TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Frank Martin Kiibus 186081IADB

Rakendus majandustarkvarade väljundi standardiseerimiseks ja haldamiseks

Bakalaureusetöö

Juhendaja: Kristiina Hakk
PhD

Tallinn 2023

Autorideklaratsioon

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

Autor: Frank Martin Kiibus

05.01.2023

Annotatsioon

Majandustarkvarad on üldiselt mõeldud andmete haldamiseks ning enamasti ei ole võimalik näha visuaalset ülevaadet sellest, mis toimub. Lisaks kui firma on globaalse mastaabiga, omab kontoreid ja haldusüksuseid erinevates riikides, siis võib juhtuda, et sama firma haldab erinevates riikides oma majandusandmeid erinevates tarkvarades mille vahel puudub ühildumisvõimalus.

Lõputöö eesmärk on luua tarkvaralahendus, mis võtab majanduslikud andmed erinevatest keskkondadest, viib nad samale kujule, ehitab kliendi poolt spetsiifiliselt nõutud raportid ning saadab edasi *PowerBI* keskkonda, kus klient saab neid edasi kasutada graafikute ja muu alusena. Käsitsi andmete viimine samasse formaati on väga ajakulukas, tõenäoliselt tekib vigu ning tuleb iga kord uuesti teha kui andmed uuenevad. Loodav lahendus kaotab ära vajaduse käsitsi andmeid töödelda ning teeb seda veakindlalt soovitud ajalises graafikus.

Lõputöö koosneb teoreetilisest ja praktilisest osas. Teoreetilises osas analüüsitakse probleemi sisu, pannakse paika nõuded ning analüüsitakse erinevaid tehnoloogiaid mille toel rakendus arendada. Praktiline osa on lõputöö raames loodava rakenduse realiseerimine, testimine ning tulemuste kaardistamine.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 32 leheküljel, 6 peatükki, 11 joonist, 3 tabelit.

Abstract

Application for Financial Data Standardizing and Management

Nowadays it is common that companies, and potentially private persons, use economic software to manage their financial data, which is used as a basis for example accounting. Usually economic software is meant only for data management and is limited in what can be done with data in the software, including but not limited to creating different graphs, reports etc. Furthermore if a company has a globaal presence, owns multiple offices and management structures across different countries then it might happen that the same company will most likely use different software to keep track of their financial data, which lack any easy way to integrate between one another.

The goal of this thesis is to create a software solution that can retrieve financial data from different economic softwares and equalize their structure to then build custom reports for a client which can be later sent to *PowerBI* environment. The *PowerBI* environment will then allow the client to further use inputted data for building different graphs etc. Manual input of financial data is very time consuming and the given thesis solution is meant to remove the need to operate on data manually, reducing the chance for errors. Furthermore the software will run automatically on a periodic basis.

The thesis consists of a theoretical and a practical part. The theoretical part will consist of analysis of the problem the thesis tries to resolve, will be used to set forth requirements for the thesis software and contains the analysis of potential technologies that can be used to create said software. The practical part will consist of documenting the creation of the thesis solution, testing and mapping of the results.

The thesis is in Estonian and contains 32 pages of text, 6 chapters, 11 figures, 3 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface* ehk rakenduse programmeerimise liides
Backend	Rakenduse teenusepoolne osa, mis vastutab kogu funktsionaalsuse eest, sealhulgas selle toimimine, töökindlus ning turvalisus
CRUD	*Create Read Update Delete* ehk loo, loe, uuenda ning kustuta
CSRF	*Cross Site Request Forgery* ehk saidiülese taotluse võltsimine
CSV	*Comma-separated values* ehk komaga eraldatud väärtused
Celery	Programm, mis käivitab pikalt kestva koodi tagataustal eraldi protsessina muust rakendusest
ERD	*Entity Relationship Diagram* ehk olemi-suhte diagramm
Frontend	Rakenduse kliendipoolne osa, mis vastutab kliendi kogemuse eest
IDE	*Integrated Development Environment* ehk Integreeritud (koodi) arenduskeskkond
JSON	*JavaScript Object Notation* ehk *JavaScript*i objekti notatsioon
Linux	Operatsioonisüsteem
MVC	*Model-View-Controller* - mudel-vaade kontroller, rakenduse Arhitektuurimuster
PowerBI	*Microsoft*i poolt omatud andmete visualiseerimise ja manipuleerimise platvorm
Ubuntu	*Linux* operatsioonisüsteemi üks versioonidest
VPN	*Virtual Private Network* ehk virtuaalne privaatvõrk
WSGI	*Web Server Gateway Interface* ehk veebiserveri lüüsi liides
XML	*Extensible Markup Language* ehk laiendatav märgistuskeel
XSS	*Cross Site Scripting* ehk saidiülene skriptimine

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Tänapäeval on tavaline, et firmadel, aga ka eraisikutel, on kasutusel majandustarkvara oma andmete haldamiseks, millele toetub kogu raamatupidamine. Tavaliselt on majandustarkvara mõeldud andmete haldamiseks, enamasti ei ole võimalik näha visuaalset ülevaadet sellest mis toimub, sealhulgas erinevaid graafikuid, raporteid jne. Lisaks kui firma on globaalse mastaabiga, omab kontoreid ja haldusüksuseid erinevates riikides, siis võib juhtuda, et sama firma haldab erinevates riikides oma majandusandmeid erinevates tarkvarades, mille vahel puudub ühildumisvõimalus.

Lõputöö eesmärk on luua tarkvaralahendus, mis võtab majanduslikud andmed erinevatest keskkondadest, viib nad samale kujule, ehitab kliendi poolt spetsiifiliselt nõutud raportid ning saadab edasi *PowerBI* keskkonda, kus klient saab neid edasi kasutada graafikute ja muu alusena. Käsitsi andmete viimine samasse formaati on väga ajakulukas, tõenäoliselt tekib vigu ning tuleb iga kord uuesti teha kui andmed uuenevad. Loodav lahendus kaotab ära vajaduse käsitsi andmeid töödelda ning teeb seda veakindlalt soovitud ajalises graafikus.

Lõputöö esimeses osas pannakse paika probleem mida lõputöö autor lahendab ning probleemile tuginedes püstitab valmiva tarkvara lahenduse eesmärgid. Järgmisena analüüsitakse probleemi täpsemalt, et paika panna loodava rakenduse nõuded, ning uuritakse ka võimalike olemasolevaid lahendusi. Edasi analüüsitakse erinevaid tehnoloogiaid mille abil lõputöö rakendust looma hakata ning valitakse nende seast sobivaimad tehnoloogiad. Peale tehnoloogiate valikut antakse ülevaade rakenduse teostamisest ja tuuakse välja loodud rakenduse tulemid koos tuvastatud probleemidega ja võimalike edasiarendustega.

2 Probleemi taust ja projekti eesmärk

Käesolevas peatükis esitatakse lõputöö ülesande püstitus ning kirjeldatakse lõputöö rakenduse seotud tarkvarad.

2.1 Eesmärgi püstitus

Lõputöö käsitleb kahte probleemi, millele otsib lahendust rakenduse tellinud klient. Klient on nelja tütarettevõtte emafirma *Finnlog*, mis haldab oma majandusandmeid kahes erinevas majandustarkvaras: *Merit* ja *Visma*.

Kliendil on vaja vastavatest majandustarkvaradest andmed saada ühtlasele kujule, millest saaksid aru sellega seotud isikud, eelkõige raamatupidajad. Praegu on tehniliselt võimalik ühest majandustarkvarast teise andmeid üle viia, kuid see on tülikas ja nõuab käsitööd, millega saab tegeleda ainult raamatupidaja. Teiseks probleemiks on ülevaate ning analüüsi võimekuse puudus andmetel. Selle lahenduseks tõstatas klient nõude võimaldada ühtlasele kujule viidud andmed edasi saata *PowerBI* keskkonda kus on võimekus justnimelt andmete analüüsimiseks, ülevaadete tegemiseks, aga ka rohkemaks [1].

Siit koorub välja tarkvaralahenduse eesmärk – andmed on vaja ilma inimtegurita ühtlustada samale kujule, mida on võimalik töödelda ning analüüsida, eriti just raamatupidajate poolt. Järgmisena on vaja võimaldada andmed edastada *PowerBI* keskkonda, kus on mugav ja kiire saada andmetes ülevaade, raporteid ehitada, aga ka filtreerida ja valmiskujul faile luua firmasiseseks kasutamiseks.

2.2 Lõputöö arendusega seotud tarkvarad

Antud peatükk kirjeldab lõputöö raames käsitletavaid majandustarkvarasid Merit ja Visma ning ärianalüütika töövahendit PowerBI.Tutvustab nende põhifunktsionaalsusi ja tähtsust rakenduse loomisel.

2.2.1 Merit

Merit asutati 1991.aasta novembris Eestis Kaja ja Andres Kert poolt, kes hakkasid arendama raamatupidamise ja laohalduse tarkvara. Sel ajal polnud Meriti arendajatel aimugi kui edukaks see saab, kuid 31 aastat hiljem on Merit üks populaarsemaid

raamatupidamisprogramme Eestis. Meriti kasutajaid on praeguse seisuga üle 31000. Meriti tarkvara arendajate meeskonda kuulub ca 30 inimest kes igapäevaselt annavad endast parima, et viia Merit lähemale kasvule, automatiseerimisele ning kasutajamugavusele. [2]

Meritis saab hallata ja toimetada järgnevate komponentidega: müügiarved, müügipakkumised, ostuarved, inventar, maksed, pearaamat, kliendid, müüjad, pangakontod, raportid, projektid ja kulukohad.

Lõputöö rakendusega seonduvalt on antud majandustarkvara tähtis, sest kliendi firma kasutab Meriti tarkvara nii Eesti-siseselt raamatupidamisandmete haldamiseks ning nende töötlemiseks, kui ka mitmes teises riigis lisaks Eestile. Järgnevalt on loetletud komponendid, mis on lõputöö rakenduse jaoks kõige tähtsamad:

- Müügi- ja ostuarved – sisaldavad infot erinevate arvetega seotud maksete kohta ning määrab nende jaotuse – mis kulukoha alla vastav arve element kuulub ning millise projektiga see seotud on.

- Pearaamat – sisaldab endas täiendavat infot müügi- ja ostuarvete kohta, kulude jaotust, seob erinevad komponendid kokku ühtseks tervikuks.

- Pangakontod – sisaldab infot erinevate pangakontode kohta, mis toetab ning täiendab müügi- ja ostuarvete ning pearaamatu kannetes olevat informatsiooni.

- Projektid – sisaldab infot projektide kohta, mis firmal on ning aitab erinevaid komponente grupeerida vastavalt projektile.

- Kulukohad – sisaldab infot kulukohtade kohta, millega erinevad müügi- ja ostuarvete või pearaamatu kanded seonduvad ning teisalt omab seost ka pangakontodega.

2.2.2 Visma

Visma on ärimaailmas juhtiv põhitarkvara pakkuja, mille eesmärk on lihtsustada firmade tööd ning aidata neil kasvada ja areneda. Firma on asutatud 1996 aastal Norras Alates 2009. aastast on ta teine kõige populaarsem raamatupidamistarkvara firma Soomes [3]. Lõputöö rakenduse jaoks on antud tarkvara tähtis, sest kliendi üks tütarettevõtetest asub

Soomes ning kasutab seda majandustarkvara oma igapäevases töös raamatupidamisalasteks ülesanneteks. [4]

Vismas saab hallata ja toimetada järgnevate põhiliste komponentidega: kliendid, firmad, ostuarved, müügiarved, raamatupidamine, klientide register, toodete register ja projektid.

Järgnevalt on loetletud komponendid, mis on lõputöö rakenduse jaoks kõige tähtsamad:

- Raamatupidamine – nimetatud erinevalt, kuid sisuliselt on pearaamatu kanded, mis sisaldavad endas infot pangakontode kohta ning seovad tarkvara erinevad komponendid ühtseks tervikuks.

- Projektid – sisaldab infot projektide kohta, mis firmal on ning aitab erinevaid komponente grupeerida vastavalt projektile.

- Ostu- ja müügiarved – sisaldab infot erinevate arvetega seotud maksete kohta ning määrab nende jaotuse – mis kulukoha alla vastav arve element kuulub ning millise projektiga see seotud on.

- Firmad – sisaldab arvete koostamiseks vajalikku täiendavat infot ostu- ja müügiarvete kohta.

2.2.3 PowerBI

PowerBI on aastal 2011 Microsofti poolt loodud andmete visualiseerimise keskkond, mis keskendub põhiliselt äriteabele. Keskkond sisaldab endast erinevaid tarkvara teenuseid, rakendusi ning ühendusallikaid millele toetudes on PowerBIs võimalik tuua omavahel mitteseotud andmeallikatest erinevad andmed kokku ühtlaseks tervikuks, anda neile visuaalne kuju ning võimaldada andmeid interaktiivselt uurida. Andmeallikatena on toetatud erinevad andmebaasid, veebilehed, aga ka struktureeritud failid nagu näiteks *CSV (Comma Separated Values*), *XML (Extensible Markup Language*) või *JSON (JavaScript Object Notation*). Lisaks andmete kokkutoomisele erinevates allikatest on võimalik keskkonnas andmeid edasi töödelda vastavalt ärinõuetele ning nende peale ehitada raporteid, erinevaid töölaudasid jne. [5]
Lõputöö jaoks on PowerBI tähtis, sest see on kliendi poolt välja valitud keskkond, kus klient soovib andmeid edasi kasutada graafikute ja muu alusena..

3 Probleemi analüüs

Lõputöös käsiteldavast probleemist tuleb välja kaks eristatavat osa: esimeseks osaks on andmete sisse laadimine erinevatest allikatest ja ühtlustamine, teine osa on andmete edasi saatmine. Lisaks sellele on kliendi poolt paika pandud kindlad nõuded millele loodav rakendus peab vastama.

3.1 Nõuded lahendusele

Enne lõputöö rakenduse loomist võtsid lõputöö autor ning projekti arhitekt endale ülesande läbi meilivestluste ja telefonikõnede kokku koguda loodava rakenduse nõuded. Arhitekt analüüsis koos lõputöö autoriga nõuded läbi ning peale projekti kinnitamist said nõuded olulisuse järjekorda pandud ja ümber kujundatud vastavalt kas funktsionaalseteks või mittefunktsionaalseteks nõueteks. Viimase sammuna andis lõputöö autor hinnangu, kui kaua aega mingi nõude lahendamine võtab ning milline on selle raskusaste.

3.1.1 Funktsionaalsed nõuded

Funktsionaalsed nõuded kirjeldavad, mida loodav rakendus tegema peab. Sinna alla kuulub äriloogika toimimine, kasutajate autentimine ja muud süsteemsed toimingud. [6] Järgnevalt on loetletud rakenduse funktsionaalsed nõuded:

1. Loodav rakendus peab oskama andmeid lugeda ja töödelda Meriti ja Visma tarkvarade API-dest (*Application Programming Interface*).

2. Sisse laetud andmeid tuleb töödelda, sealhulgas viia läbi lisaarvutusi ning viimase sammuna tuleb andmeväljad teisendada vastavalt kliendi poolt ette antud reeglitele.

3. Sisse laetud ning töödeldud andmed tuleb viia kujule mida on võimalik kliendil arusaadavalt kasutada ja analüüsida, samal ajal aru saades kust andmed pärit on ning millise firmaga andmed on seotud.

