TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Siim Mõtshärg 185621IADB

Kliendiandmete ekspordi automatiseerimine majandustarkvarasse Directo

Bakalaureusetöö

Juhendaja: Kristiina Hakk
PhD
Kaasjuhendaja: Anthony Rouillot
MSc

Tallinn 2022

Autorideklaratsioon

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

Autor: Siim Mõtshärg

16.05.2022

Annotatsioon

Ettevõte Adcash OÜ kasutab reklaamijatest klientide arvete haldamiseks Directo nimelist finantsarvestuse teenust. Directo tarkvarasse sisestatakse kõik reklaamijate arved ning arvete edastamiseks vajalikud klientide isiklikuandmed. Hetkel peavad ettevõtte finantsosakonna töötajad kõik reklaamijate andmed ning arved käsitsi sisestama, selline lähenemine on väga ajakulukas ning veaaldis. Sellest tulenevalt on ettevõte otsustanud automatiseerida kogu andmete ja arvete edastamise protsessi.

Bakalaureusetöö eesmärk on luua tarkvaralahendus, mis automatiseeriks klientide andmete ning arvete edastamist Directo majandustarkvarasse.

Töö teoreetilises osas kaardistatakse ettevõtte veebirakenduse arhitektuuri- ning arendusmustrid ja kasutatud tehnoloogiad. Nende leidude põhjal saab leida lahenduse, mis kõrvaldaks lõputöös käsiteldava probleemi ning samaaegselt säilitaks veebirakenduse, kuhu loodav lahendus luuakse, terviklikkuse.

Töö praktilise osana valmib tarkvaralahendus kliendiandmete ja arvete edastamiseks majandustarkvarasse Directo.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 34 leheküljel, kuut peatükki, 12 joonist, ühte tabelit.

Abstract

Automating Client Data Export to Business Software Directo

The company Adcash OÜ is using a business software called Directo to handle the invoices for their advertiser clients. The employees of the company's financial department must insert all the advertisers' invoices and personal data that is necessary for sending the invoices to Directos' software. Currently all the data entry is done manually, and this process is very time consuming and prone to error. That is why Adcash has decided to reduce the workload of the financial department and automate the client data exporting process. The aim of this bachelor's thesis is to create a software solution in the company's web application that would automate the client data export to business software Directo.

The theoretical part describes the existing web applications software design and architectural patterns, also the technologies used. This analysis aids in finding the best possible approach to the problem that would simultaneously solve the issue and maintain the coherency and structure of the web application.

The practical part describes the created solutions architectural design and business logic. Also, all the different components of the solution are described, both separately and together as a whole. In the centre of the solution, there is a cron job that works daily and handles the iteration of the data that need to be sent to Directo. In the iteration all the low-level actions are called. Those actions include the mapping of the data object from the Adcash web application specific format to Directos' format, parsing those created objects to an XML structure and finally sending that XML structure to Directo via Directos' API.

At the end of the thesis, there is small analysis of the results and possible further developments.

The thesis is in Estonian and contains 34 pages of text, six chapters, 12 figures, one table.

Lühendite ja mõistete sõnastik

ACID	*Atomicity, Consistency, Isolation, Durability*, atomaarsus, täielikkus, isoleeritus, püsivus
API	*Application Programming Interface*, rakendusliides
klientrakendus	*Client*, serverilt teenuseid saav funktsionaalüksus
Cron	UNIX-i käsk, ettemääratud tegumite perioodiliseks täitmiseks ettemääratud ajal
DBMS	*Database manaagement system*, andmebaasi haldussüsteem
DDD	*Domain-Driven Design*, domeenipõhine disain, tarkvara arhitektuuriline muster
domeeniobjekt	*Domain object*, objekt, mis vastab rakenduse andmebaasi tabelile
DRY	*Don't repeat yourself*, tarkvaraarenduse põhimõte, mis innustab vältima koodi kordusi
DSP	*Demand-side platform*, nõudluspoolne platvorm
eepos	*Epic*, nõudmiste kogum
ehitaja muster	*Builder pattern*, disainimuster, mille eesmärk on lihtsustada keeruliste objektide loomist
erind	*Exception*, tarkvara kontekstis erand ehk programmi täitmise ajal tekkida võiv olukord
erinditöötlus	*Exception handling*, erinditöötlus on konstruktsioon erinditega tegelemiseks
fassaadimuster	*Facade pattern*, struktuurikujunduse muster, mis komplekteerib keerulise äriloogika ühte arusaadavasse klassi
hoidla	*Repository*, tarkvarahoidla ehk tarkvarapakettide salvestuskoht
hoidla muster	*Repository pattern*, arendusmuster, mis rakendab hoidlaid
HTML	*HyperText Markup Language*, hüperteksti märgistuskeel
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastusprotokoll, protokoll teabe edastamiseks arvutivõrkudes
käitumismuster	*Behavioral pattern*, disainimuster, mis tuvastavad objektide vahel levinud suhtlemismustreid
loominguline muster	*Creational pattern*, disainimuster, mille eesmärk on lihtsustada ülemklassi pärivate objektide loomisprotsessi

MVC	*Model-View-Controller*, mudel-vaade-kontroller - tarkvaraline arhitektuuri muster
vaatleja muster	*Observer pattern*, käitumismuster, mis määratleb sõltuvuse objektide vahel, kus ühe muudatuse tulemusel teavitatakse kõiki tema vaatlejaid
PHP	*Hypertext Preprocessor*, mitmeotstarbeline programmeerimiskeel
SAFe	*Scaled agile framework*, organisatsiooni- ja töövoomustrite kogum
SQL	*Structured Query Language*, struktuurpäringukeel
SSP	*Supply-side platform*, pakkumispoolne platvorm
struktuurikujunduse muster	*Structural design pattern*, mis hõlbustavad disaini, tuvastades lihtsa viisi üksuste vaheliste suhete realiseerimiseks
sõltuvuse süstimine	*Dependency Injection*, struktuurikujunduse muster, mille puhul objekt võtab vastu muid objekte, millest see sõltub
tarbekeskkond	*Production environment*, tegelikke talitlusprotsesse toetav kogum
tagarakendus	*Backend*, rakenduse osa, mis töötab tagataustal
tegum	Ühest või mitmest käsust koosnev jada
tehase meetodi muster	*Factory method pattern*, tarkvaraline arhitektuuri muster
testimiskeskkonda	*Test environment*, testimisprotsesse toetav kogum
UNIX	Arvuti operatsioonisüsteem, mida arendati 1963-1969 aastatel
URL	*Uniform Resource Locatior*, internetiaadress
ühiktestimine	*Unit testing*, arvutiprogrammi väikseimate osade testimine
XML	*Extensible Markup Language*, laiendatav märgistuskeel

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Antud bakalaureusetöö kirjutamise hetkel töötab autor ettevõttes Adcash OÜ, mille tooteks on samanimeline ülemaailmne veebireklaamiplatvorm. Adcashi platvorm võimaldab reklaamijatel (*advertiser*) jõuda ülemaailmse vaatajaskonnani ning reklaami avaldajatel minimaalse pingutusega veebiliiklusest raha teenida. Adcashi platvormi kasutatakse läbi veebilehitseja.

