcp Utility“, Loetud aadressil: https://learn.microsoft.com/en-us/sql/tools/bcp-utility?view=sql-server-ver16 [Kasutatud 20.11.2022].
[32] Sharpened Productions, „TechTerms – Java“, Loetud aadressil: https://techterms.com/definition/java [Kasutatud 02.01.2023]

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

Mina, Rommi Parman

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Korteriühistu laenutaotluse otsuse automatiseerimine AS-i LHV Pank näitel“, mille juhendaja on Meelis Antoi ning Artjom Pahhomov.
   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.

13.11.2022

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.

Lisa 2 – Eelnevalt täidetud PDF-taotlus

Korteriühistu laenu taotlus

KUUPÄEV

Taotleja

NIMI	REG. KOOD
AADRESS	
ESINDAJA NIMI (JUHATUSE LIIGE)	ESINDAJA ISIKUKOOD
ESINDAJA E-POST	ESINDAJA TELEFON

Info elamu kohta

EHITUSAASTA	EHR KOOD	REKONSTRUEERIMISE AASTA (JUHUL KUI ON TEHTUD)

HOONE KONSTRUKTSIOON

☐ PANEEL ☐ KIVI ☐ PUIT ☐ SEGU/MUU

ELURUUMIDE ÜLDPINDALA	ÄRIRUUMIDE ÜLDPINDALA
KORTERIOMANDITE ARV	KÜ RESERVKAPITAL

Olemasolevad võlakohustused

KELLE EES	KOHUSTUSE JÄÄK	KUUMAKSE	LÕPPTÄHTAEG

Taotletava laenu info

LAENU SIHTOTSTARVE

☐ HOONE RENOVEERIMINE/ REKONSTRUEERIMINE

☐ KINNISTU MUU OBJEKTI RENOVEERIMINE/ REKONSTRUEERIMINE

☐ MUU

TÖÖDE MAKSUMUS	LAENUGRAAFIKU PIKKUS
OMAFINANTSEERINGU SUMMA KOKKU	SOOVITAV MAKSEKUUPÄEV
S.H KREDEXI TOETUSE SUMMA	SOOVITAV MAKSEPUHKUS
LAENUSUMMA	

TAGATIS

☐ VARALISTE ÕIGUSTE PANT

☐ KREDEXI KÄENDUS

OMANIKUJÄRELEVALVE TEOSTAJA

Üldkoosoleku info

KÜ ÜLDKOOSOLEKUL OSALENUD LIIKMETE ARV	LAENU POOLT HÄÄLETANUTE ARV

REMONDI FONDI MAKSE 1M² KOHTA KUUS ENNE RENOVEERIMISOTSUST

REMONDI FONDI MAKSE 1M² KOHTA KUUS PÄRAST RENOVEERIMISOTSUST

Muu

LIIKMETE TASUMATA ARVED (iga võlgnik eraldi real)

KORTERI NUMBER	VÕLASUMMA ÜLE 30 PÄEVA

TÖÖDE TEOSTAJA(D)

ETTEVÕTTE NIMI	REG. KOOD

Juhatuse liikmed

JUHATUSE LIIKMETE KONTAKTID

NIMI	TELEFON	E-POST

Kinnitused

- Kinnitan, et ühistul ei ole viimase kahe aasta jooksul esinenud maksehäireid.
- Kinnitan taotluses esitatud andmete õigsust ning annan oma nõusoleku AS-ile LHV Pank (LHV) taotluses esitatud andmete (sealhulgas isikuandmete) töötlemiseks ja kasutamiseks (sealhulgas edastatud andmete kontrollimiseks ja järelpärimiste tegemiseks avalikest registritest) taotlejale finantseerimispakkumise tegemisel ja käenduse aktsepteerimisel kooskõlas LHV Kliendiandmete Töötlemise Põhimõtetega.
- Kinnitan, et olen saanud kolmandatelt isikutelt, kelle andmeid taotluses avaldatakse, nõusoleku nende andmete (sh isikuandmete) edastamiseks LHV-le ning nad on nõustunud ja teadlikud, et LHV-l on õigus nendega kontakteeruda ja taotluses avaldatud kolmandate isikute andmeid töödelda samadel alustel teiste taotlusel avaldatud andmetega kooskõlas taotluse ja lepingu sõlmimise eesmärgi ning LHV Kliendiandmete Töötlemise Põhimõtetega

TAOTLEJA NIMI JA ALLKIRI