4. Ühtlasel kujul olevad andmed tuleb salvestada ja säilitada kujul, mida on võimalik edastada *PowerBI* keskkonda, kus kliendil on võimalik lihtsalt andmeid analüüsida ja oma otstarbeks kasutada ka vähese tehnilise oskuse juures.

5. Rakendus peab töötama vähemalt kord ööpäevas automaatselt.
6. Rakendus peab olema käivitatav manuaalselt kliendi soovi korral.
7. Võimalus tulevikus rakendusele juurde lisada majandustarkvarasid mida rakendus suudab töödelda.

3.1.2 Mittefunktsionaalsed nõuded

Mittefunktsionaalsed nõuded on panevad paika rakenduse käitumise, näiteks kättesaadavus, kasutatavus ja töökindlus [7].

Järgnevalt on loetletud rakenduse mittefunktsionaalsed nõuded:

1. Lähtekood ei ole avalik.
2. Lähtekood on kättesaadav kliendile.
3. Rakendusele tohib olla ligipääs ainult kliendi poolt määratud isikutel ja rakenduse arendajal, lõputöö autoril.
4. Rakendus peab töötama *Linux* (*Ubuntu*) operatsioonisüsteemiga masina peal

3.2 Olemasolevad lahendused

Antud peatükis uurib lõputöö autor, kas leidub sarnaseid olemasolevaid rakendusi, mis täidavad lõputöö raames valmiva rakenduse funktsionaalsust ning analüüsib nende kattuvusi ja erinevusi.

3.2.1 *Scoro* automaatliides

Antud rakendus on varasemalt loodud ettevõtte RedFunction poolt, kus lõputöö autor töötab. Rakendus toetab majandustarkvarade *Scoro*, *Directo*, *Merit* ja *Shopify* andmete töötlemist ning edastamist, küll aga puudub liidestatavus majandustarkvaraga *Visma*. Lisaks liidestuse puudusele *Vismaga* ei too rakendus erinevate majandustarkvarade andmeid kokku ühtseks tervikuks, vaid tõstab nad ümber ühest majandustarkvarast teise ning hoiab neid sünkroonis. [8]

3.2.2 *Syncfy*

Ettevõtte Paybook poolt loodud rakendus, mis on mõeldud ettevõttete finantsandmetega seonduva lihtsustamiseks. Läbi rakenduse saab kokku ühendada andmeid erinevatelt platvormidelt mis toetavad *Open Banking Connect* [9] tehnoloogiat, et saada ühtne tervik erinevatest andmestikest. Sellega vastava rakenduse funktsionaalsus ka piirdub. Rakendusel puudub võimekus kliendi poolt soovitud majandustarkvaradest andmeid saada. [10]

3.2.3 LedgerSync

Ettevõtte InventorSoft poolt loodud rakendus, mis koondab pankadest saadud finantsandmed ühte kohta ning võimaldab neid ühtse tervikuna analüüsida. Lõputöö rakendusega võrreldes jääb puudu võimekus kliendi poolt soovitud majandustarkvaradest andmeid saada ning puudub võimekus andmeid edasi töödelda. [11]

3.3 Tehnoloogiate valik

Antud peatükis uurib lõputöö autor, milline programmeerimiskeel ja raamistik lõputöö rakenduse aluseks võtta. Tehnoloogiate valiku teeb lõputöö autor laial levinud tehnoloogiate seast, millel on suur kasutajaskond [12] ja millega on lõputöö autor varasemalt kokku puutunud, et kiirendada rakenduse arenduskäiku. Lisaks valib lõputöö autor välja sobivaima andmebaasi ning kirjeldab kliendipoolse tehnoloogia seotust teenusepoolse tehnoloogia ning andmebaasiga.

3.3.1 Teenusepoolse tehnoloogia valik

Teenusepoolne tehnoloogia on rakenduse see osa, mida rakendust kasutav klient ei näe, kuid ilma milleta on rakendus tavaline staatiline veebidokument. Teenusepoolne tehnoloogia vastutab andmete hoidmise, töötlemise ning väljastamise eest [13]. Teenusepoolse tehnoloogia valikul tuleb arvestada ennekõike aluseks olevast masinast mille peal rakendus oma tööd tegema hakkab, see võib, olenevalt valikust, piirata võimalike variante, milline see tehnoloogia olla võib. Edasi tuleb vaadelda juba spetsiifilist programmeerimiskeelt ning selle poolt pakutavaid raamistikke, mis kõige paremini sobiksid rakenduse realiseerimiseks. Lisaks on lõputööle on paika pandud nõue, et rakendus hakkab töötama *Linux* (*Ubuntu*) operatsioonisüsteemiga masina peal, mis tähendab, et võimalikke programmeerimiskeelte hulka kuuluvad:

- Python – algajasõbralik, suure kasutajaskonnaga interpreteeritud ja objektorienteeritud programmeerimiskeel. Väga paindlik, võimalik täiendada läbi teiste programmeerimiskeelte ja on kergelt skaleeritav. Miinuseks on aeglus, sest pole kompileeritud keel, võib palju mälu võtta ning töökeskkonna esialgne seadistamine võib osutuda keeruliseks. [14]

- Java – kompileeritud ja objektorienteeritud programmeerimiskeel. Eeliseks on laialdane kasutatavus, kerge rakendusi luua ja hooldada, stabiilne ning kasutab vähe mälu. Miinusteks on suur ruumi kasutamine, litsentsi vajamine ning kasutab välja venivat koodi struktuuri mis ei pruugi nii hästi loetav olla. [15]

- C# - kompileeritud ja objektorienteeritud programmeerimiskeel. Plussideks on väga põhjalik dokumentatsioon, suur kasutajaskond ning see, et keel on staatiliselt tüübitud, see tähendab, et kui on paika pandud, et miski on number, siis numbriks ta ka jääb. Lisaks kuulub C keelte perekonda ja seetõttu on sealt edasi kergem õppida teisi C keeli (C, C++). C# kõrgetasemeline keel tänu millele ei pea põhjalikult sügavuti aru saama kuidas erinevad masinad töötavad. Negatiivne on pikk õppimiskõver, toetub tugevalt *.NET* platvormile ilma milleta on suur osa funktsionaalsusest puudu. [16]

- JavaScript – kompileeritud programmeerimiskeel. Eelisteks on kiirus, algajasõbralikus ja ühilduvus teiste keeltega. Miinusteks on turvalisuse vähesus, ühilduvus vanemate versioonidega. [17]

Kõik loetletud programmeerimiskeeled on dokumenteeritud ning on praeguseks pikaajaliselt kasutuses olnud, tehes lahenduste leidmise olemasolevatele probleemidele lihtsamaks. Järgnevalt on välja toodud teenusepoolsete keelte võrdlus kus võrreldakse lõputöö autori kogemust ja programmeerimiskeele õppimiskeerukust ning kiirust (Tabel 1).

Tabel 1. Programmeerimiskeelte võrdlus.

Keel	**Kogemus**	**Õppimiskeerukus**	**Kiirus**
Python	Väga hea	Madal [18]	Aeglane [19]
Java	Rahuldav	Keskmine [20]	Kiire [21]
C#	Hea	Madal [22]	Kiire [23]
JavaScript	Rahuldav	Madal [24]	Kiire [25]

Arvestades autori varasemat kogemust erinevate programmeerimiskeeltega, selgub, et lõputöös kõige kiiremini rakendatavad keeled on Python ja C# . Mõlemad keeled on lisaks ka madala õppimiskeerukusega, kuid kiiruse poolest sobib paremini C#. Java ja JavaScript on kiiruse poolest head, kuid lõputöö autoril jääb nendes kogemust puudu. Lisaks on Java õppimiskeerukus kõrgem kui teistel keeltel. Enne lõpliku valiku tegemist võrdleb lõputöö autor ka eelnevalt välja toodud programmeerimiskeelte raamistike.

Pythoni raamistikud on näiteks:

- Flask – veebiraamistik, mis lahendab kasutaja eest ära madalatasemelised nõuded nagu näiteks protokolli valik ja lõime haldamine. Raamistik on kergelt õpitav ning võimaldab koheselt asuda arendama [26].

- Django – Kõrgetasemeline veebiraamistik, mis toetab kiiret, pragmaatilist ja puhast disaini. Ehitatud arendajate poolt arendajatele. Töötab väga kiirelt, on turvaline ning väga kergelt skaleeritav [27].

Java raamistik on näiteks:

- Spring – raamistik millel on oma testimise süsteem, Java sarnaste keelte toetus, keeletugi ja veel hüvesid. On mõeldud arendajatelt võtma kõik kõrvalise ära, et saaks keskenduda puhtalt äriloogiliste probleemide lahendamisele [28].

C# raamistikud on näiteks:

- .NET – avatud lähtekoodiga platvorm, mis toetub *MVC*-le (*Model-View-Controller*) ja millega saab erinevaid rakendusi luua. Toetab erinevaid programmeerimisekeeli lisaks C#-le ja on kasutatav igal operatsioonisüsteemil. [29].

JavaScript mõned tuntumad raamistikud on:

- Vue – raamistik, mis on mõeldud peamiselt esirakenduste tegemiseks ja kasutajale meeldiva disaini ning kasutuskogemuse tagamiseks. [30].

- React – Sarnaselt Vue-le on mõeldud rohkem esirakenduste tegemiseks ja on otsene konkurentsi pakkuja Vue-le [31].

Sarnaselt programmerimiskeeltele on samuti kõik raamistikud korralikud dokumenteeritud ja omavad suurt kasutajaskonda kellele saab toetuda probleemide lahendamisel ja ideede kogumisel. Tabel 2 võrdleb erinevaid raamistike kasutades võrdluspunktidena lõputöö autori kogemust, raamistiku õppimiskeerukust, paindlikust ning turvalisust.

Tabel 2. Raamistike võrdlus.

Raamistik	**Kogemus**	**Õppimiskeerukus**	**Paindlikus**	**Turvalisus**
Flask	Halb	Madal [32]	Kõrge [32]	Puudub [32]
Django	Väga hea	Keskmine [32]	Keskmine [32]	Kõrge [32]
Spring	Puudub	Keskmine [33]	Keskmine [34]	Kõrge [35]
.NET	Hea	Keskmine [36]	Keskmine [37]	Kõrge [38]
Vue	Halb	Madal [39]	Kõrge [39]	Keskmine [40]
React	Puudub	Madal [39]	Kõrge [39]	Kõrge [41]

Raamistike võrdlusest paistab välja, et JavaScript [42] raamistike vaatepunktist sobib ainult esirakenduse loomiseks, kuid antud juhul on soov leida raamistik tagarakenduse jaoks. Seega jäävad valikusse Python, C# ja Java raamistikud. Python, C# ja Java vahel võrreldes on kõige kiirem arendada Pythonis, kus suure osa koodist saab kompaktselt

kirjutada, samal ajal kui C# ja Java vajab eraldi struktuurile mõtlemist ning seadistamist. Tänu sellele on Pythonit kergem arendada nii üksi kui meeskonnaga . C# ja Java on oma kiiruse ja mälu kasutuse poolest paremad kui Python. Võttes arvesse lõputöö nõudeid ning vajadusi, siis parem kiirus ning mälu kasutus ei kaalu üle kasutatavust ning koodi loetavust. Raamistike hulgast on kõige suuremate eelistega Pythoni raamistik Django millel on rohkem eeliseid ning vähem miinuseid kui teiste keelte raamistikel. C# raamistik .NET oleks ka valikus, kuid olemasolev rakendus on planeeritud tööle panna *Linux* operatsioonisüsteemiga masinas ning .NET ei ühildu alati kõige paremini *Linux*-iga. Java raamistiku Spring puhul aga saab takistavaks asjaoluks keerukus ning XML-ide kasutamise vajadus. Sellest tulenevalt otsustas lõputöö autor kasutusele võtta Python programmeerimiskeele koos Django raamistikuga.

3.3.2 Kliendipoolse tehnoloogia valik

Kliendipoolse tehnoloogia valik on vastavalt kliendi soovidele *PowerBI* keskkond, kuhu jõuavad välja rakenduse poolt valmis saadud tulemid. *PowerBI* on platvorm kuhu saab erinevatest allikatest andmed kokku koondada, nende peale ehitada erinevaid andmemudeleid, graafe, jooniseid jne. Peale seda kui andmete peale on ehitatud erinevad visualiseeringud, saab need üles laadida ja kättesaadavaks teha kõigile, kellele sellistele andmetele ligipääs vajalik. Andmete peale ehitatut saab klient edasi analüüsida, et teada saada peamised kulud, tulud, aga ka erinevate ettevõtmiste kasumlikkuse või kahjumlikkuse, et nii vastavale infole tuginedes edasi liikudes teadlikuid otsuseid vastu võtta. Peale firma enda vaatepunkti saavad erinevad töötajad parema ülevaate oma tegevustest, et saavutada parem ülevaade toimuvast ja vastavalt oma tegevusi korrigeerida.

3.3.3 Andmebaasi valik

Andmebaase on erinevat tüüpi, näiteks relatsioonilised, hierarhilised või objektorienteeritud. Veebirakenduste puhul on ennekõike kasutusel relatsioonilised andmebaasid, mis tähendab, et antud lõputöö raames uuritakse just neid. Andmebaasi valiku tegemisel võrdleb lõputöö autor praegu kolme kõige populaarsemat relatsioonilist andmebaasi. [43]

Need andmebaasi valikud on:

- SQLite – Iseseisev, null konfiguratsiooniga relatsiooniline transaktsiooniline andmebaas, mis baseerub ühel ainsal failil kõvakettal. On üleplatvormilise toega, väikese mahuga ning järjepidevalt läbi katsetatud [44].

- MySQL – Üks tuntumaid ja rohkem kasutatavaimaid relatsioonilisi andmebaase maailmas, avatud lähtekoodiga ning kiire. Toetab erinevaid tüüpe rakendusi ning on suure kasutajaskonnaga [45].

- PostgreSQL – Tasuta ja vabavaraline relatsiooniline andmebaas, millega tuleb kaasa palju lisasid. Pidevalt läbi testitud ning tugeva kasutajaskonnaga, mis on turvaline, hästi skaleeritav ja on keeletoega [46].

Tabel 3 võrdleb autor erinevaid andmebaase kasutades lõputöö autori kogemust, andmebaasi turvalisust ning ühilduvust PowerBI keskkonnaga.

Tabel 3. Andmebaaside võrdlus

Andmebaas	**Kogemus**	**Turvalisus**	**Ühilduvus *PowerBI*-ga**
SQLite	Väga hea	Madal [47]	Ei [48]
PostgreSQL	Väga hea	Kõrge [49]	Jah [48]
MySQL	Hea	Kõrge [50]	Jah [48]

Kogemuse poolest sobivad kõik andmebaasid, kuid võrdlusest jääb hästi silma SQLite madal turvalisuse tase, mis välistab selle kasutatavuse rakenduse andmebaasina, sest rakenduse keskne osa, mis kõik omavahel kokku seab, on just andmebaas ja seetõttu ei saa see olla madala turvalisusega. Alles jäänud valikutest on PostgreSQL ja MySQL mõlemad kõrge turvalisuse tasemega, kuid ainult MySQL on võimeline *PowerBI* platvormiga ühilduma ilma, et kliendi poolel peaks seadistamise tegema IT valdkonnaga tegelev inimene, mistõttu antud lõputöö rakenduse andmebaasiks saab valitud MySQL.