Ettevõte Adcash OÜ kasutab reklaamijate andmete ning arvete haldamiseks tasulist Directo finantsarvestuse teenust, mis võimaldab mugavalt hallata kasutajate raamatupidamist. Eelnimetatud teenuse suurimaks puudjäägiks on manuaalne andmete sisestamine Directo keskkonda, mis on ajakulukas ning veaaldis. Töö kirjutamise hetkel peavad Adcashi finantsosakonna töötajad käsitsi sisestama arveid ning klientide informatsiooni. Sellest tulenevalt on otsustanud ettevõte kergendada finantsosakonna töökoormust ning automatiseerida see protsess.

Bakalaureusetöö eesmärk on luua tarkvaralahendus, mis automatiseerib Adcashi klientide andmete ning arvete edastamist Directo finantsarvestuse rakendusliidesele.

Töö on jaotatud neljaks sisupeatükiks. Esimeses neist kirjeldatakse detailsemalt lahendatavat probleemi ning oodatavat lahendust. Teises peatükis keskendutakse olemasoleva süsteemi kaardistamisele ning lahenduses kasutust leidvate tehnoloogiate kirjeldusele. Kolmandas peatükis keskendutakse nõuete ja lahenduse kirjeldamisele. Neljandas peatükis kirjeldatakse ja analüüsitakse töö tulemusi ning pakutakse võimalusi edasiarenduseks.

Lahendus valmib arvestades ettevõtte Adcash OÜ nõudeid ning olemasolevaid tehnilisi kriteeriumeid.

2 Probleemi püstitus ja projekti eesmärk

Alljärgnevas peatükis kirjeldatakse lähemalt lõputöös käsitletavat probleemi ning selle püstitust koos probleemi tausta ja lähtetingimustega. Samuti sõnastatakse projekti ning lõputöö eesmärgid, pannakse paika projekti skoop ning antakse ülevaade planeeritavast metoodikast.

2.1 Taust

Lõputöö loomise hetkel töötab autor ettevõttes Adcash OÜ. Adcash pakub samanimelist ülemaailmset digireklaamiplatvormi. Adcash keskendub reklaamilahenduste pakkumisele nii reklaami tellijatele ehk reklaamijatele, kui ka veebikeskkondade omanikele ehk publitseerijatele (*publisher*). Reklaamijad otsivad enda sihtrühmani jõudmiseks reklaamidele kõige sobivamaid veebilehti ning publitseerijad soovivad enda keskkonnas reklaame avalikustada. Sellest tulenevalt jagunebki Adcashi platvorm kaheks: DSP (*demand-side platform*) ehk nõudluspoolne platvorm ning SSP (*supply-side platform*) ehk pakkumispoolne platvorm [1]. Antud lõputöö keskendub platvormi DSP poolele.

Töö autor töötab ettevõtte tagarakenduse (*backend*) meeskonnas, kuhu kuuluvad koos autoriga kolm arendajat ning üks tootejuht. Autor saab ülesandeks välja mõelda ning arendada põhiline lahenduse äriloogika ning ülejäänud tiimiliikmed on nõustavas rollis, näiteks abistavad lahenduse loogika välja mõtlemisel ning viivad läbi rutiinseid koodi hindamised (*code review*).

2.2 Probleemi püstitus

Adcashi DSP kaudu on reklaamijatel võimalik osta publitseerijate pakutavat reklaamipinda. Kuna tegu on tasulise teenusega, on ettevõttes spetsiaalne finantsosakond, kes vastutab nii klientide kui ka firmasiseste finantside haldamise eest. Klientide arvete esitamiseks kasutatakse ettevõttes Directo OÜ pakutavat samanimelist finantsarvestuse teenust. Kuigi eelmainitud finantsarvestuse teenus aitab Adcashi finantsosakonnal aega

kokku hoida, tehes arvete vormistamise ning esitamise palju mugavamaks ja kiiremaks, on teenuse suurimaks puudujäägiks andmete edastamine DSP-st Directo keskkonda.

Hetkel peavad finantsosakonna töötajad manuaalselt sisestama arvete andmeid Directo keskkonda. Samuti on manuaalselt vaja uuendada arvete esitamiseks vajalikke klientide isiklikke andmeid, et need oleks identsed DSP-s olevate andmetega. Selline metoodika on veaaldis ning ajakulukas.

Eelpool mainitud asjaoludest lähtuvalt on antud bakalaureusetöö eesmärgiks kaotada vajadus manuaalselt sisestada klientide andmeid ning arveid Directo keskkonda.

2.3 Lähtetingimused

Loodav tarkvaralahendus peab sobituma ettevõtte olemasoleva koodivaramuga. Sellest tulenevalt on ettemääratud lahenduse tehnoloogia valik, milleks on programmeerimiskeel PHP (*Hypertext preprocessor*) koos Yii2 raamistikuga.

Lõputöö autori ning ettevõtte vahel sõlmitud konfidentsiaalsuse lepingu tingimustest lähtuvalt ei ole autoril õigust antud töös käsitleda klientide andmeid ega näidata tarkvaralahenduses esinevat konfidentsiaalset äriloogikat.

Kuna loodavast lahendusest ei tehta eraldiseisvat mikroteenust, vaid lahendus integreeritakse olemasolevasse privaatsesse koodivaramusse, ei sisalda antud bakalaureusetöö tervet loodavat lahendust. Antud töö sisaldab ainult katkendeid loodavast lahendusest, mis ei ole vastuolus eelmainitud konfidentsiaalsuse lepingu tingimustega. Samuti on autoril lubatud esitada nõudeid, kasutajalugusid ning teisi analüüsi ja töökäiku illustreerivaid andmeid, mis ei sisalda konfidentsiaalset informatsiooni.

2.4 Eesmärk

Lõputöö eesmärk on luua tarkvaralahendus Adcash OÜ jaoks, mis automatiseerib reklaamijate andmete edastamist Directo finantsarvestuse tarkvarasse. Lahendus peab iga päev automaatselt edastama nii uuendatud kasutajate andmed, kui ka loodud arved Directo finantsarvestuse tarkvarasse.

Loodav lahendus peab olema kooskõlas olemasoleva koodivaramu standardite ning stiiliga, et kindlustada tarkvara korrektne ning oodatav käitumine. Samuti aitab standardite ning stiili jälgimine parandada koodi loetavust ning arusaadavust, mis hõlbustab edaspidist koodiga töötamist, kui tulevikus tekib vajadus olemasolevat lahendust muuta või täiustada.

Bakalaureusetöö lõpptulemusena peab olema valminud lahendus, mis täidab lisaks eelpool mainitud nõuetele ka ettevõtte poolt püstitatud nõudeid – vt täpsemalt ptk 4.1, on läbinud kasutajaandmetega testimise, saanud arendajate, tootejuhi ning finantsosakonna heakskiidu ning olema kasutusel reaalsete kasutajate andmete ning arvete edastamiseks.

Lõputööga soovib autor anda ülevaate eelpool mainitud tarkvaralahenduse arendamisprotsessist ning lahendusest endast.

2.5 Skoop

Bakalaureusetöö skoobis on tarkvaralahenduse loomine, mis automatiseerib kliendiandmete saatmist Directo majandustarkvarasse, koos sellele eelneva analüüsiga. Lisaks kuulub skoopi olemasoleva süsteemi kaardistamine, nõuete kirjeldamine ning rakenduses kasutust leidvate tehnoloogiate tutvustus. Lähtuvalt antud lõputöö eesmärgist luua lahendus reklaamijate andmete ekspordi automatiseerimiseks, keskendub autor Adcashi rakenduse DSP poolele ning ettevõtte SSP poolt töös ei käsitleta. Rakenduse DSP poole detailset loogikat antud töös ei käsitleta, küll aga analüüsitakse olemasoleva süsteemi kaardistamise käigus rakenduse DSP poole arhitektuurilisi omadusi.

Töö skoopi ei kuulu loodud lahendusele automaattestide kirjutamine, kuna testide kirjutamise eest ei vastuta lõputöö autor, vaid teine arendusmeeskonna liige. Küll aga kirjeldatakse lühidalt testimismetoodikat.

2.6 Metoodika

Lõputöö teoreetilises osas keskendub autor olemasoleva süsteemi kaardistamisele ja analüüsimisele. Olemasolevas süsteemis kaardistatakse ja kirjeldatakse kasutust leidvaid arhitektuuri- ja arendusmustrid. Analüüsi tulemuste põhjal saab leida parima võimaliku viisi lõputöö probleemi lahendamiseks, järgides varem mainitud lähtetingimusi. Teoreetilises osas kaardistatakse ja kirjeldatakse ka rakenduses kasutust leidvaid

tehnoloogiad ning antakse lühiülevaade asjaoludest, miks just neid tehnoloogiaid rakenduses kasutatakse.

Lõputöö praktiline osa viiakse läbi SAFe (*Scaled Agile Framework*) agiilse arendusmissioonina. Agiilne arendusmetoodika muudab arendusmeeskonnad paindlikkumaks ja kohanemisvõimelisemaks, võimaldades meeskondadel ootamatustega paremini toime tulla. SAFe raamistik on organisatsiooni- ja töövoomustrite kogum, mis aitab agiilset lähenemist skaleerida ja organisatsiooni tasandil rakendada. Raamistik koosneb rollipõhiste tööülesannete ning üldisest töö planeerimise ja juhtimise juhendist [2].

Missiooni kollektiiv koosneb kolmest arendajast ning ühest tootejuhist. Missioon jaguneb omakorda kahe nädala pikkusteks tsükliteks ehk sprintideks. Sprindi alguses jagatakse arendajatele ülesanded, mis peaksid kahe nädala jooksul valmima, et kindlustada missiooni tähtajast kinnipidamine.

Missiooni esimeses faasis viiakse läbi analüüs kaasates kõik osapooled, et välja selgitada missiooni eesmärgid ning kriteeriumid. Pannakse paika täpsemalt milliseid andmeid on vaja edastada, mis formaadis, kui tihti jne. Samuti alustatakse suhtlust Directo OÜ-ga spetsiaalse API (*Application Programming Interface*) väljatöötamise osas, mis luuakse ettevõtte Adcash nõuete järgi. Eelnimetatud API võimaldab saata andmeid korrektses formaadis Adcashi platvormilt otse Directo keskkonda.

Teiseks faasiks on arendusfaas, kus luuakse esialgne töötav lahendus, mille eest vastutab lõputöö autor. Lisaks praktilisele arendustööle ja testimisele, kuuluvad arendusfaasi skoopi koodi hindamised, mille käigus tiimiliikmed hindavad ning analüüsivad teiste liikmete koodi, eesmärgiga seda paremaks teha. Samuti viiakse läbi igapäevaselt püstijalakoosolekuid, kus arendajad jagavad enda tööülesannete arengut, mis võimaldab aegsasti teavitada kõiki tiimiliikmeid võimalikest väljakutsetest ning probleemidest.

Kolmandas faasis toimub lahenduse esitlemine. Demonstratsiooni käigus näidatakse kaasatud osapooltele esmast töötavat lahendust, eesmärgiga koguda tagasisidet. Vajadusel luuakse saadud tagasiside põhjal uued kasutajalood ja ülesandepiletid ning jätkatakse arendamisega, kuni kõik puudujäägid saavad likvideeritud.

Viimaks toimub põhjalik testimine ning avalikustamine, kus lahendust kasutatakse esialgu testandmetega, et veenduda korrektses käitumises ning vajadusel tehakse viimased parandused enne lahenduse avalikku kasutuselevõttu.

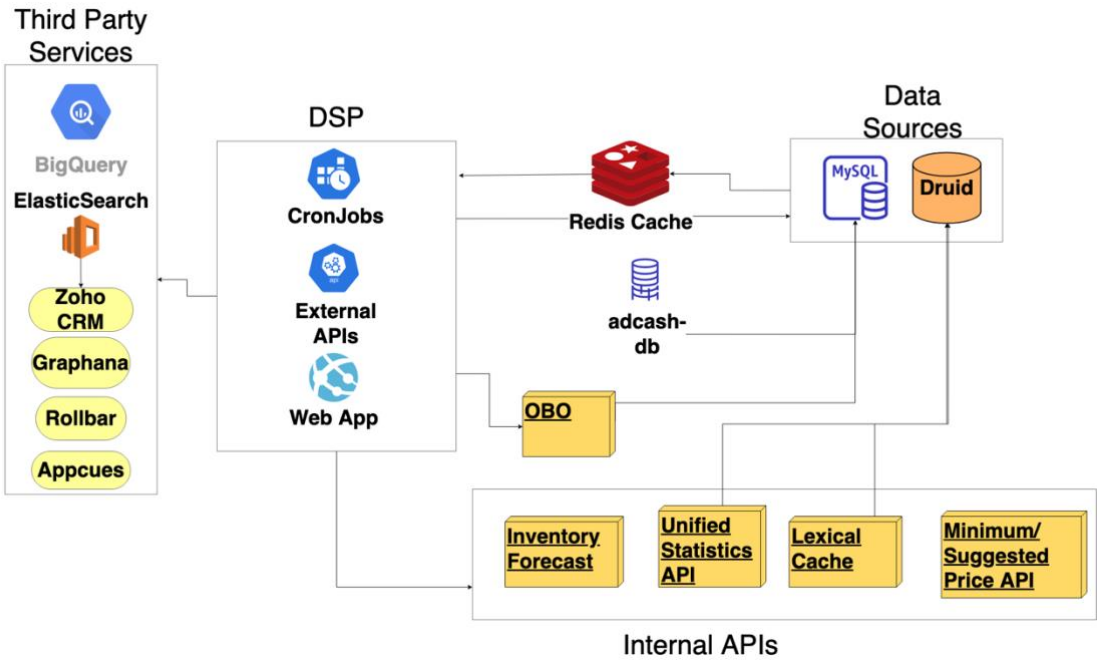
3 Analüüs

Alljärgnevas peatükis keskendutakse olemasoleva süsteemi kaardistamisele. Lisaks räägitakse lähemalt lahenduses kasutustleidvatest tehnoloogiatest.