3.4 Arenduskeskkonna valik

Arenduskeskkonna alla kuulub koodihaldus, koodihoidla ning koodi arenduskeskkond koos erinevate abivahenditega.

3.4.1 Koodihaldus

Koodihalduse all on mõeldud koodi muudatuste jälgimist lähtekoodi hoidlas, mis võib olla arendajal endal masinas, aga ka pilves. Koodihaldus on mõeldud ajaloo talletamiseks ning aitab erinevate versioonide vahel konflikte lahendada. Kui näiteks erinevad kaastöötajad muudavad sama koodi, siis nende muudatuste vahel tekib konflikt, mida koodihaldus suudab tuvastada ja oskab aidata selle lahendamisel, et kas mõlemad muudatused jääksid peale, mingi osa muudatustest, või siis mitte midagi kui muudatused üldse ei sobi. Veel aitab koodihaldus vältida koodi kadumist, sest kõik muudatused on ajaloos salvestatud ning nende juurde saab igal ajal tagasi minna. [51]

3.4.2 Koodihoidla

Koodihaldus on tihedalt seotud koodihoidlaga, sest kõik koodi muudatused jõuavad lõpuks koodihoidlasse, kust hiljem saab need muudatused uuesti kätte.

Järgnevalt on välja toodud kolm tuntumat koodihoidlat millega on lõputöö autoril kokkupuude olemas ning mida saab kasutada lõputöö raames koodi haldamiseks:

GitHub – Kõige suurem avatud lähtekoodi hoidlatega koodihoidla, on väga suure kasutajaskonnaga, omab meeskonna administreerimise võimalusi koos projekti halduse vahenditega. On tasuta võimalusega, lisa funktsionaalsust saada meeskonna või ettevõtte tasemel. [52]

BitBucket – Meeskondadele keskendunud koodihoidla millel on integratsioon ettevõtte teise tootega: Jira, mis on üks parimaid keskkondasid projekti haldamiseks ja planeerimiseks [53]. Koodihoidla keskendub rohkem agiilsele arendusele ja on vastavalt ülesehitatud. On tasuta võimalusega, lisa funktsionaalsust saada meeskonna või ettevõtte tasemel. [52]

GitLab – Kõige paremini sobilik infrastruktuuriga seonduva hoidmiseks, kuid sobib ka arenduseks. Põhifunktsionaalsused on sarnased GitHub-iga ning nagu GitHub on olemas erievad paketid, mille vahel valida. [52]

Koodihoidlatest jääb kõige tugevamini silma BitBucket, mis toetab tugevalt projekti haldamist ja planeerimist, funktsionaalsused mida lõputöö autor plaanib efektiivselt ära kasutada. BitBucket on hea valik meeskondadele ja selline struktuur on tavaliselt ka ettevõttes – teatud arv inimesi tegeleb ühe projekti kallal. Lisaks sellele on BitBucket

valikute seast ainus, millel on vastav sertifikaat, mis lubab nende poolt, et lähtekood pole kättesaadav kõrvalistele isikutele. [54]

3.4.3 Koodi arenduskeskkond ja abivahendid

Koodi arenduskeskkonna all on tänapäeval peamiselt mõeldud *IDE*-sid (*Integrated Development Environment*) läbi mille toimub suurem osa rakenduse arendamisest. *IDE* on tarkvara, mis seob tihedalt kasutatavad arendusvahendid ühtseks tervikuks läbi graafilise kasutajaliidese. *IDE* peamisteks osadeks on: koodi redaktor, silur ja koodi kompileerimine ja jooksutamine.Lisaks peamistele osadele võivad *IDE*-d sisaldada lisavahendeid olenevalt tarkvarast, seda siis kas vastava ettevõtte enda poolt või kasutajaskonna poolt. [55]

IDE lihtsustab tugevalt arendaja tööd ning võimaldab kiirelt rakendusi arendada, testida ja siluda. Välja valitud programmerimiskeele Python jaoks on järgnevalt välja toodud parimad *IDE* valikud [56]:

IDLE – Python programmerimiskeelega kaasaskäiv arendamiskeskkond, mis toetab põhifunktsionaalsusi nagu näiteks käsurea kasutus, failidest otsimine, konfigureerimine ja silumine. Puudu jääb võimalusest läbi graafilise liidese koodihaldusega tegeleda ning seda tuleb teha läbi käsurea. Töötab erinevatel operatsioonisüsteemidel samamoodi. Programm on tasuta. [57]

PyCharm – Pakub lünkadesse vastavas projektis olevaid variante kiireks arenduseks, töötab erinevatel operatsioonisüsteemidel, hea koodi navigeeritavus, vigade ning hoiatuste tuvastamine, kirjavigade tuvastus, aitab kirjutada koodi vastavalt ettenähtud stiilistandarditele, toetab kiiret koodi refaktoreerimist ja teisaldamist kui üks koodi osa paisub liiga suureks, võimalik arendada läbi interneti ühenduse teisest masinast, otsene integratsioon Django ja teiste Python-i raamistikega, tööriistad teaduslike eksperimentide jaoks, andmebaasi haldamise ja koodihoidla tugi otse läbi graafilise liidese. Programm on tasuta, kuid saab lisafunktsionaalsuseid litsentsiga. [58]

Visual Studio Code – Töötab erinevatel operatsioonisüsteemidel, head silumise vahendid, koodi välja pakkumine vastavalt eelnevalt kirjutatule, failide otsing ja erinevate koodi osade välja toomine nähtavalt, toetab koodihaldust otse läbi graafilise liidese. Programm on tasuta. [59]

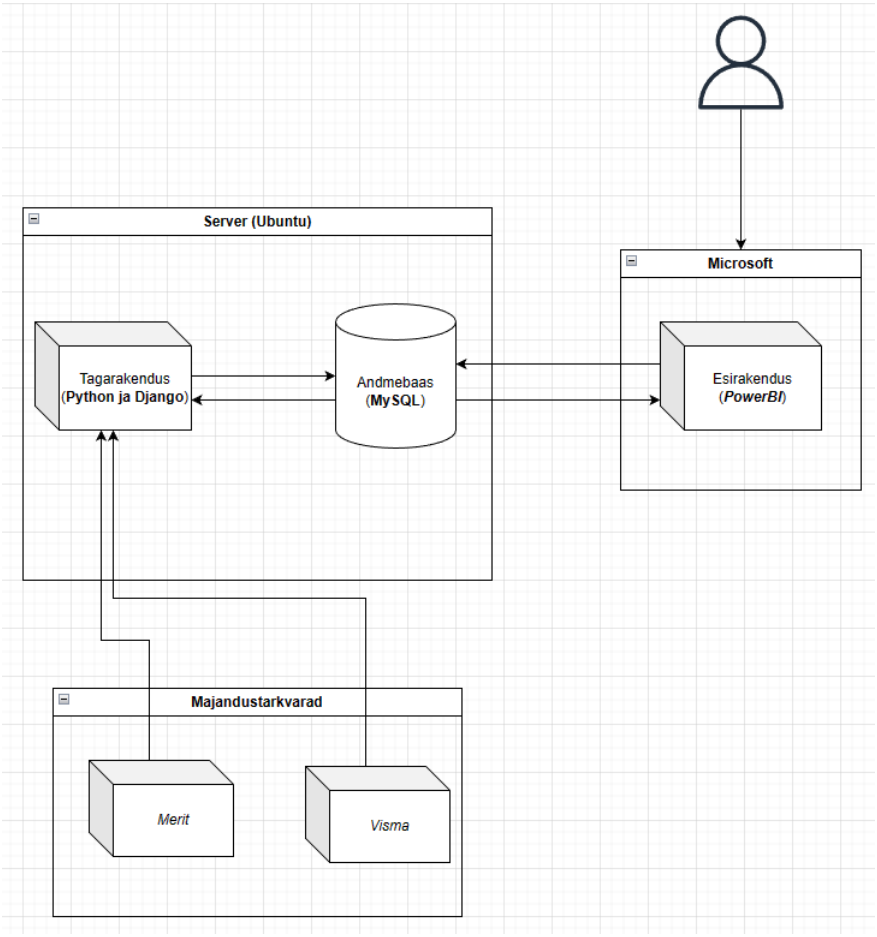
Eelnevalt kirjeldatud valikutest paistab esimesena silma PyCharm oma paljude funktsionaalsustega ning võimalustega. Kõik keskkonnad toetavad koodi kirjutamist ja failide navigeerimist, teksti redigeerimist, kuid PyCharm toetab veel erinevate Python raamistike kasutust, mis kiirendab ja abistab suurelt arendaja tööd. Koodihaldust ei saa teha ainult IDLE-s, kuid see on tänapäeva arenduse tähtis osa ning seetõttu antud lõputöö raames see ei sobi. PyCharm ja Visual Studio Code toetavad mõlemad koodihaldust, kuid lisaks sellele toetab PyCharm veel andmebaaside haldust – ei ole vaja eraldi rakendust läbi mille andmebaasiga ühendus luua ning seal toimetada.

4 Tulemuste realiseerimine

Antud peatükis kirjeldatakse rakenduse ülesehitust ja väljavalituid tehnoloogiate kasutatamist rakenduse realiseerimisel.

4.1 Ülesehitus

Rakendus töötab *Linux* (*Ubuntu*) operatsioonisüsteemiga masina peal, kuhu lõputöö autor paigaldab kõik vajaliku rakenduse funktsioneerimiseks. Rakenduse teenusepoolseks tehnoloogiaks on Python programmeerimiskeel koos Django raamistikuga. Kliendipoolseks tehnoloogiaks on kliendi poolt väljavalitud platvorm *PowerBI* ja andmebaasiks on MySQL, mis vastutab selle eest, et teenusepoolse tehnoloogia tulem jõuaks kliendipoolsesse tehnoloogiasse. Joonis 1 illustreerib milline näeb välja lõputöö rakenduse üldine arhitektuur.



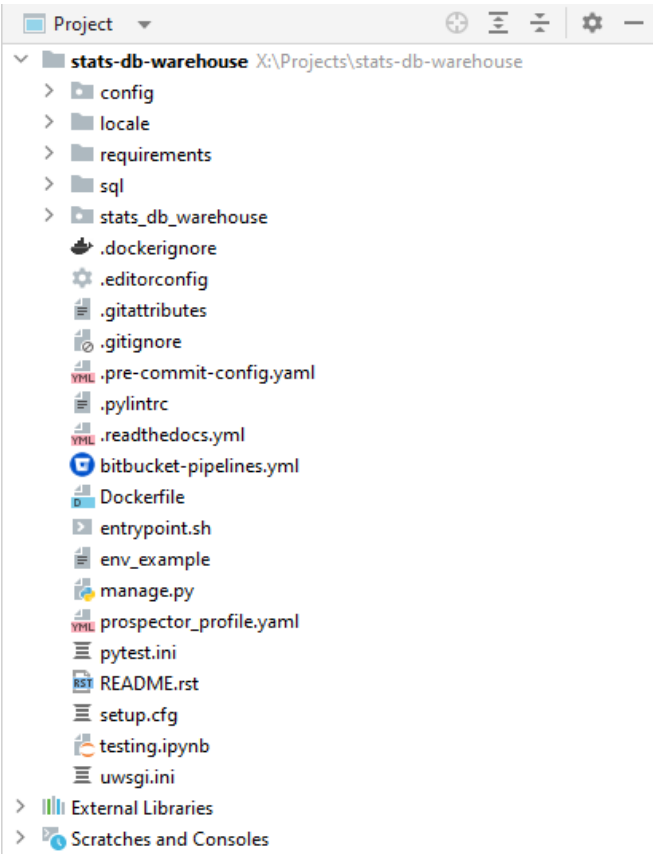
Joonis 1. Üldine arhitektuuri joonis.

4.1.1 Koodihoidla

Lõputöö rakenduse projekt saab alguse koodihoidlast BitBucket-is, kus luuakse uus hoidla rakenduse jaoks, mille nimi antud juhul on hoidlas *stats-db-warehouse*. Siit edasi toimub hoidla allalaadimine lokaalsesse arenduskeskkonda.

4.1.2 Projekti struktuur

Lõputöö rakenduse projekt on seadistatud võttes arvesse Python programmeerimiskeele ning kasutatava Django raamistiku häid tavasid. Programmeerimiskeel määrab ära, kuidas erinevate osade nimetamine käib, kuidas põhifunktsionaalsused töötavad ning paneb paika stiilinõuded, millest on mõistlik kinni hoida, et rakendus oleks loetav ja kergemini hallatav. Django raamistik paneb paika projekti arhitektuurilise struktuuri, kuidas erinevad rakenduse osad paiknevad teineteise suhtes – näiteks erinevates failides või kaustades. Joonis 2 illustreerib projekti kõige pealmist struktuuri, nii nagu seda on näha välja valitud PyCharm arenduskeskkonnas.



Joonis 2. Projekti pealmine struktuur.

Joonise pealt ülevalt alla liikudes näeme esimesena *config/* kausta, kus asuvad raamistiku jaoks järgnevad seadistused:

- *WSGI (Web Server Gateway Interface*) algseadistus, mis on Python-i standard, kuidas veebiserver suhtleb veebirakendustega ja kuidas erinevaid veebirakendusi saab päringute täitmiseks kasutada. [60]

- Rakenduse erinevate URL-ide seadistus, kuhu antud rakenduse puhul kuulub graafiline administratiivne liides, kust saab otse andmebaasis olevate andmetega teha *CRUD (Create Read Update Delete*) päringuid.

- Raamistiku jaoks vajalikud sätted, mis on jagatud kolme erineva faili vahel – lokaalseks arenduseks vajalikud sätted, arenduse keskkonna sätted ning toodangu keskkonna sätted.

Järgmisena näeme *locale/* kausta, mis sisaldab rakenduse poolt kasutatavaid tõlkeid. Peale seda tuleb *requirements/* kaust, mis sisaldab kolme erinevat nõuete faili, kus on lokaalse keskkonna, arenduse keskkonna ning toodangu nõuded. Need nõuded on erinevad Python teegid, mis on vajalikud rakenduse normaalseks tööks ning need teegid on versioonide täpsusega paika pandud vältimaks murrangulisi muudatusi ja kättesaadavad peamiselt Pythoni enda teekide keskkonnast *PyPi*, kuid vajadusel saab ette anda ka muu koha, kust teeke pärida. Edasi on *sql/* kaust, kus on kokku kirjutatud SQL päringud, mille põhjal on võimalik luua ostu- ja müügiraport otse andmebaasi. Kaustadest viimasena on projekti nimeline kaust *stats-db-warehouse/* kus asub kogu projekti tuum, mis vastutab andmete struktuuri, andmete pärimise ja kõige muu eest.

Veel tasub eraldi ära mainida *manage.py* fail, mis on Django raamistiku puhul sisenemispunkt koodi ning millega saab samm-sammu haaval läbi käia erinevad etapid, mida rakendus peab suutma teha, ning *bitbucket-pipelines.yaml*-i kuhu on kirjeldatud sammud, mida tuleb teha kui lokaalsed muudatused ära salvestada ja üles lükata koodihoidlasse. Koodihoidlas toimub peale igat muudatuse ülesse lükkamist stiilikontroll ning testide läbimine, mis aitab takistada vigase koodi jõudmist aktiivsesse keskkonda.