3.1 Olemasoleva süsteemi kaardistamine

Esmajoones on mõistlik analüüsida olemasolevat süsteemi ning leida süsteemi iseärasused. Nende leidude põhjal saab formuleerida lõputöös käsiteldavale probleemile parima võimaliku lahenduse, mis sujuvalt ühilduks olemasoleva koodibaasiga.

Adcashi reklaamiplatvormi rakendus koosneb mitmetest erinevatest firmasisestest ja -välistest osadest, näiteks kolmandate osapoolte teenused, firmasisesed API-d ja erinevad andmebaasid. Kõik need osad on hädavajalikud ning moodustavad ühe terviku, mille keskmes on juba varem mainitud DSP. DSP ise koosneb erinevatest cronidest - vt täpsemalt ptk 4.2.6, välistest API-dest ning kasutajate jaoks tähtsaimast osast ehk veebirakendusest, mille kaudu kliendid saavad Adcashi teenust kasutada. Allpool on toodud skeem (joonis 1) terve Adcashi rakenduse arhitektuuri kohta.



Joonis 1. Adcash reklaamiplatvormi rakenduse arhitektuur.

3.1.1 Arhitektuur

Tarkvaraarhitektuur viitab tarkvarasüsteemi struktuuri ja süsteemi osade loomise distsipliinile. Piltlikult on tegu süsteemi abstraktse joonisega, mis määrab ära süsteemi komponentide omavahelise suhtluse ning koordineerimise. Korrektse arhitektuuri valiku tegemine on äärmiselt oluline protsess, eriti ambitsioonikamate projektide jaoks. Hea valik suurendab tõenäosust, et süsteemi valmimisel on olemas võimekas ning töökindel lahendus. Arhitektuuri valiku puhul tuleb kaaluda näiteks plaanitava süsteemi suurust ja keerukust, aga tuleb ka arvestada arendajate kogusega ehk kui oluline on meeskondade vaheline koostöö [3], [4].

DSP rakenduse arhitektuuri iseloomustab modulaarne monoliit arhitektuur ning domeenipõhine disaini. Järgnevas alapeatükis kirjeldatakse just neid DSP rakenduse arhitektuuri omadusi.

Modulaarse monoliidi arhitektuur

Olemasolevas süsteemis kasutatakse modulaarset monoliit arhitektuuri. Monoliitsüsteem kirjeldab tarkvara kui ühe-astmelist rakendust, kus kõik erinevad komponendid moodustavad ühe terviku [5].

Monoliitrakendused on loomulik viis, kuidas rakendused arenevad. Enamik rakendusi luuakse ühe konkreetse eesmärgiga. Aja jooksul tekivad uued ärivajadused ning nende rahuldamiseks hakatakse esialgsesse rakendusse uusi funktsionaalsuseid lisama. Rakenduse elutsükli alguses on selline lähenemine edukas, töö käib kiirelt ja sujuvalt. Küll aga rakenduse kasvades võib tekkida olukord, kus rakenduse haldamine ja edasine arendamine on paljude erinevate loogikate sidestuse (*coupling*) tõttu liiga keeruline [6].

Modulaarse monoliidi arhitektuur (*modular monolithic architecture*) on monoliit arhitektuuri ümberkorraldus. Eesmärk on säilitada monoliitsüsteemile omane sujuv komponentidevaheline suhtlus, hoides kõik komponendid ühes koodibaasis, aga vähendada koodibaasis esinevat koodi sidestust. Tarkvaratehnikas mõõdab sidestus, kui tihedalt erinevad rakenduse komponendid üksteisest sõltuvad. Rakenduse modulaarseks tegemine võimaldab sidestuse vähendamist. Antud kontekstis tähendab modulaarsus rakenduse tükeldamist väikesteks osadeks ehk mooduliteks, millest igaüks käsitleb erinevat äriloogikat. Eesmärk on luua keskkond, milles on madal sidestus, aga kõrge sidusus (*low coupling, high cohesion*) [7], [8].

Modulaarse monoliidi eelised standardse monoliit arhitektuuri suhtes on [9]:

- Vähendatud keerukus – iga moodul on ainult seotud koodiga mis on tema funktsioneerimiseks hädavajalik.
- Kergem refaktoreerida – mooduli muutmisel mõju teistele moodulitele on väiksem, või puudub üldse.
- Mugavus meeskondadele – arendajatel on kergem samaaegselt töötada koodibaasi erinevate osadega.

Domeenipõhine disain

DDD (*Domain-Driven design*) ehk domeenipõhine disain on lähenemine tarkvaraliste rakenduste mõistmiseks, disainimiseks ning ehitamiseks. DDD lähenemise keskpunktis on domeen ehk antud kontekstis äriloogika. Sellest tulenevalt on DDD eesmärk äriloogika kontrollimine ning eraldamine ülejäänud rakenduse osadest [9].

Kuna DDD hõlmab endas väga palju erinevaid valdkondi, keskendub bakalaureusetöö autor ainult DDD-stiilis andmebaasiga suhtlusele, sest just seda lähenemist on kasutatud Adcashi DSP rakenduses. Edaspidi kutsutakse DDD-stiilis andmebaasiga seotud klasse olemiklassideks.

DDD kohaselt peavad olemiklassid täielikult valdama enda olemit ning sellest sõltuvaid olemeid. Olemiklassides määratakse kõik omadused (*property*) privaatseteks, ehk neile ei ole võimalik klassiväliselt ligi pääseda. Sellest tingituna tuleb olemiklassis kasutada ainult konstruktorit ja meetodeid, et andmeid luua, muuta või kustutada. Nii saavutataksegi olukord, kus kogu andmebaasi olemiga seotud loogika ning valideerimine koondatakse olemiklassi. Nõrkuseks DDD olemiklasside puhul on korrektseks töötamiseks vajav mahukas koodihulk, küll aga pikas perspektiivis tasub see ennast ära [9].

Olemiklasside kasutamise eelised on [9]:

- Kogu olemiga seotud loogika on ühes kohas – lihtsustab suurest koodibaasist klasside leidmist. Lisaks rakendab selline lähenemine automaatselt DRY (*Don't repeat yourself*) põhimõtet, ehk väldib koodi kordumist.

- Olemite muutmiseks kasutatakse ühte klassi koos selgelt sõnastatud ning eraldatud meetoditega – igal meetodil on üks kindel funktsionaalsus, näiteks olemi muutmine, lisamine või kustutamine.

- Andmebaasi sisestatud andmed ei saa olla vales olekus – kuna üks kindel klass vastutab andmete andmebaasi sisestamise eest, läbivad kõik andmed täpselt sama kontrolli. Ehk olukorras, kus olemi klassile on koostatud korrektne valideerimine, on minimaalne tõenäosus, et andmebaasi satuvad ebakorrektsed andmed.

3.1.2 Arendusmustrid

Objektorienteeritud tarkvaraarenduse kontekstis on mustrid selgelt määratletud abstraktsed lahendused mingitele levinud probleemidele. Mustrid ei ole täielikud lahendused, mida on võimalik otse töötavaks programmiks muuta, vaid kirjeldused või mallid probleemi lahendamiseks. Arendusmustrite teadmine aitab arendajal ära tunda üldlevinud probleeme, samuti muudab mustrite kasutamine koodi arusaadavamaks ka teiste mustreid tundvate arendajate jaoks. Eriti otstarbekas on kasutada arendusmustreid suuremates projektides, mille kallal töötavad paljud arendajad [10], [11].

DSP rakenduses kasutatakse mitmeid arendusmustreid: tehase meetodi muster (*Factory method pattern*), ehitaja muster (*Builder pattern*), vaatleja muster (*Observer pattern*), fassaadimuster (*Facade pattern*), hoidla muster (*Repository pattern*). Allpool on toodud tabel (tabel 1) eelmainitud arendusmustritest ja nende kasutusviisidest DSP rakenduses.

Tabel 1. Arendusmustrite kasutamine DSP rakenduses.

Arendusmuster	**Kasutusviisid**
Tehase meetodi muster	Maksete ja arvete loomiseks, kuna neid on mitu tüüpi.
Ehitaja muster	Keerukate objektide loomiseks, näiteks arved ja klientide profiilid.
Vaatleja muster	Logide loomiseks ja salvestamiseks. Väliste teekidega sünkroniseerimiseks.
Fassaadimuster	Keerukate andmestruktuuride, mitmetest erinevatest objektidest koosnevate komplektide koopiate tegemine. Üldiselt väga levinud muster keerulise äriloogika peitmiseks.
Hoidla muster	Iga andmebaasiolemi jaoks on hoidla klass.

Järgnevalt kirjeldatakse täpsemalt tabelis 1 loetletud ja DSP rakenduses leiduvaid arendusmustreid.

Tehase meetodi muster

Tehase meetodi muster on loominguline muster (*Creational pattern*), mis kasutab tehase meetodeid objektide loomiseks. Objektide loomine hõlmab tihti keerukaid protsesse, mida on ühe koodibaasi raames vaja kasutada mitmes kohas, selline käitumine viib koodi korduseni, mis raskendab rakenduse haldamist ning arendamist. Tehase meetodid võimaldavad alamklassidel luua objekte, ilma et peaks täpsustama loodava objekti täpset klassi. Antud mustriga saab ühtlustada sarnaste objektide loomise loogika, määrates loomise vastutuse ühele ülemklassile, muutes koodi lihtsamini hallatavamaks. Objekti loomise protsess algab alamklassis, kui kutsutakse välja tehase meetod, andes parameetriks mõne objekti või alamklassi ise. Parameetrite põhjal loob ja tagastab tehase meetod enda loogika põhjal uue objekti [12].

Tehase meetodi muster on võimalik tänu objektide pärimise omadusele. Pärimise käigus pärib alamklass omadused ja käitumised oma ülemklassilt. Päritud omadustele ja käitumistele lisaks saavad alamklassid luua enda ülemklassist sõltumatuid omadusi ning käitumisi. Pärimine võimaldab luua spetsiifilisteks olukordadeks ühe klassi eeskujul mitmeid erinevaid versioone, säilitades vajaliku ühisosa ning ülemklassi algupärase struktuuri [13]. Allpool on toodud näidis (joonis 2) tehase meetodi kohta, mis on autori poolt loodud lahenduse arendamise käigus.

```
class DirectoResourceFactory
{
    public static function getResource(AbstractDirectoRecord
$directoRecord): AbstractDirectoResource
    {
        if ($directoRecord instanceof DirectoCustomerRecord) {
            return make(DirectoCustomerResource::class);
        } elseif ($directoRecord instanceof DirectoInvoiceRecord) {
            return make(DirectoInvoiceResource::class);
        } else {
            $modelClass = get_class($directoRecord);
            throw new RuntimeException("Could not find directo
resource for class {$modelClass}");
        }
    }
}
```

Joonis 2. Tehase meetodi näide.

Ehitaja muster

Ehitaja muster on samuti loominguline muster, mille eesmärk on lihtsustada keeruliste objektide korduvat loomist. Selle asemel, et luua objekt läbi klassi konstruktori ning andes sellele konstruktorile kõik soovitud parameetrid, luuakse üks ehitaja klass, millel on selgelt nimetatud meetodid iga parameetri määramiseks. Meetoditega objektide loomine teeb objektide loomise koodist loetavamaks, seda eriti keeruliste ning mitmete võimalike parameetrite kombinatsioonidega klasside puhul. Eriti kasulik on ehitaja klass juhul, kui soovitakse korduvalt luua klassi objekti, aga erinevate parameetritega, sellisel juhul ei pea arendaja jälgima konstruktori parameetrite järjekorda, vaid saab selgelt sõnastatud meetodeid välja kutsuda [14].

Fassaadimuster

Fassaadimuster on struktuurikujunduse muster (*Structural design pattern*), mis toimib liidesena (*interface*), mis varjab keerukamat alus- või struktuurikoodi. Eesmärk on teha rakenduse keerukas äriloogika nii loetavamaks, kui ka arusaadavamaks, aga ka võimaldada seda keerukat loogikat ümberkirjutamata taaskasutada. Fassaadi saab kasutada nii rakenduse sisese äriloogika peitmiseks, kui ka väliseteegi lihtsustamiseks. Selle asemel, et kasutada kõiki teegi komponente, luuakse fassaadi klass ning selles kasutatakse ainult neid teegi osasid, mis on tõesti vajalikud. Fassaadimustriga peab olema ettevaatlik, et klass, mille eesmärk oli äriloogika komplekteerimine, ei muutuks äriloogika lisamisel liiga pikaks ja keeruliseks [15].