4.1.3 Andmebaas

Andmebaas on antud lõputöö rakenduses siduvaks osaks teenusepoolse ja kliendipoolse tehnoloogia vahel. Teenusepoolne tehnoloogia teeb äriloogilised toimetused ära ning

salvestab kõik tulemused andmebaasi. Edasi võtab andmebaasiga ühendust kliendipoolne tehnoloogia ning loeb kõik andmed otse andmebaasist.

Andmebaasi tekivad tabelid läbi raamistiku, käivitades käsu **`python3 manage.py migrate`**, mis otsib üles projektis leiduvad migratsioonifailid ning nende põhjal loob või muudab tabeleid vastavalt migratsioonifailides olevale sisule.

Esimesel migreerimisel loob raamistik lisaks rakenduse spetsiifilistele migratsioonidele ka raamistiku enda jaoks vajalikud tabelid, mis tagab raamistiku põhifunktsionaalsuste toimimise. Andmebaasi migratsioonifailid tekivad rakenduses, kui käivitada käsk **`python3 manage.py makemigrations`**, mille käigus otsib raamistik üles talle ette öeldud rakenduse alamosad, kust ta otsib faili nimega **models.py**. Igas **models.py** failis on raamistiku reeglite järgi ja vahenditega loodud Python objektid, millel on kirjeldatud väljad, mida vastav objekt peab sisaldama. Raamistik oskab need objektid teha migratsiooniks, mille tulemusel tekib andmebaasi objekti põhjal tehtud tabel.

Edasiste muudatuste käigus, kui migratsioonifaile looma hakata, tuvastab raamistik kõik objektidega seotud muudatused ning oskab tuvastada näiteks, kas lisandus väljasid juurde, midagi muutus või kustutati äkki objekt sootuks ära. Lisaks automaatsetele migratsioonidele võib ka käsitsi luua migratsioonifaile, seda näiteks juhul, kui migreerida andmeid või eriloogikat. Järgnevalt välja toodud joonisel 3 on näide, milline näeb välja mudeli põhjal genereeritud migratsioonifail Visma majandustarkvara komponendi näitel.

```
import uuid
from django.db import migrations, models
import django.db.models.deletion


class Migration(migrations.Migration):

    initial = True

    dependencies = []

    operations = [
        migrations.CreateModel(
            name='Journal',
            fields=[
                ('Id', models.UUIDField(db_column='id', default=uuid.uuid4,
editable=False, primary_key=True, serialize=False, unique=True)),
                ('EntryId', models.CharField(db_column='entry_id',
max_length=20, null=True)),
                ('JournalNo', models.TextField(blank=True,
db_column='journal_no', null=True)),
                ('JournalName', models.TextField(blank=True,
db_column='journal_name', null=True)),
                ('JournalEntryNo', models.TextField(blank=True,
db_column='journal_entry_no', null=True)),
                ('JournalDate', models.DateField(blank=True,
db_column='journal_date', null=True)),
                ('AccountingPeriod', models.TextField(blank=True,
db_column='accounting_period', null=True)),
                ('AccountNo', models.TextField(blank=True,
db_column='account_no', null=True)),
                ('Amount', models.DecimalField(db_column='amount',
decimal_places=4, max_digits=10, null=True)),
                ('TaxAmount', models.DecimalField(db_column='tax_amount',
decimal_places=4, max_digits=10, null=True)),
                ('Description', models.TextField(blank=True,
db_column='description', null=True)),
                ('InvoiceNo', models.TextField(blank=True,
db_column='invoice_no', null=True)),
                ('InvoiceDate', models.DateField(blank=True,
db_column='invoice_date', null=True)),
                ('DueDate', models.DateField(blank=True,
db_column='due_date', null=True)),
                ('CompanyId', models.TextField(blank=True,
db_column='company_id', null=True)),
                ('CompanyName', models.TextField(blank=True,
db_column='company_name', null=True)),
                ('Sk1', models.TextField(blank=True, db_column='sk1',
null=True)),
                ('Sk2', models.TextField(blank=True, db_column='sk2',
null=True)),
                ('Sk3', models.TextField(blank=True, db_column='sk3',
null=True)),
                ('Sk4', models.TextField(blank=True, db_column='sk4',
null=True))])
```

Joonis 3. Automaatselt genereeritud migratsioonifail.

Eelnevalt välja toodud joonisest on näha kuidas iga andmebaasi mineva tabeli veerg on eraldi klassi objekt, millel on oma kuju ning struktuur ning mis vastab kindlale andmetüübile andmebaasis. Järgmisena välja toodud joonisel 4 on näide, kuidas migratsioonifaili saab ise sisu kirjutada.

```
from django.db import migrations


class Migration(migrations.Migration):

    dependencies = [
        ('core', '0013_reportaccount_exclude_from_consolidated'),
    ]

    operations = [
        # Enable on update cascade for company model
        # core_company
        migrations.RunSQL(
            sql='alter table `core_company` drop foreign key
`core_company_api_b7a75a70_fk_core_api_api`;'
                'alter table `core_company` add constraint
`core_company_api_b7a75a70_fk_core_api_api` foreign key (`api`) references
`core_api` (`api`) on delete cascade on update cascade;',
            reverse_sql='alter table `core_company` drop foreign key
`core_company_api_b7a75a70_fk_core_api_api`;'
                        'alter table `core_company` add constraint
`core_company_api_b7a75a70_fk_core_api_api` foreign key (`api`) references
`core_api` (`api`) on delete cascade;'
        )
```

Joonis 4. Käsitsi loodud migratsioonifail.

Raamistiku migratsioonifailide sisu säilitab ajaloolised muudatused andmebaasi tabelitele ning tänu sellele on võimalik iga hetk muudatusi tagasi lükata, näiteks kui midagi läks valesti. Migratsioonifailid sisaldavad lisaks viiteid teistele rakenduse alamosadele, juhul kui erinevate alamosade objektid on omavahel seotud, siis oskab raamistik enne luua seoses olevad objektid õiges järjekorras ning siduda nad omavahel. Selliselt on tagatud, et ei loodaks andmebaasi seost tabeliga mida päriselt veel olemas ei ole.

Järgmisena välja toodud joonisel 5 on näha viidet teisele migratsioonile sõltuvuste (*dependencies)* koodi osas, mis peab enne migreeritud olema kui saab asuda praegust migratsiooni läbi viima.

```
from django.db import migrations, models
import django.db.models.deletion


class Migration(migrations.Migration):

    dependencies = [
        ('core', '0003_api'),
    ]

    operations = [
        migrations.CreateModel(
            name='Company',
            fields=[
                ('id', models.AutoField(db_column='id', primary_key=True,
serialize=False)),
                ('company', models.CharField(db_column='company',
max_length=100, unique=True)),
                ('api', models.ForeignKey(db_column='api',
on_delete=django.db.models.deletion.CASCADE, related_name='companies',
to='core.api', to_field='api')),
            ],
            options={
                'abstract': False,
            },
        ),
    ]
```

Joonis 5. Tabeli loomine, millel on viide teise tabelisse.

Andmebaasi tabeleid saab grupeerida ning seda on mõistlik teha, et paremini aru saada millised tabelid on seotud millise rakenduse osaga. Siinkohal aitab raamistik palju kaasa, andes igale tabeli nimele ette rakenduse alamosa nime, nii on aru saada, kust vastavad tabelid tulevad.

Lõputöö rakendus sisaldab endas raamistiku poolt genereeritud tabeleid ja projekti jaoks spetsiifilisi tabeleid. Järgnevalt lkirjeldatakse andmebaasi erinevad tabelid vastavalt oma grupeeringule ning selgitatakse nende otstarvet.

- Raamistiku poolt genereeritud tabelid – tabelid ilma milleta ei töötaks suur osa raamistiku funktsionaalsusest. Nende tabelite põhjal töötab kogu kasutajate ning õiguste süsteem, aga ka migratsioonide jälgimine, et oleks teada millised migratsioonid on juba andmebaasis olemas ja millised mitte. Lisaks sellele sisalduvad seal tabelid, mis määravad ära, millise ajalise graafikuga peab rakendus midagi tegema. Vastavate tabelite ERD (*Entity Relationship Diagram)* skeem on välja toodud lisas 2.

- Rakenduse spetsiifilised tabelid – tabelid, mis on rakenduse funktsionaalsuse toimimiseks vajalikud. Nende tabelite seas on tabelid, mis on kaardistatud vastavalt majandustarkvarast tulnud struktuurile, et neid oleks rakenduses mugavam edasi töödelda raamistiku poolt pakutavate vahenditega. *Visma* majandustarkvara kaardistavate tabelite ERD skeem on välja toodud lisas 3 ning *Merit* kaardistavate tabelite ERD skeem on välja toodud lisas 4. Lisaks kaardistavatele tabelitele on funktsionaalsed tabelid, kuhu salvestatakse rakenduse käigus tekkivad vahetulemused ning tabelid mille pealt ehitatakse kokku kliendi poolt nõutud bilansi- ning kasumiraport. Viimasena on andmebaasis raportite tabelid, mida näeb klient ka *PowerBI* platvormil ning need tekivad dünaamiliselt rakenduse töö lõpus. Funktsionaalsete tabelite ERD skeem on välja toodud lisas 5.

4.1.4 Teenusepoolne tehnoloogia

Lõputöö rakenduse teenusepoolne osa on loodud ettevalmistatud malli peale, kus esmased Django raamistiku seadistused on juba ära tehtud, võimaldamaks kohe arendama asuda.

Esimese sammuna luuakse majandustarkvaradele *Visma* ja *Merit* raamistiku terminites mudel, kuhu hakkavad salvestuma sisselaetud andmed. Selle jaoks on kaardistatud mõlema majandustarkvara andmestruktuurid ning taasloodud rakenduses endas, et esmalt ei pea kohe andmeid töötlema ega ühtlustama hakkama.

Järgmine samm on leida, kuidas mõlema majandustarkvara andmed kokku sobituksid ning selle jaoks loob lõputöö autor abistavad mudelid, kuhu mõlema majandustarkvara vaatest kaardistab elemendid abistavatesse mudelitesse. Järgnevalt välja toodud joonisel 6 on näide sellest, milline näeb välja *Merit* majandustarkvara pearaamatu komponent rakenduse koodis.

```
class GeneralLedger(AbstractBase):
    batch_code = models.TextField(blank=False, null=False,
db_column='batch_code')
    no = models.TextField(blank=True, null=True, db_column='no')
    document = models.TextField(blank=True, null=True, db_column='document')
    batch_date = models.DateField(blank=True, null=True,
db_column='batch_date')
    currency_code = models.TextField(blank=False, null=False, max_length=3,
db_column='currency_code')
    currency_rate = models.DecimalField(blank=False, null=False,
max_digits=65, decimal_places=9,
                                        db_column='currency_rate')
    total_amount = models.DecimalField(blank=False, null=False,
max_digits=65, decimal_places=9,
                                       db_column='total_amount')
    price_incl_vat = models.TextField(null=False, blank=False,
db_column='price_incl_vat')

    @classmethod
    def create_model(cls, data: dict):
        """
        Create a class instance from given data

        :param data: data in dict format
        :return: Class Instance
        """
        return cls(
            id=data.get('GLBId', uuid4()),
            batch_code=data.get('BatchCode', None),
            no=data.get('No', None),
            document=data.get('Document', None),
            batch_date=data.get('BatchDate', None),
            currency_code=data.get('CurrencyCode', None),
            currency_rate=data.get('CurrencyRate', None),
            total_amount=data.get('TotalAmount', None),
            price_incl_vat=data.get('PriceInclVat', None),
            identifier=data.get('Identifier', None)
        )

    def __str__(self):
        return f'General Ledger ID: {str(self.id)}'

    def __repr__(self):
        return f'Id: {str(self.id)}'
```

Joonis 6. Merit majandustarkvara pearaamatu komponent rakenduse koodis.

Nüüd kui andmed on ühtlustatud saab valmistuda kliendi poolt soovitud raportite loomiseks. Ostu- ja müügiraporti saab kohe valmis teha, sest need ei vaja lisaarvutusi, mistõttu otsustas lõputöö autor selle kokku panna otse SQL päringuga, mis on salvestatud koodihoidlasse ja läheb automaatselt rakenduse töö käigus tööle. Kõigepealt kustutab SQL päringvana andmebaasi tabeli ära ning seejärel loob uue, kuhu sisse ühendab mõlema majandustarkvara andmed kokku. Järgmisena saab asuda bilansi raportit looma, kuid esmalt on vaja välja arvutada majandustarkvarade andmete pealt kaks lisavälja: bilansiaasta kasum ning eelmiste bilansiaastate kogunenud kasum. Peale lisaväljade

arvutamist tühjendatakse andmebaasis olev bilansiraporti tabel ning täidetakse tänase päeva seisuga mille koodi on näha lisas 6. Peale bilansi raportit on veel alles jäänud kasumiraport. Selle jaoks tuleb kõik pangakontodega seotud andmed ükshaaval üle käia ning leida nende kontode muudatused kuu täpsusega. Kui see on tehtud, siis edasi sarnaselt bilansi raportile tühjendatakse esmalt praegune andmebaasis olev kasumiraporti tabel ära ning täidetakse tänase päeva seisuga olevate andmetega ning mille koodi on näha lisas 7.

Peale kõikide raportite valmis saamist sooritatakse kontroll, et kõik tabelid oleksid uuenenud nelja päevase puhvriga, see tähendab et kui neljal päeva pole muudatusi toiminud, siis on midagi valesti ning tuleb asuda parandama. Selle jaoks saadab rakendus meili lõputöö autorile kiire tähelepanu andmiseks. Kui kontroll on edukalt läbitud, kopeerib rakendus andmed oma andmebaasist kliendi jaoks tehtud eraldi andmebaasi, kus leiduvad vaid kliendi jaoks mõeldud raportid. Peale raportite tabelite kopeerimist on rakenduse töö tehtud ning tsükkel kordub uuesti kliendi poolt ette määratud perioodilisusega, milleks praegu on kord ööpäevas.