Vaatleja muster

Vaatleja muster on käitumismuster (*Behavioral pattern*), mis määratleb sõltuvuse objektide vahel, kus ühe muudatuse tulemusel teavitatakse kõiki tema vaatlejaid. Selle asemel, et üks klass peab teise käest konstantselt ja manuaalselt pärima või üks klass peab tervet rakendust konstantselt teavitama enda muutuste kohta, võimaldab vaatleja muster teavitada ning kuulata ainult neid klasse mis seda vajavad. Lisaks korrektsele ja sujuvamale rakenduse toimimisele, ehk ainult neid klasse teavitatakse, kes seda vajavad, muudab vaatleja muster ka rakenduse koodi loetavamaks ning hallatavamaks, määrates terve teavitamise loogika ühe klassi kohustuseks [16]. Allpool on toodud näidis (joonis 3) vaatleja mustrit rakendava klassi kohta, mis kutsutakse välja, kui antud klassi rakendava klassi eksemplari uuendatakse või luuakse. Näide on autori loodud, lahenduse arendamise käigus.

```
class DirectoSyncBehavior extends Behavior
{
    /**
     * BaseActiveRecord events list to bind to
     * @var array
     */
    public $events = [
        BaseActiveRecord::EVENT_AFTER_INSERT,
        BaseActiveRecord::EVENT_AFTER_UPDATE
    ];

    public function attach($owner)
    {
        if (!($owner instanceof DirectoSyncableInterface)) {
            $exception = "{$owner->className()} is not
supported to sync with Directo";
            throw new RuntimeException($exception);
        }
        parent::attach($owner);
    }

    public function handleEvent(AfterSaveEvent $event)
    {
        make(DirectoSyncInterface::class)->addObjectToQueue(
            $this->owner,
            SyncDspToDirectoAction::UPSERT
        );
    }
}
```

Joonis 3. Vaatleja mustri klassi näide.

Hoidla muster

Hoidla muster ühtsustab andmebaasiga suhtlemisega seotud klassid. Hoidlad (*repository*) on klassid, mis vastutavad teatud klassi andmete salvestamise, uuendamise ja kustutamise eest andmebaasis. Mustri kasumlikkus ilmneb suuremate ning mitmete andmebaasiolemitega rakendustes, väiksemates rakendustes kaotab hoidlate kasutamine oma kasumlikkuse ning mustri rakendamine ainult suurendab koodibaasi keerukust. Mustri eesmärk on andmebaasiga suhtlemise abstraheerimine, määrates hoidlate klassidele samad meetodid, et ühtsustada rakenduse loogika, ning tsentraliseerimine, eraldades andmebaasiga suhtlemise ülejäänud rakendusest. Lisaks suurendavad hoidlad rakenduse testimise võimalusi, lihtsustades andmebaasiga seotud toimingute ühiktestimist [17], [18]. Allpool on näide (joonis 4) hoidla kohta. Näide on autori loodud, lahenduse arendamise käigus.

```
class SyncDspToDirectoRepository extends AbstractRepository
{
    /**
     * Check if queue item with given data exists
     */
    public function queueItemExists(string $primary, string $action,
string $modelClass): bool
    {
        return SyncDspToDirectoData::find()
            ->andWhere(['primary' => $primary])
            ->andWhere(['action' => $action])
            ->andWhere(['model' => $modelClass])
            ->andWhere(['is_failed' => 0])
            ->exists();
    }

    /**
     * Get all untried queue items, sorted from newest to oldest
     */
    public function getQueueItems(int $limit): array
    {
        return SyncDspToDirectoData::find()
            ->andWhere(['is_failed' => 0])
            ->orderBy(['id' => SORT_ASC])
            ->limit($limit)
            ->all();
    }
}
```

Joonis 4. Hoidla näidis.

S.O.L.I.D

Kõiki varem kirjeldatud arendusmustrid rakendavad ning sunnivad arendajaid jälgima S.O.L.I.D printsiipe.

S.O.L.I.D on akronüüm viiele objektorienteeritud programmeerimise printsiibile, mis näitavad kuidas jagada meetodeid ja andmestruktuure klassidesse ning kuidas neid klasse omavahel siduda [19].

S.O.L.I.D põhimõtted on järgmised [19]:

- Ainsa vastutuse printsiip (*Single-responsiblity principle*) – ühel klassil peaks olema üks konkreetne ülesanne. Kui klassil on rohkem, kui üks ülesanne, võivad seda klassi muutes tekkida konfliktid teiste klasside ja nende ülesannetega.

- Avatud/suletud printsiip (*Open/closed principle*) – klassid tuleb struktureerida selliselt, et neid saab vajadusel uue funktsionaalsusega laieneda. Samas seda osa koodist, mis on juba kirjutatud, ei tohiks kunagi muuta, välja arvatud vigade parandamiseks.

- Liskovi asendamise printsiip (*Liskov substitution principle*) – alamklassid tuleb kirjeldada nii, et objekti kasutaja jaoks ei ole vahet kui kasutusse antakse alamklassi tüüpi objekt. Kui alamklass teeb midagi sellist, mida ülemklassist ei eeldaks, on see printsiibi rikkumine.

- Liidese eraldamise printsiip (*Interface-segregation principle*) – liidese kasutaja ei pea sõltuma meetoditest, mida otseselt vaja ei lähe. Liidesed, kus on palju meetodeid, jagatakse väiksemateks.

- Sõltuvuse inversiooni printsiip (*Dependency-Inversion principle*) – tarkvarakomponendid peaks sõltuma abstraktsioonidest, mitte konkreetsetest klassidest.

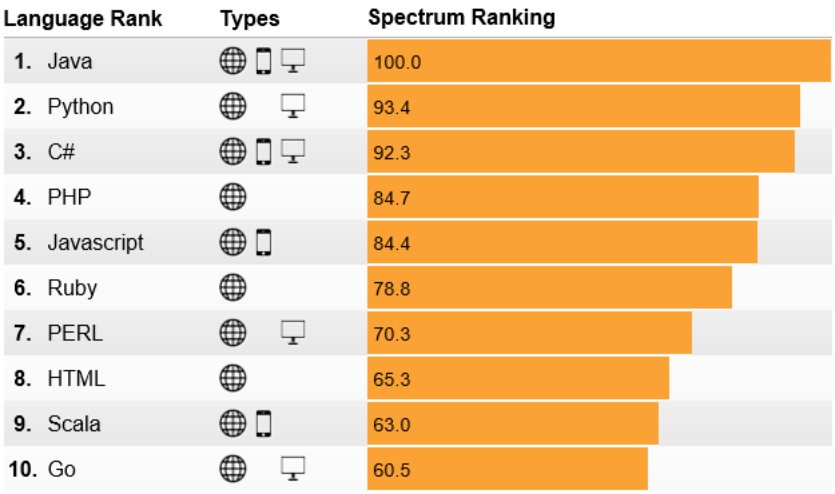
DSP rakenduses jälgitakse mainitud mustreid ja S.O.L.I.D printsiipe.

3.2 Rakenduses kasutust leidvad tehnoloogiad

Viimaks tuleks kaardistada rakenduses kasutust leidvaid tehnoloogiaid, mis vastavalt varem mainitud lähtetingimustele on autori tööandja poolt ette määratud. Arvestades, et DSP projekti planeerimist alustati 2014. aastal ning arendust 2015. aastal, on enamus tarkvara valikutest tehtud selle ajastu trende ning standardeid järgides.

PHP

Ettevõtte IT-osakond otsustas DSP süsteemi ehitada kasutades PHP programmeerimiskeelt, valik tehti selle populaarsuse ning IT-osakonna töötajate varasema kogemuse tõttu. Allpool on toodud edetabel (joonis 5) 2014. aasta populaarsemate veebirakenduste programmeerimiskeelte kohta, tulemused baseeruvad ajakirja IEEE Spectrum poolt läbi viidud uuringul.