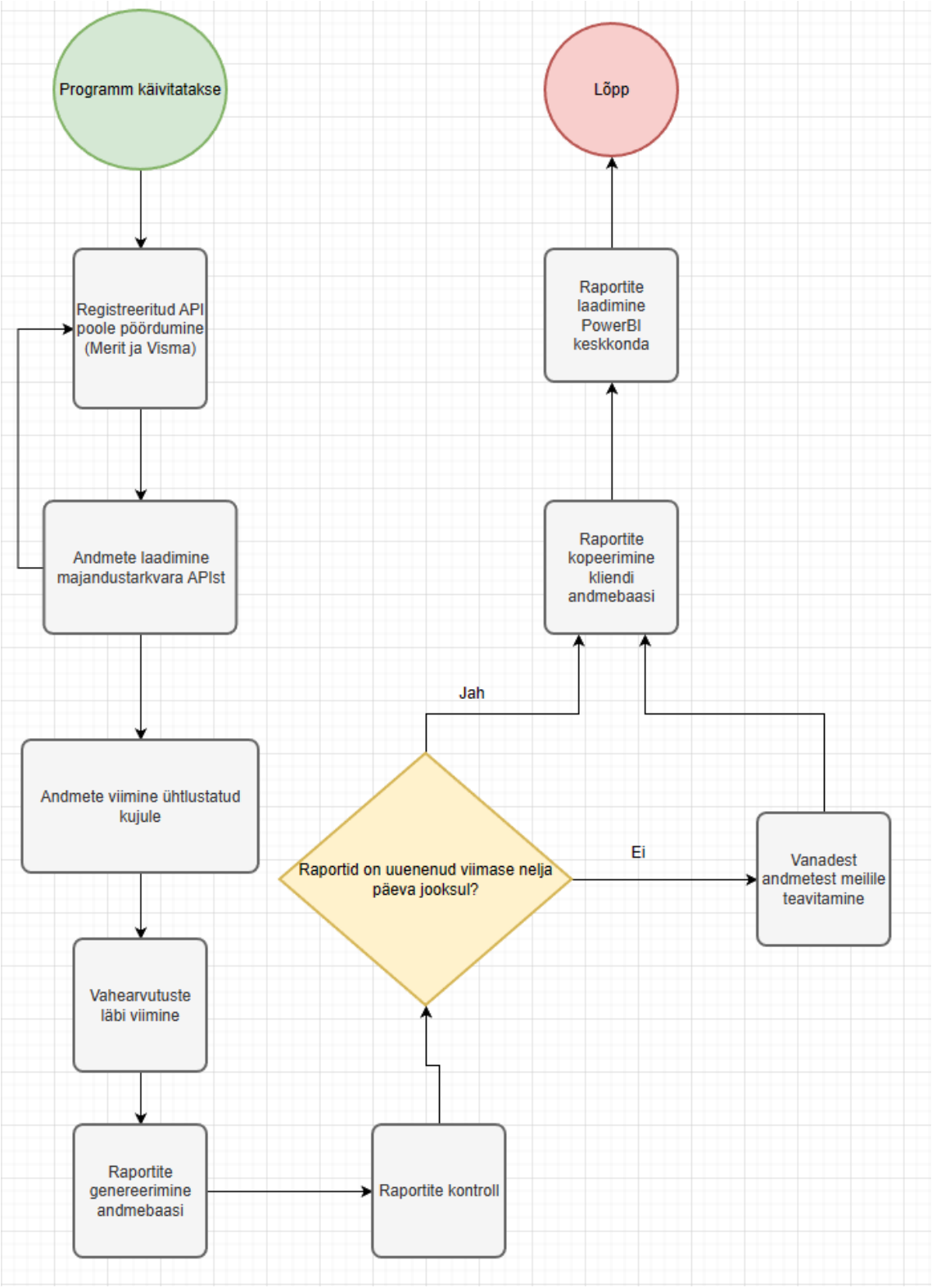
Kogu rakenduse toimimise protsess toimub läbi *Celery* programmi, mis paneb tagataustal kõik vajalikud protsessid tööle kindlalt ette määratud järjekorras. Tänu sellele saab kergelt protsesside järjekorda vajadusel muuta ning välja vahetada ning ei pea mitmes kohas muudatusi tegema protsesside suunamiseks.

Järgmisel joonisel 7 on välja toodud koodiblokk, mis vastutab rakenduse töö eest.

```
@app.task(name='core-refresh-data-warehouse')
def refresh_data_warehouse():
    """
    Refresh all data required for data warehouse to function
    """
    start: datetime = datetime.now()
    logger.info('Refreshing data warehouse data')
    external_data_tasks: list = []
    internal_data_tasks: list = []
    report_tasks: list = [
        refresh_profit_report.si(),
        refresh_balance_report.si(),
        refresh_purchase_sale_report.si()
    ]
    logger.info('Preparing tasks for refreshing data warehouse')
    for company in Company.objects.select_related('api').all():
        if company.api.api == 'Merit':
            temp_external_tasks = [
                merit_fetch_all.si(company.company)
            ]
            external_data_tasks.extend(temp_external_tasks)
        elif company.api.api == 'Visma':
            temp_external_tasks = [visma_fetch_all.si(company.company)]
            external_data_tasks.extend(temp_external_tasks)
        else:
            raise Exception('Unknown API: %s' % company.api.api)
        temp_internal_tasks =
[calculate_balance_accounts.si(company.company),
calculate_profit_accounts.si(company.company)]
        internal_data_tasks.extend(temp_internal_tasks)
    logger.info('Executing tasks for refreshing data warehouse')
    chain(
        group(*external_data_tasks, immutable=True),
        update_data.si(),
        group(*internal_data_tasks, immutable=True),
        group(*report_tasks, immutable=True),
        copy_reports_live.si(),
        check_reports.si(),
        add_excel_data.si(),
        immutable=True
    )()
    logger.info('Data warehouse successfully refreshed!')
    end: datetime = datetime.now()
    logger.info(f'Ran from {start} to {end}')
```

Joonis 7. Celery protsess, mis vastutab rakenduse töö eest.

Joonisel 8 on esitatud voodiagramm, mis võtab kokku rakenduse protsessid.

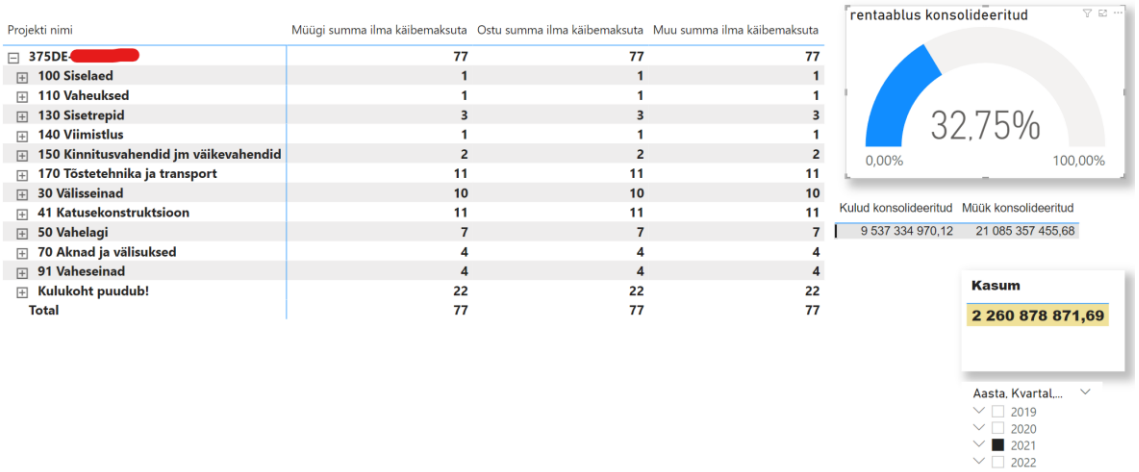


Joonis 8. Rakenduse voodiagramm

4.1.5 Kliendipoolne tehnoloogia

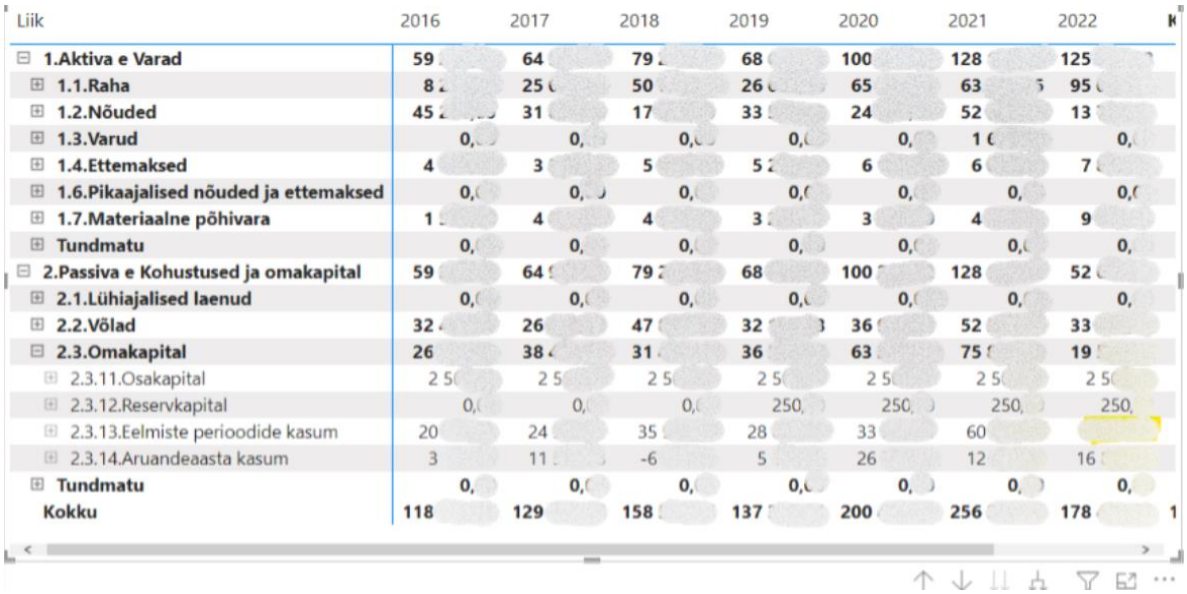
Lõputöö rakenduse kliendipoolseks tehnoloogiaks on kliendi poolt määratud *PowerBI* platvorm. *PowerBI*s loob kasutaja ühenduse lõputöö rakenduse andmebaasiga, kus on talle ette määratud piiratud õigustega kasutaja oma andmebaasi osaga, milles on olemas ainult kliendi poolt tellitud raportid: ostu- ja müügiraport, bilansiraport ning kasumiraport. Peale andmebaasiga ühenduse loomist saab kasutaja määrata, milliseid tabeleid on soov kuvada *PowerBI* platvormil ning lisaks on juba võimalik ka andmete eeltöötlust teha enne nende sisselaadimist. Näiteks on võimalik andmete andmetüüpe muuta, juhul kui platvorm ei tuvasta kohe ära andmebaasi andmetüüpe, ning saab ka juurde lisada andmebaasi tabelist tulnud tabelile veerge, kus on näiteks mitme välja pealt kokku arvutatud tulem.

Peale andmete eeltöötlust jääb üle andmete sisselaadimine. Seda saab teha kõigile andmebaasi tabelitele korraga, aga ka ükshaaval, juhul kui ei ole kõiki andmeid vaja. Peale tabelite määramise ja eeltöötluse saab määrata ka andmete uuenemise, kas see toimib käsitsi värskendamise nuppu vajutades või automaatselt kasutaja poolt määratud tingimuste põhjal. Need tingimused võivad olla: perioodiline tingimus, allikaks olevate andmete kontroll, et värskendada olemasolevaid või laadida sisse uued andmed, mis on juurde tekkinud. Joonisel 9 on välja toodud ostu- ja müügiraporti põhjal tehtud näide illustreerimaks *PowerBI* platvormil nähtavat.



Joonis 9. Ostu- ja müügiraporti põhjal tehtud spetsiifilise projekti aruanne

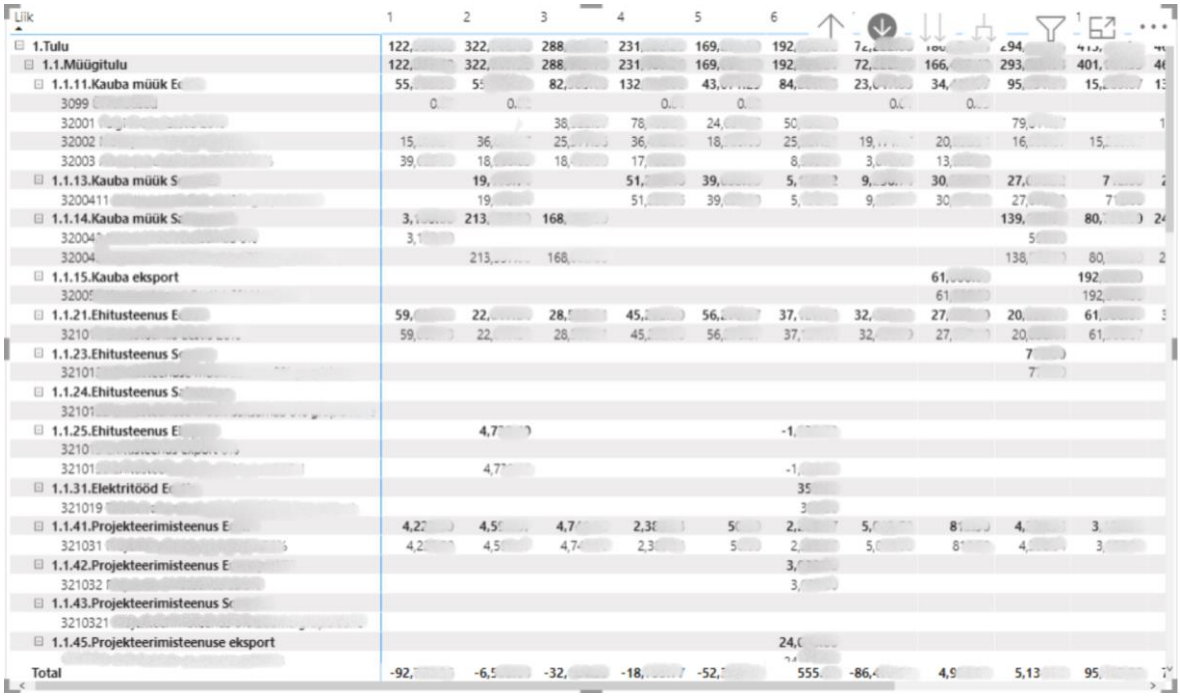
Joonisel 10 on välja toodud bilansiraporti näide ühe firma andmete lõikes. Andmed on osaliselt hägustatud, sest sisaldavad konfidentsiaalset teavet.



Liik	2016	2017	2018	2019	2020	2021	2022
1.Aktiva e Varad	59	64	79	68	100	128	125
1.1.Raha	8	25	50	26	65	63	95
1.2.Nõuded	45	31	17	33	24	52	13
1.3.Varud	0,	0,	0,	0,	0,	1	0,
1.4.Ettemaksed	4	3	5	5	6	6	7
1.6.Pikaajalised nõuded ja ettemaksed	0,	0,	0,	0,	0,	0,	0,
1.7.Materiaalne põhivara	1	4	4	3	3	4	9
Tundmatu	0,	0,	0,	0,	0,	0,	0,
2.Passiva e Kohustused ja omakapital	59	64	79	68	100	128	52
2.1.Lühiajalised laenud	0,	0,	0,	0,	0,	0,	0,
2.2.Võlad	32	26	47	32	36	52	33
2.3.Omakapital	26	38	31	36	63	75	19
2.3.11.Osakapital	2 5	2 5	2 5	2 5	2 5	2 5	2 5
2.3.12.Reservkapital	0,	0,	0,	250,	250,	250,	250,
2.3.13.Eelmiste perioodide kasum	20	24	35	28	33	60	
2.3.14.Aruandeaasta kasum	3	11	-6	5	26	12	16
Tundmatu	0,	0,	0,	0,	0,	0,	0,
Kokku	118	129	158	137	200	256	178

Joonis 10. Bilansiraport ühe firma andmete näitel.

Joonisel 11 on välja toodud näide kasumiraportist ühe aasta lõikes kuude kaupa. Andmed on osaliselt hägustatud, sest sisaldavad konfidentsiaalset teavet.



Liik	1	2	3	4	5	6
1.Tulu	122,	322,	288,	231,	169,	192,
1.1.Müügitulu	122,	322,	288,	231,	169,	192,
1.1.11.Kauba müük E	55,	5	82,	132	43,	84,
3099	0	0.		0.	0.	
32001			38,	78,	24,	50,
32002	15,	36,	25,	36,	18,	25,
32003	39,	18,	18,	17,		8,
1.1.13.Kauba müük S		19,		51,	39,	5,
3200411		19,		51,	39,	5,
1.1.14.Kauba müük S	3,	213,	168,			
32004	3,					
32004		213,	168,			
1.1.15.Kauba eksport						
32005						
1.1.21.Ehitusteenus E	59,	22,	28,	45,	56,	37,
3210	59,	22,	28,	45,	56,	37,
1.1.23.Ehitusteenus S						
32101						
1.1.24.Ehitusteenus S						
32101						
1.1.25.Ehitusteenus E		4,7				-1,
3210						
32101		4,7				-1,
1.1.31.Elektritööd E						35
321019						3
1.1.41.Projekteerimisteenus E	4,2	4,5	4,7	2,3	5	2,
321031	4,2	4,5	4,7	2,3	5	2,
1.1.42.Projekteerimisteenus E						3,
321032						3,
1.1.43.Projekteerimisteenus S						
3210321						
1.1.45.Projekteerimisteenuse eksport						24,
Total	-92,	-6,5	-32,	-18,	-52,	555.

Liik	7	8	9	10	11	12
1.Tulu	72,	166,	293,	401,		
1.1.Müügitulu	72,	166,	293,	401,		
1.1.11.Kauba müük E	23,	34,	95,	15,		
3099	0.	0.				
32001			79,			
32002	19,	20,	16,	15,		
32003	3,	13,				
1.1.13.Kauba müük S	9,	30,	27,	7		
3200411	9,	30,	27,	7		
1.1.14.Kauba müük S			139,	80,		
32004			5			
32004			138,	80,		
1.1.15.Kauba eksport		61,		192,		
32005		61		192,		
1.1.21.Ehitusteenus E	32,	27,	20,	61,		
3210	32,	27,	20,	61,		
1.1.23.Ehitusteenus S			7			
32101			7			
1.1.24.Ehitusteenus S						
32101						
1.1.25.Ehitusteenus E						
3210						
32101						
1.1.31.Elektritööd E						
321019						
1.1.41.Projekteerimisteenus E	5,	8	4,	3		
321031	5,	8	4,	3,		
1.1.42.Projekteerimisteenus E						
321032						
1.1.43.Projekteerimisteenus S						
3210321						
1.1.45.Projekteerimisteenuse eksport						
Total	-86,	4,9	5,13	95,		

Joonis 11. Kasumiraport ühe aasta lõikes kuude kaupa.

4.1.6 Turvalisus

Rakenduse loomisega paralleelselt on tähtis turvalisuse tagamine, seda esmajärjekorras kliendi jaoks tundlike andmete pärast, kuid lisaks lõputöö autori firma jaoks, et keegi ei pääseks ligi rakendusele ega serverile, mille peal rakendus töötab. Selle saavutamiseks peale serveri esialgset seadistamist pani lõputöö autor püsti tulemüüri, mis lubab otse serverile ligi ainult lõputöö firma autori firma poolt lubatud isikutele. Tulemüürist saavad läbi ka isikud kellele on antud ligipääs firma sisesele *VPN*-ile (*Virtual Private Network*) Peale seda, kui kõige esmane turvalisuse tase sai tagatud, läks edasi rõhk andmebaasi peale, kuhu tuli lasta kliendil otseühendus luua, et klient saaks andmetele ligi PowerBI keskkonnas. Selle jaoks võimaldas lõputöö autor andmebaasiga ühenduda ainult ühel kasutajal, millel on turvaline parool peal. Vastav andmebaasi kasutaja on ainult lugemisõigusega ning pääseb ligi andmebaasile, mis sisaldab lõpptulemeid, mida klient soovis saada. Kõik tehnilisemad andmebaasi tabelid, mida on vaja rakenduse funktsioneerimiseks, on eraldiseisvas andmebaasis, kuhu ei pääse keegi serveriväliselt. Lisaks andmebaasile on rakendusel olemas veebirakenduse osa, mis on suures osas kaetud Django raamistiku poolt pakutavate turvalisust tagavate lahendustega. Need lahendused on:

- Kaitse murdskriptimise ehk *XSS* (*Cross Site Scripting*) vastu.
- Kaitse päringuvõltsingu ehk *CSRF* (*Cross Site Request Forgery*) vastu.
- Kaitse *SQL Injection* vastu.
- Kaitse *clickjacking* vastu.
- SSL/HTTPS – takistab autentimisandmete ning muu informatsiooni leket kliendi ja serveri vahel.
- Päringu „*Host“* päise validatsioon – kontrollib eraldi, et päise väärtus on olemas lubatud väärtuste hulgas milleks tavaliselt on serveri suhtes: localhost, 127.0.0.1, serveri IP aadress.
- Kasutajate paroolide krüpteerimine ning soolamine (salting).

5 Tulemused

Antud peatükis kirjeldatakse lõputöö raames valminud rakenduse funktsionaalsust ning selle vastavust soovitule. Lisaks käiakse üle rakenduse testimine ja rakendusest saadav kasumlikkus. Viimasena kirjeldatakse rakenduse edasiarenduse võimalusi.

5.1 Funktsionaalsused

Lõputöö raames valminud rakendus on lõppkasutaja, raamatupidaja poolt heakskiidu saanud ning töötab nüüd täisautomaatselt ilma käsitöö vajaduseta. Rakendus töötab graafiku põhiselt esialgu kord ööpäevas võimalusega seda vajadusel muuta. Lisaks sellele on kasutajal võimalus rakendus tööle panna manuaalselt. Rakendus laeb alla andmed *Merit* ja *Visma* majandustarkvaradest, viib nad ühtlustatud kujule ning töötleb neid. Peale andmete töötlemist tehakse valmiskolme liiki raportid: bilansi raport, kasumiraport ning ostu- ja müügiraport, mis tehakse lõppkasutajale ligipääsetavaks läbi *PowerBI* platvormi.

5.2 Testimine

Rakendust testisid kliendi firma arendusjuht koos seda peamiselt kasutama hakkava pearaamatupidajaga. Testimine toimus järk-järgult iga kord kui uus funktsionaalsus sai juurde arendatud. Testimisest selgus, et mõned kliendile tähtsad andmed on andmemudelites peidetud ning nõrgalt seostatud oma põhimudeliga. Lõputöö autoril kulus selleks omajagu aega, et peidetud andmed kätte saada ja koos nendega raportid korda teha. Veel sai täiendatud arvutuste läbiviimist kliendi soovil – olenevalt kust andmed pärit on, korrutatakse andmed ette antud kordajaga läbi, et saada õige tulemus majandustarkvaradest saadud andmetest.

5.3 Edasiarenduse võimalused

Loodud lahendus on praegusel kujul edukalt toimiv. Mõningad lõputöö rakenduse edasiarenduse võimalused oleks:

- Vahekihi lisamine andmebaasi ja *PowerBI* vahele – võtab ära vajaduse andmebaasi kättesaadavaks tegemise ülejäänud maailmale ning aitab tagada parema turvalisuse kliendi andmetele.

- Roteeruvad paroolid – praegu nii rakenduse veebiliidese osas kui ka andmebaasiga ühendamisel on kasutusel paroolid, mida ei vahetata ning mis aja jooksul võivad välja lekkida. Selle vastu aitab paroolide roteerumine paika pandud perioodilisusega.

- Logimine – rakenduse logimine on kohati puudulik, mistõttu võib probleemide diagnoosimine olla raskendatud. Lisaks sellele tuleb täiendada logitavat informatsiooni metaandmetega, mis võimaldaksid kiiret probleemide otsimist ning analüüsimist.

- Andmebaasi vahetus – praegu kasutusel olev MySQL andmebaas on piiratud võimekusega erinevate tabelite andmete pealt dünaamiliste andmete genereerimiseks, mistõttu on plaanis kasutusele võtta PostgreSQL. Uues andmebaasis saab erinevate tabelite peale ehitada materialiseeritud vaated, mis uuenevad aluseks olevate tabelite tulemusel, ning kus saab läbi viia juba andmebaasi tasemel mingi osa arvutusi ja teisendamisi.

- Tehnoloogiate kaasajastamine – vahepeal on edasiarenguid toimunud rakenduse aluseks oleva Python programmeerimiskeelel kus tuli välja versioon, mis reklaamib ennast kui 10-60% kiiremana eelmistest versioonidest. Antud lisakiirus teeks rakenduse palju kiiremaks ja avaks uued võimalused tehnoloogia osas edasi arendusteks.

Antud loetelu ei ole lõplik ning leidub muid võimalusi edasisteks arendusteks, kuid välja toodud valikud on kahtlemata tähtsad ning vajaksid esimeses järjekorras tegemist.

6 Kokkuvõte

Lõputöö eesmärk oli luua tarkvaralahendus, mis suudab hankida majanduslikud andmed erinevatest keskkondadest, viia nad samale kujule ning ehitada nendest kliendi poolt paika pandud raportid. Lisaks peab rakendus saatma valmis raportid *PowerBI* keskkonda, kus klient saab neid analüüsida ning nende peale edasi ehitada visuaalseid ülevaateid firmas toimuvast.

Lõputöö käigus suheldi rakenduse tellinud kliendiga ning pandi paika funktsionaalsed ja mittefunktsionaalsed nõuded mida lõputöö raames loodav rakendus peab täitma. Peale nõuete paika panekut tehti analüüs olemasolevate rakenduste suhtes, et kindlaks teha loodava rakenduse otstarbekus ning asuti analüüsima erinevaid tehnoloogiaid, mida rakenduse loomisel kasutada. Edasi toimus võimalike tehnoloogiate analüüs rakenduse erinevate osade jaoks. Kliendipoolne tehnoloogia on kliendi poolt paika pandud *PowerBI* platvorm, kuid sai analüüsida teenusepoolseid tehnoloogiaid ning andmebaase. Andmebaasiks sai valitud MySQL selle kerge ühilduvuse tõttu *PowerBI* platvormiga, ning teenusepoolseks tehnoloogiaks sai valitud Python programmeerimiskeel koos Django raamistikuga. Järgmine osa oli rakenduse enda loomine ja testimine kliendi poolt.

Lõputöö raames valminud rakendus võimaldab majandustarkvaradest *Merit* ja *Visma* andmeid sisse laadida. Sisse laetud andmeid oskab rakendus töödelda ümber ühtlasele kujule. Peale ühtlasele kujule toimetamist loob rakendus andmete põhjal kolm erinevat raportit: ostu- ja müügiraport, kasumiraport ning bilansiraport. Raportite loomisel tekivad osad raporti veerud olemasolevate veergude arvutuste tulemusena ning kõik tulemused salvestatakse andmebaasi millele on piiratud õigustega ligipääs kliendil. Rakendus töötab perioodiliselt koos võimalusega käivitada manuaalselt. Rakenduse poolt loodud raportid saadetakse edasi *PowerBI* platvormile kus nendega toimetatakse edasi.

Tulevikus on võimalik lõputöö rakendust edasi täiendada erinevates valdkondades – on võimalik täiendada rakenduse turvalisust, sooritada arhitektuurilisi muudatusi ning täiendada logimist eesmärgiga ennetada võimalike vigade teket ja reageerida neile operatiivselt.

Kasutatud kirjandus

[1] Microsoft, „What is PowerBI?,“ [Võrgumaterjal]. Saadaval: https://powerbi.microsoft.com/en-us/what-is-power-bi/. [Kasutatud 29 11 2022].

[2] Merit, „Kui raamatupidamine on kiire ja lihtne, on teil rohkem aega oma ettevõtte arendamiseks,“ [Võrgumaterjal]. Saadaval: https://www.merit.ee/ettevottest/. [Kasutatud 26 12 2022].

[3] Visma Group, „A history of software innovation,“ [Võrgumaterjal]. Saadaval: https://www.visma.com/organisation/history/. [Kasutatud 26 12 2022].

[4] Visma Group, „Who we are,“ [Võrgumaterjal]. Saadaval: https://www.visma.com/organisation/. [Kasutatud 26 12 2022].

[5] Wikipedia, „Microsoft Power BI,“ [Võrgumaterjal]. Saadaval: https://en.wikipedia.org/wiki/Microsoft_Power_BI. [Kasutatud 29 12 2022].

[6] P. Gorbachenko, „What are Functional and Non-Functional Requirements and How,“ [Võrgumaterjal]. Saadaval: https://enkonix.com/blog/functional-requirements-vs-non-functional/. [Kasutatud 29 11 2022].

[7] M. Martin, „What is Non-Functional Requirement in Software Engineering?,“ [Võrgumaterjal]. Saadaval: https://www.guru99.com/non-functional-requirement-type-example.html. [Kasutatud 29 11 2022].

[8] RedFunction OÜ, „RedFunction,“ [Võrgumaterjal]. Saadaval: https://redfunction.ee/et/. [Kasutatud 29 11 2022].

[9] Mastercard, „Open Banking Connect,“ [Võrgumaterjal]. Saadaval: https://developer.mastercard.com/product/open-banking-connect/. [Kasutatud 2 1 2023].

[10] Paybook, „Syncfy,“ [Võrgumaterjal]. Saadaval: https://syncfy.com/. [Kasutatud 29 11 2022].

[11] InventorSoft, „LedgerSync,“ [Võrgumaterjal]. Saadaval: https://inventorsoft.co/projects/ledgersync. [Kasutatud 29 11 2022].

[12] L. Dev, „Top 8 Most Demanded Programming Languages in 2022,“ [Võrgumaterjal]. Saadaval: https://www.devjobsscanner.com/blog/top-8-most-demanded-languages-in-2022/. [Kasutatud 18 12 2022].

[13] N. Ferguson, „What's the Difference Between Frontend and Backend Web Development?,“ [Võrgumaterjal]. Saadaval: https://careerfoundry.com/en/blog/web-development/whats-the-difference-between-frontend-and-backend/. [Kasutatud 29 11 2022].

[14] V. Bhagat, „Pros and Cons of Python Programming Language,“ [Võrgumaterjal]. Saadaval: https://www.pixelcrayons.com/blog/python-pros-and-cons/. [Kasutatud 29 11 2022].

[15] DataFlair, „Pros and Cons of Java | Advantages and Disadvantages of Java,“ [Võrgumaterjal]. Saadaval: https://data-flair.training/blogs/pros-and-cons-of-java/. [Kasutatud 29 11 2022].

[16] AlexSoft, „The Good and the Bad of C# Programming,“ [Võrgumaterjal]. Saadaval: https://www.altexsoft.com/blog/c-sharp-pros-and-cons/. [Kasutatud 29 11 2022].

[17] WEEKLYHOW, „Pros and Cons of JavaScript in 2020 (Become a Developer),“ [Võrgumaterjal]. Saadaval: https://weeklyhow.com/pros-and-cons-of-javascript/. [Kasutatud 29 11 2022].

[18] Coursera Inc, „How Long Does it Take to Learn Python?,“ [Võrgumaterjal]. Saadaval: https://www.coursera.org/articles/how-long-does-it-take-to-learn-python-tips-for-learning. [Kasutatud 18 12 2022].

[19] N. Eubank, „Speed in Python,“ [Võrgumaterjal]. Saadaval: https://www.practicaldatascience.org/html/performance_understanding.html. [Kasutatud 18 12 2022].

[20] Noble Desktop, „How Long Does it Take to Learn Java?,“ [Võrgumaterjal]. Saadaval: https://www.nobledesktop.com/learn/java/how-long-does-it-take-to-learn-java. [Kasutatud 18 12 2022].