Joonis 5. 2014. aasta populaarseimad veebirakenduste programmeerimiskeeled [20].

PHP on 1994. aastal loodud mitmeotstarbeline programmeerimiskeel, mis on kujunend üheks enimlevinud serverrakenduste loomise keeleks. PHP on üks esimestest serverrakendustele mõeldud programmeerimiskeeltest mida on võimalik kirjutada HTML-i (*HyperText Markup Language*) koodiplokkide vahele, mis võimaldab vähese vaevaga muuta staatilise veebilehe funktsionaalseks. Lisaks keele võimekustele, tagab PHP populaarsuse selle lihtsus ning tal on laialdane teekide, andmebaaside ja operatsioonisüsteemide tugi. Viimati uuendati PHP versiooni 2020. aasta novembris, mis kinnitab, et PHP on endiselt aktiivse kasutajabaasiga programmeerimiskeel [21].

Yii2

Lisaks keele valikule, tuli ka langetada otsus raamistiku osas. Tarkvararaamistikud on koodi kogumikud, mis lihtsustavad arendajate tööd, võimaldades arendamist alustada juba olemasolevast rakendusest, millel on kõige algelisemad funktsionaalsused olemas, näiteks veebirakenduse jaoks vajalikud kasutajate autentimine ja andmebaasiga ühendamise võimekus. Lisaks annavad raamistikud juurde osalise abstraktsiooni, määrates õrnad piirangud, mis sunnivad arendajaid jälgima teatud standardeid või arendusmustreid [22].

Arvestades raamistikuga kaasnevaid mugavusi, otsustasid Adcashi arendajad kasutada objektorienteeritud Yii2 raamistikku - selle populaarsuse, jõudluse ning põhjaliku dokumentatsiooni tõttu. Allpool on toodud graafik (joonis 6) kolme PHP raamistiku populaarsuse kohta aastatel 2013 kuni 2015, otsingumootor Google andmete põhjal.

Joonis 6. PHP raamistike populaarsus 2013 – 2015.aastal, otsingumootori Google kohaselt [23].

Yii2 on 2008. aastal avalikustatud Yii-nimelise raamistiku teine verisoon, mille esmane stabiilne verisoon avalikustati 2014. aastal [24]. Yii on veebirakenduse loomiseks mõeldud raamistik. Tänu oma komponendipõhisele arhitektuurile ja headele vahemälu haldamise võimalustele, sobib Yii raamistik just suuremahuliste projektide jaoks, näiteks foorumite, veebipoodide, sisuhaldussüsteemide jaoks. Lisaks paljudele funktsionaalsustele mida raamistik pakub, rakendab Yii2 arhitektuurimustrit MVC (*Model-View-Controller*), mille eesmärk on eraldada koodi komponendid nende funktsionaalsuste ning kohustuste põhjal [25].

3.3 Rakenduses kasutatav andmebaas

Andmebaas on korrastatud infokogum, mida üldiselt hoiustatakse elektroonilisel kujul arvutites. Andmebaasi haldamiseks kasutatakse DBMS-i (*Database management system*) ning andmebaasi ja DBMS-i kombinatsioon moodustab andmebaasisüsteemi, mida tihtipeale kutsutaksegi lihtsalt andmebaasiks. Andmebaasisüsteemid aitavad rakenduses kasutatavaid andmeid talletada, mis on hädavajalik, et kindlustada rakenduse ootuspärane käitumine [26].

Andmebaase jaotatakse oma struktuuri järgi erinevatesse kategooriatesse, mida kasutatakse vastavalt rakenduse vajadustele. Enimlevinud on relatsioonilised andmebaasid. Relatsioonilised andmebaasid on kasutusel olnud juba 1970-ndate aastate algusest ning nende vastupidavus, jõudlus ja töökindlus on nende populaarsust säilitanud. Relatsioonilisi andmebaase iseloomustab andmete struktureeritus omavahel seoses

olevates tabelites ning ACID (*Atomicity, Consistency, Isolation, Durability*) omadused andmebaasi transaktsioonides [27].

ACID omadused on järgmised [28]:

- Atomaarsus (*Atomicity*) – andmetehing peab olema jagamatu (põhimõttel „kõik või mitte midagi“) tegevuste jada, mis õnnestub ja täidetakse või nurjub ning tühistatakse täielikult, taastades andmetehingu eelse olukorra andmebaasis.

- Täielikkus (*Consistency*) – tegevuste jada peab viima andmebaasi sisu ühest täielikust olekust teise. See tähendab, et hiljemalt andmetehingu kinnitamise hetkeks pole andmetehing rikkunud ühtegi andmebaasi piirangut.

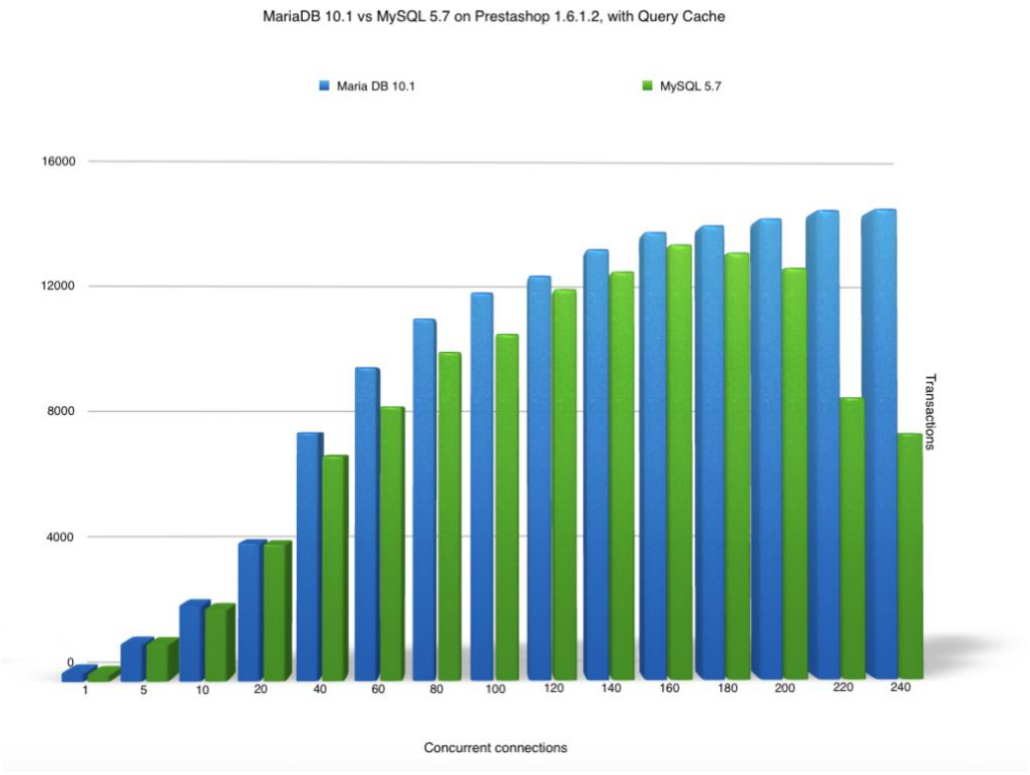
- Isoleeritus (*Isolation*) – andmetehingu sisesed tegevused peavad olema varjatud teiste samaaegselt toimuvate andmetehingut eest.