[21] G. Brihadiswaran, „A Performance Comparison Between C, Java, and Python,“ [Võrgumaterjal]. Saadaval: https://medium.com/swlh/a-performance-comparison-between-c-java-and-python-df3890545f6d. [Kasutatud 18 12 2022].

[22] CodeBerry, „Beginner’s Guide to C# programming,“ [Võrgumaterjal]. Saadaval: https://codeberryschool.com/blog/en/beginners-guide-to-c-sharp-programming/. [Kasutatud 18 12 2022].

[23] P. Pedamkar, „C# vs Java Performance,“ [Võrgumaterjal]. Saadaval: https://www.educba.com/c-sharp-vs-java-performance/. [Kasutatud 18 12 2022].

[24] Thinkful Inc, „How Long Does It Take to Learn JavaScript?,“ [Võrgumaterjal]. Saadaval: https://www.thinkful.com/blog/how-long-does-it-take-to-learn-javascript/. [Kasutatud 18 12 2022].

[25] N. Raval, „NodeJS vs Python: The Great Backend Dilemma,“ [Võrgumaterjal]. Saadaval: https://radixweb.com/blog/nodejs-vs-python. [Kasutatud 18 12 2022].

[26] pythonbasics.org, „What is Flask Python,“ [Võrgumaterjal]. Saadaval: https://pythonbasics.org/what-is-flask-python/. [Kasutatud 29 11 2022].

[27] Django Software Foundation, „Meet Django,“ [Võrgumaterjal]. Saadaval: https://www.djangoproject.com/. [Kasutatud 29 11 2022].

[28] VMWare, „Spring Framework,“ [Võrgumaterjal]. Saadaval: https://spring.io/projects/spring-framework. [Kasutatud 29 11 2022].

[29] Microsoft, „What is .NET?,“ [Võrgumaterjal]. Saadaval: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet. [Kasutatud 29 11 2022].

[30] VueJS, „What is Vue?,“ [Võrgumaterjal]. Saadaval: https://vuejs.org/guide/introduction.html. [Kasutatud 29 11 2022].

[31] Meta Platforms Inc, „React. A JavaScript library for building user interfaces,“ [Võrgumaterjal]. Saadaval: https://reactjs.org/. [Kasutatud 29 11 2022].

[32] L. Rogerson, „Flask vs. Django – Which Framework to Learn First?,“ [Võrgumaterjal]. Saadaval: https://blog.teamtreehouse.com/flask-vs-django. [Kasutatud 18 12 2022].

[33] AmiyaRanjanRout, „Best Way to Master Spring Boot – A Complete Roadmap,“ [Võrgumaterjal]. Saadaval: https://www.geeksforgeeks.org/best-way-to-master-spring-boot-a-complete-roadmap/. [Kasutatud 18 12 2022].

[34] JavaTPoint, „Java Spring Pros and Cons,“ [Võrgumaterjal]. Saadaval: https://www.javatpoint.com/java-spring-pros-and-cons. [Kasutatud 18 12 2022].

[35] M. Behler, „Spring Security: Authentication and Authorization In-Depth,“ [Võrgumaterjal]. Saadaval: https://www.marcobehler.com/guides/spring-security. [Kasutatud 18 12 2022].

[36] J. Hilton, „What to focus on when learning ASP.NET Core?,“ [Võrgumaterjal]. Saadaval: https://jonhilton.net/what-to-focus-on-when-learning-aspnet/. [Kasutatud 18 12 2022].

[37] Microsoft, „What is .NET? Introduction and overview,“ [Võrgumaterjal]. Saadaval: https://learn.microsoft.com/en-us/dotnet/core/introduction. [Kasutatud 18 12 2022].

[38] Microsoft, „Key Security Concepts,“ [Võrgumaterjal]. Saadaval: https://learn.microsoft.com/en-us/dotnet/standard/security/key-security-concepts. [Kasutatud 18 12 2022].

[39] J. Olawanle, „Vue.js vs React - How to Choose the Right Framework,“ [Võrgumaterjal]. Saadaval: https://hygraph.com/blog/vuejs-vs-react. [Kasutatud 18 12 2022].

[40] VueJs, „Security,“ [Võrgumaterjal]. Saadaval: https://vuejs.org/guide/best-practices/security.html#best-practices. [Kasutatud 18 12 2022].

[41] A. Dziuba, „React.js Security Guide: Threats, Vulnerabilities, and Ways to Fix Them in 2023,“ [Võrgumaterjal]. Saadaval: https://relevant.software/blog/react-js-security-guide/. [Kasutatud 18 12 2022].

[42] The Mozilla Foundation, „What is JavaScript?,“ [Võrgumaterjal]. Saadaval: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript. [Kasutatud 29 11 2022].

[43] StackOverflow, „StackOverflow,“ [Võrgumaterjal]. Saadaval: https://survey.stackoverflow.co/2022/#technology-most-popular-technologies. [Kasutatud 7 12 2022].

[44] Hwaci, „About SQLite,“ [Võrgumaterjal]. Saadaval: https://www.sqlite.org/about.html. [Kasutatud 29 11 2022].

[45] Oracle, „What is MySQL?,“ [Võrgumaterjal]. Saadaval: https://www.oracle.com/mysql/what-is-mysql/. [Kasutatud 29 11 2022].

[46] PostgreSQL Global Development Group, „What is PostgreSQL?,“ [Võrgumaterjal]. Saadaval: https://www.postgresql.org/about/. [Kasutatud 29 11 2022].

[47] SQLite, „Defense Against The Dark Arts,“ [Võrgumaterjal]. Saadaval: https://www.sqlite.org/security.html. [Kasutatud 18 12 2022].

[48] Microsoft, „Power BI data sources,“ [Võrgumaterjal]. Saadaval: https://learn.microsoft.com/en-us/power-bi/connect-data/power-bi-data-sources. [Kasutatud 18 12 2022].

[49] Satoricyber, „3 Pillars of PostgreSQL Security,“ [Võrgumaterjal]. Saadaval: https://satoricyber.com/postgres-security/3-pillars-of-postgresql-security/. [Kasutatud 18 12 2022].

[50] Satoricyber, „MySQL Security – Common Threats And 8 Best Practices,“ [Võrgumaterjal]. Saadaval: https://satoricyber.com/mysql-security/mysql-security-common-threats-and-8-best-practices/. [Kasutatud 18 12 2022].

[51] Atlassian, „Source code management,“ [Võrgumaterjal]. Saadaval: https://www.atlassian.com/git/tutorials/source-code-management. [Kasutatud 18 12 2022].

[52] H. Escafit, „GitHub vs. GitLab vs. BitBucket,“ [Võrgumaterjal]. Saadaval: https://blog.mergify.com/github-vs-gitlab-vs-bitbucket/. [Kasutatud 18 12 2022].

[53] Atlassian, „Move fast, stay aligned, and build better - together,“ [Võrgumaterjal]. Saadaval: https://www.atlassian.com/software/jira. [Kasutatud 18 12 2022].

[54] Atlassian, „How to choose the best source code repository,“ [Võrgumaterjal]. Saadaval: https://bitbucket.org/product/code-repository. [Kasutatud 18 12 2022].

[55] Red Hat, „What is an IDE?,“ [Võrgumaterjal]. Saadaval: https://www.redhat.com/en/topics/middleware/what-is-ide. [Kasutatud 18 12 2022].

[56] Simplilearn Solutions, „Top 10 Python IDEs in 2023: Choosing The Best One,“ [Võrgumaterjal]. Saadaval: https://www.simplilearn.com/tutorials/python-tutorial/python-ide. [Kasutatud 18 12 2022].

[57] The Python Software Foundation, „IDLE,“ [Võrgumaterjal]. Saadaval: https://docs.python.org/3/library/idle.html. [Kasutatud 23 12 2022].

[58] JetBrains, „ALL THE PYTHON TOOLS,“ [Võrgumaterjal]. Saadaval: https://www.jetbrains.com/pycharm/. [Kasutatud 23 12 2022].

[59] Microsoft, „Why did we build Visual Studio Code?,“ [Võrgumaterjal]. Saadaval: https://code.visualstudio.com/docs/editor/whyvscode. [Kasutatud 23 12 2022].

[60] WSGI.org, „What is WSGI?,“ [Võrgumaterjal]. Saadaval: https://wsgi.readthedocs.io/en/latest/what.html. [Kasutatud 23 12 2022].

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

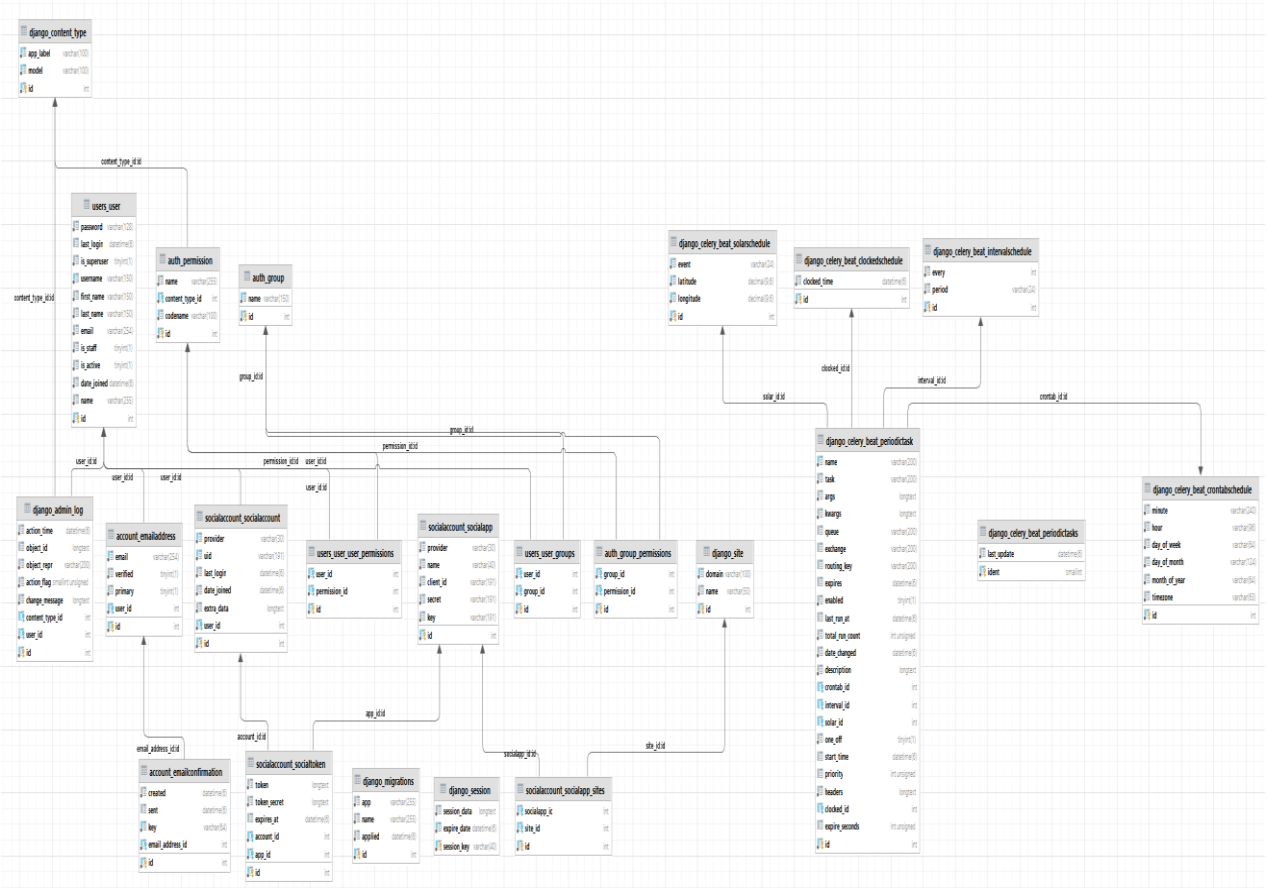
Mina, Frank Martin Kiibus

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

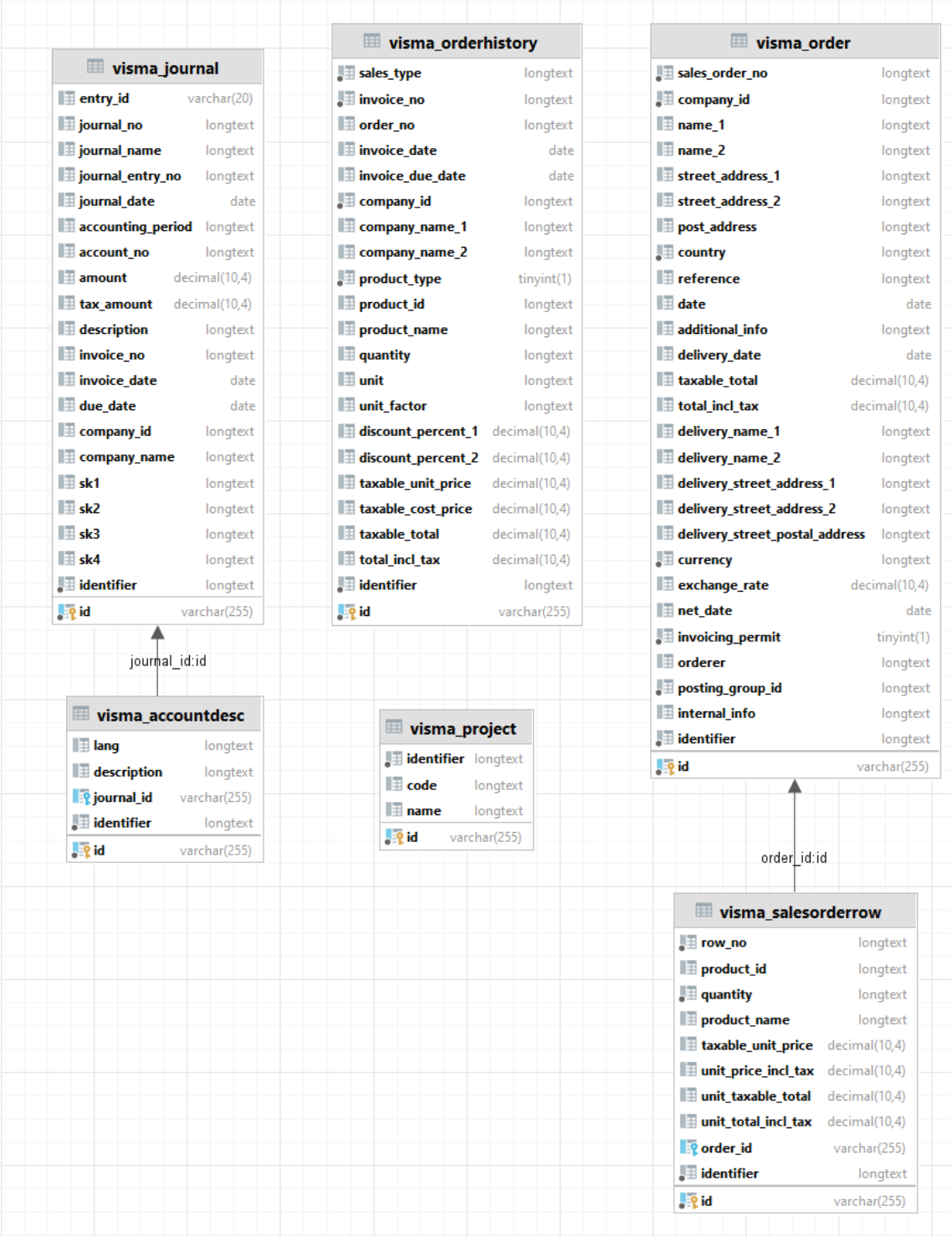
05.01.2023

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

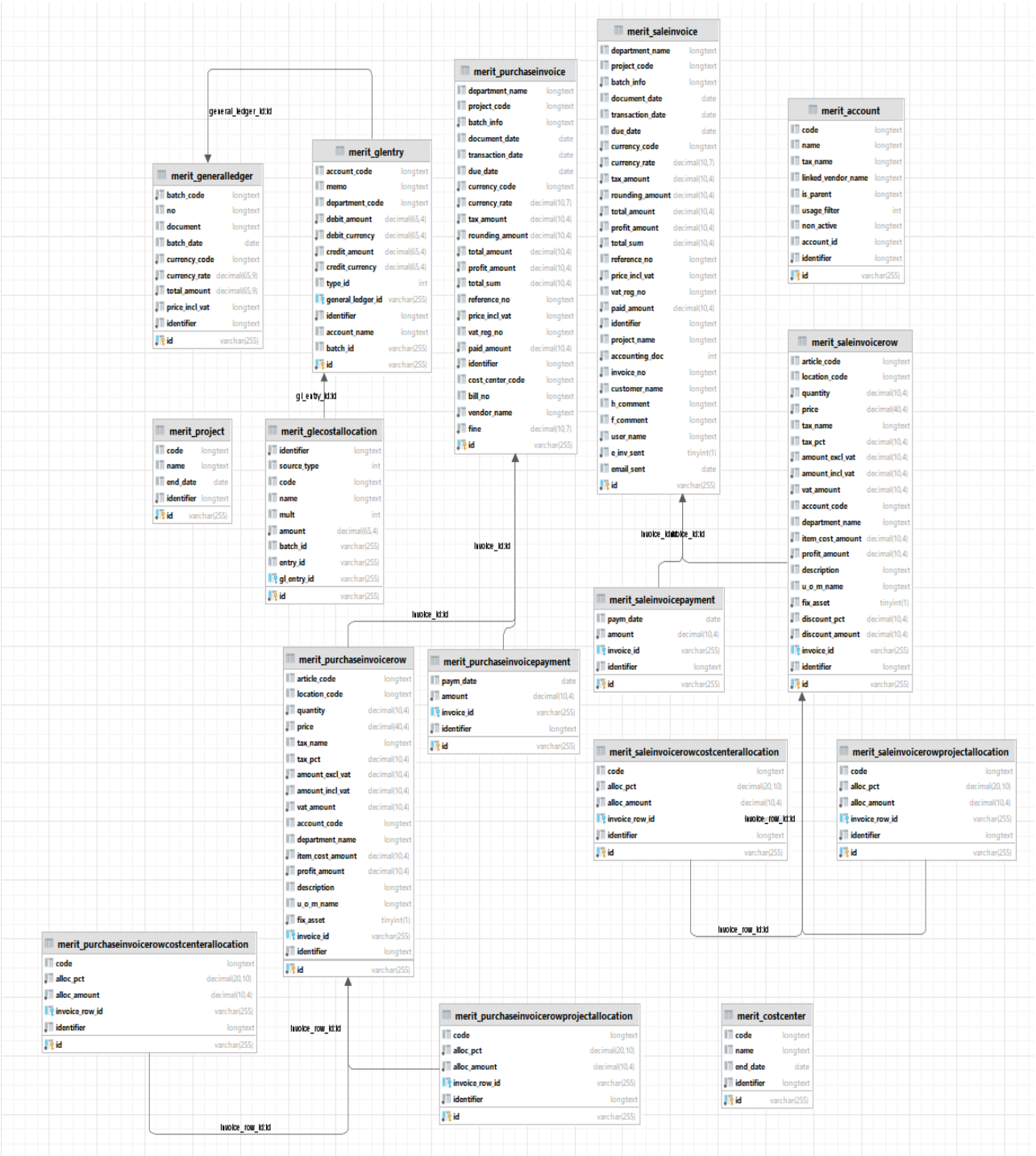
Lisa 2 – Raamistiku tabelite ERD skeem



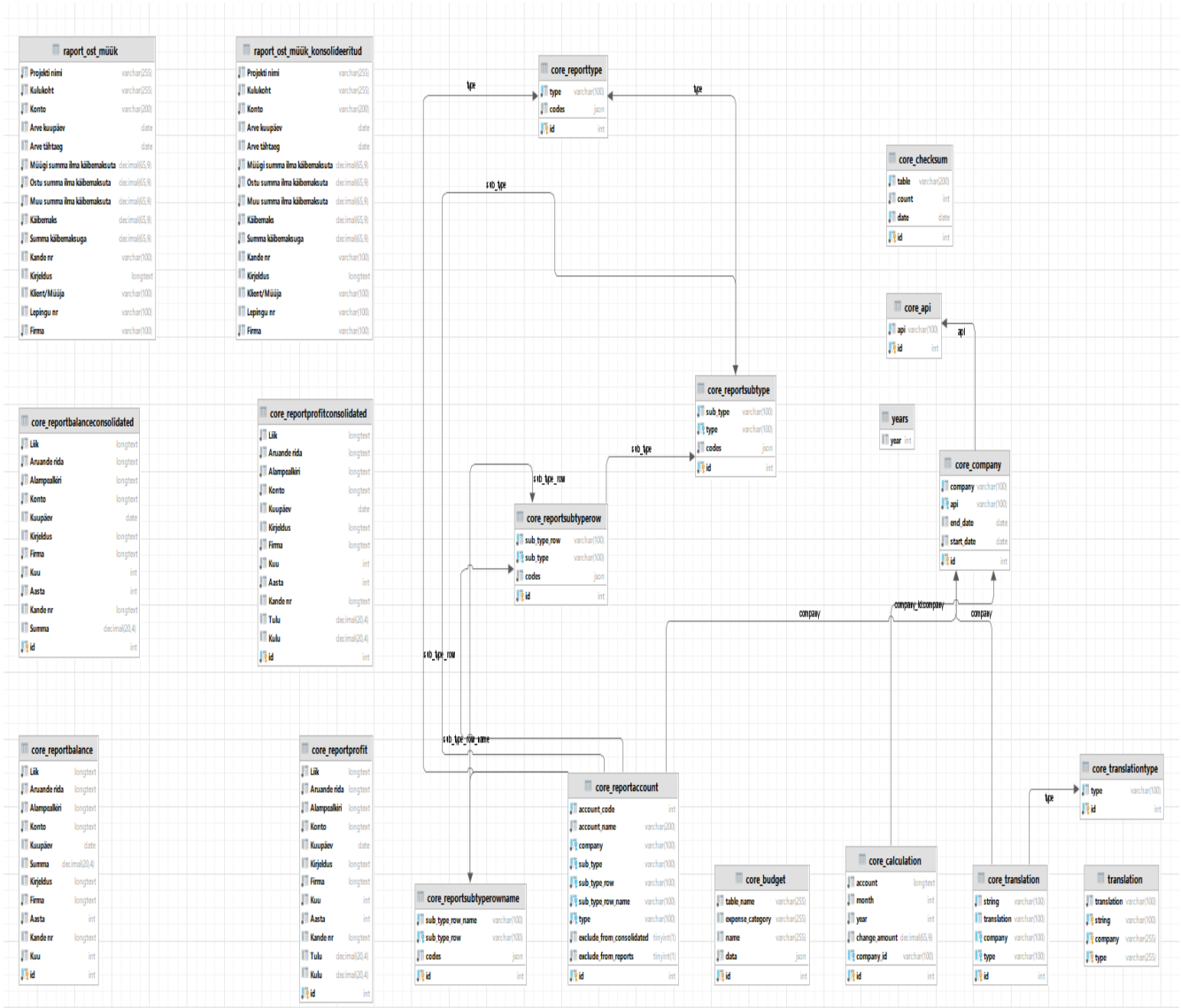
Lisa 3 – Visma majandustarkvara kaardistavate tabelite ERD skeem



Lisa 4 – Merit majandustarkvara kaardistavate tabelite ERD skeem



Lisa 5 – Funktsionaalsete tabelite ERD skeem



Lisa 6 – Bilansiraporti loomise kood

```
def core_balance(
        refresh_data: bool = False,
        comps: list or None = None,
        accs: list or None = None
) -> None:
    """
    Create Balance report (both unconsolidated and consolidated)

    :param refresh_data: Whether to refresh business logic data or
not, default False
    :param comps: Optional list of company names to use during balance
creation
    :param accs: Optional list of accounts to use during balance
creation
    """
    logger.info('Starting ReportBalance Reports creation...')

    logger.info('Querying companies with their start and end dates
from Company Table...')

    companies = Company.objects.filter(**({} if not comps else
{'company__in': comps})).all()

    if refresh_data:
        logger.info('Revalidating information tables...')
        core_update_data()

        for company in companies:
            logger.info(f'Re-calculating account changes per month per
year for {company.company}')
            core_calculate_account_change(company.company, 'Bilanss')

    # Delete old data
    logger.info('Deleting old data from ReportBalance Report
Tables...')

    ReportBalance.objects.filter(
        **{
            **({} if not comps else {'company__in': comps}),
            **({} if not accs else {'account__in': accs})
        }
    ).all().delete()

    ReportBalanceConsolidated.objects.filter(
        **{
            **({} if not comps else {'company__in': comps}),
            **({} if not accs else {'account__in': accs})
        }
    ).all().delete()

    logger.info('Querying balance type From ReportType Table...')

    rtype: ReportType = ReportType.objects.get(codes__0__lte=2)

    for company in companies:
        api: str = company.api.api
```

```
        logger.info(f'Querying accounts From ReportAccount Table for
Type: {rtype} Company: {company.company}...')
        accounts: QuerySet = ReportAccount.objects.filter(
            type=rtype,
            company=company,
            exclude_from_reports=False,
            **({} if not accs else {'account_code__in': accs})
        )

        unconsolidated_list: list = []
        consolidated_list: list = []

        for account in accounts:
            if api in ['Merit', 'Visma']:
                result: QuerySet = Calculation.objects.filter(
                    company=company.company,
                    account=account.account_code).all()

                for entry in result:
                    model: ReportBalance = ReportBalance(
                        type=account.sub_type.sub_type,
                        sub_type=account.sub_type_row.sub_type_row,

sub_type_row=account.sub_type_row_name.sub_type_row_name,
                        account=account.account_name,
                        date=None,
                        total=entry.change_amount,
                        description=None,
                        company=company.company,
                        month=entry.month,
                        year=entry.year,
                        batch_info=None
                    )

                    unconsolidated_list.append(model)

                    if not account.exclude_from_consolidated:
                        model: ReportBalanceConsolidated =
ReportBalanceConsolidated(
                            type=account.sub_type.sub_type,

sub_type=account.sub_type_row.sub_type_row,

sub_type_row=account.sub_type_row_name.sub_type_row_name,
                            account=account.account_name,
                            date=None,
                            total=entry.change_amount,
                            description=None,
                            company=company.company,
                            month=entry.month,
                            year=entry.year,
                            batch_info=None
                        )

                        consolidated_list.append(model)
            else:
                raise Exception(f'API: {api} is invalid!')

        logger.info(f'Saving {len(unconsolidated_list)} entries from
{company.company} to Balance Report Table...')
        ReportBalance.batch_update_or_create(unconsolidated_list)
```

```
        logger.info(
            f'Saving {len(consolidated_list)} entries from
{company.company} to Consolidated Balance Report Table...'
        )

ReportBalanceConsolidated.batch_update_or_create(consolidated_list)

unconsolidated_list.clear()
consolidated_list.clear()
```

Lisa 7 – Kasumiraporti loomise kood

```
def core_profit(
        refresh_data: bool = False,
        comps: list or None = None,
        accs: list or None = None
) -> None:
    """
    Create Profit report (both unconsolidated and consolidated)

    :param refresh_data: Whether to refresh business logic data or
not, default False
    :param comps: Optional list of company names to use during balance
creation
    :param accs: Optional list of accounts to use during balance
creation
    """
    logger.info('Starting ReportProfit Reports creation...')

    logger.info('Querying companies with their start and end dates
from Company Table...')

    companies = Company.objects.filter(**({} if not comps else
{'company__in': comps})).all()

    if refresh_data:
        logger.info('Revalidating information tables...')
        core_update_data()

        for company in companies:
            logger.info(f'Re-calculating account changes per month per
year for {company.company}')
            core_calculate_account_change(company.company,
'Kasumiaruanne')

    # Delete old data
    logger.info('Deleting old data from ReportProfit Report
Tables...')

    ReportProfit.objects.filter(
        **{
            **({} if not comps else {'company__in': comps}),
            **({} if not accs else {'account__in': accs})
        }
    ).all().delete()

    ReportProfitConsolidated.objects.filter(
        **{
            **({} if not comps else {'company__in': comps}),
            **({} if not accs else {'account__in': accs})
        }
    ).all().delete()

    logger.info('Querying balance type From ReportType Table...')

    rtype: ReportType = ReportType.objects.get(codes__0__gte=3)

    for company in companies:
        api: str = company.api.api
```

```
        logger.info(f'Querying accounts From ReportAccount Table for
Type: {rtype} Company: {company.company}...')
        accounts: QuerySet = ReportAccount.objects.filter(
            type=rtype,
            company=company,
            exclude_from_reports=False,
            **({} if not accs else {'account_code__in': accs})
        )

        unconsolidated_list: list = []
        consolidated_list: list = []

        for account in accounts:
            if api in ['Merit', 'Visma']:
                result: QuerySet = Calculation.objects.filter(
                    company=company.company,
                    account=account.account_code
                ).all()

                for entry in result:
                    code: int = int(str(account)[0])

                    model = ReportProfit(
                        type=account.sub_type.sub_type,
                        sub_type=account.sub_type_row.sub_type_row,

sub_type_row=account.sub_type_row_name.sub_type_row_name,
                        account=account.account_name,
                        date=None,
                        profit=entry.change_amount if code == 3 else
Decimal(0.00),
                        loss=entry.change_amount if code > 3 else
Decimal(0.00),
                        description=None,
                        company=company.company,
                        month=entry.month,
                        year=entry.year,
                        batch_info=None
                    )

                    unconsolidated_list.append(model)

                    if not account.exclude_from_consolidated:
                        model = ReportProfitConsolidated(
                            type=account.sub_type.sub_type,

sub_type=account.sub_type_row.sub_type_row,

sub_type_row=account.sub_type_row_name.sub_type_row_name,
                            account=account.account_name,
                            date=None,
                            profit=entry.change_amount if code == 3
else Decimal(0.00),
                            loss=entry.change_amount if code > 3 else
Decimal(0.00),
                            description=None,
                            company=company.company,
                            month=entry.month,
                            year=entry.year,
                            batch_info=None
```

```
                        )

                        consolidated_list.append(model)
            else:
                raise Exception(f'API: {api} is invalid!')

        logger.info(f'Saving {len(unconsolidated_list)} entries from
{company.company} to Profit Report Table...')
        ReportProfit.batch_update_or_create(unconsolidated_list)

        logger.info(
            f'Saving {len(consolidated_list)} entries from
{company.company} to Consolidated Profit Report Table...'
        )

ReportProfitConsolidated.batch_update_or_create(consolidated_list)

unconsolidated_list.clear()
consolidated_list.clear()
```