- Püsivus (*Durability*) – teostatud tegevused püsivad andmebaasis isegi võimalike süsteemi rikete korral.

MariaDB

DSP projekti andmete salvestamiseks otsustati kasutada relatsioonilist andmebaasi, kuna kavandatava projekti jaoks olid vajalikud omavahel seoses olevad andmed ning paljudel tarkvararaamistikel on suurem tugi relatsioonilistele andmebaaside jaoks, kui mitterelatsioonilistele. Täpsemalt otsustati kasutada MariaDB andmebaasi, mis on avatud lähtekoodiga relatsioonilise andmebaasi haldamise süsteem, mille esmaväljalase oli 2009. aastal. MariaDB on loodud MySQL-i andmebaasi põhjal ehk nad omavad väga palju sarnasusi [29].

Otsus kasutada MariaDB-d ja mitte teisi relatsioonilise andmebaase haldamise süsteeme, nagu MySQL või PostgreSQL, tugines MariaDB kiirusel ning jõudlusel. Lisaks, tänu Adcashi arendajate varasemale MySQL-i andmebaasi kogemusele, oli MariaDB-ga harjumine sujuv ning kiire. Allpool on toodud graafik (joonis 7) MySQL ja MariaDB jõudluse võrdluse kohta 2015. aastal.



Joonis 7. MySQL ja MariaDB jõudluse võrdlus 2015. aastal [30].

SQL

SQL (*Structured Query language*) on programmeerimiskeel, mida kasutatakse relatsiooniliste andmebaaside haldamiseks, mis loodi juba 1970ndate aastate alguses koos relatsiooniliste andmebaasidega. SQL-is kirjutatud koodijuppi nimetatakse päringuks (*query)*, päringuga saab näiteks andmeid lisada, muuta, kustutada ja kuvada. Lisaks saab keerukamate tehingute jaoks mitu päringut omavahel siduda, näiteks selleks, et kuvada korraga mitmest tabelist andmeid. Kuigi SQL on standardiseeritud alatest 1986. aastast, muudavad osad DBMS-id enda süsteemi jaoks SQL-i süntaksit, eesmärgiga seda paremaks teha või lisada uut standardivälist funktsionaalsust [31].

4 Lahenduse loomine

Järgnevas peatükis keskendutakse lõputöö probleemi lahenduse arendusprotsessi kirjeldamisele, lisaks kirjeldatakse lahenduse nõudeid.

4.1 Nõuded lahendusele

Loodava lahenduse nõuete kogunemisel kasutatakse SAFe raamistiku metoodikat, mis on kirjeldatud peatükis 2.6.

Alguses pannakse paika eepose (*epic*) hüpotees, seejärel defineeritakse eepose tehniline skoop. Lõpuks pannakse paika aktsepteerimiskriteeriumid (*acceptance criteria*) ehk nõuded.

Antud juhul on hüpotees: andmete ekspordi automatiseerimine vähendab finantsosakonna manuaalse töö hulka ja tehniline skoop: andmete saatmine ja pärimine.

Nõuded koostavad Adcashi töötajad: lõputöö autori meeskonda kuuluv tootejuht ning firmasisese finantsosakonna töötajad. Nõuete kogumiseks peetakse koosolekuid ning kogutud materjali põhjal koostatakse aktsepteerimiskriteeriumid. Kriteeriumite põhjal koostatakse eepos, mis omakorda lammutatakse väiksemateks osadeks ehk kasutajalugudeks, mis kirjeldavad iga funktsionaalsuse erinevaid aspekte.

Kogutakse nii funktsionaalsed kui mittefunktsionaalseid nõudeid. Funktsionaalsed nõuded defineerivad lahenduse kindlaid käitumisi või tegevusi. Funktsionaalsed nõuded peavad vastama küsimusele mida süsteem tegema peab, mitte aga täpselt selgitama kuidas süsteem neid tegevusi võimaldab. Mittefunktsionaalsed nõuded on funktsionaalsete nõuete vastandid, mis keskenduvad süsteemi käitumise täpsustamisele, mitte konkreetse funktsionaalsuste või omaduste kirjeldamisele [32].

Koostatud funktsionaalsed nõuded on järgmised:

- Uued arved edastatakse koos kasutajate andmete ja muude andmeridadega automaatselt Directo majandustarkvarasse.

- DSP ja Directo andmetüüpide vahel on loodud vasted.
- Klientide bilanss Directo tarkvaras uueneb automaatselt.
- Eksisteerib võimalus pärida andmeid Directo tarkvarast, eesmärgiga neid võrrelda DSP rakenduses olevate andmetega.
- Arved, mille andmeid on muudetud peale Directo tarkvarasse edastamist, tuleb märgistada.

Mittefunktsionaalsed nõuded on järgmised:

- Andmeid peab uuendama kord ööpäevas.
- Lahendus peab olema testitud, enne reaalset kasutuselevõttu.

Firmasiseselt koostatud ingliskeelne eepos on välja toodud lisas 2.

Esimeseks ja tähtsaimaks nõudeks on uute arvete ja kliendiandmete automaatne edastamine Directo majandustarkvarasse. Nõude täitmiseks on vaja arendada tarkvaralahendus, mis rutiinselt edastab kõik uued ja uuendatud andmed automaatselt Directo majandustarkvarasse. Enne saatmist tuleb DSP andmeid töödelda. Selleks, et kindlustada andmete korrektne vormistus, tuleb teisendada DSP rakenduse sisesed kasutajate ning arvete andmed Directo OÜ API jaoks sobivasse XML (*Extensible Markup Language*) formaati. Samuti on nõue, et kasutajate bilanss Directo keskkonnas muutuks vastavalt arvete andmetele, aga selle nõude rahuldab korrektsete andmete edastamine, bilansi muutuse eest vastutab juba Directo tarkvara ise.

Teiseks tähtsamaks nõudeks on võimalus Directo keskkonnast andmeid pärida ning neid võrrelda Adcashi DSP rakenduses olevate andmetega. See funktsionaalsus on vajalik, et veenduda Directo keskkonnas olevate andmete õigsuses. Kui tekib olukord, et DSP andmed ja Directost päritud andmed ei ühti, siis DSP-s olevad andmed loetakse automaatselt õigeteks ning muuta tuleb Directo keskkonnas olevaid andmeid.

Viimaseks nõudeks on märgistada DSP keskkonnas arved, mille detaile on muudetud peale nende Directosse edastamist. Sellisel juhul saavad finantsosakonna töötajad käsitsi erinevused likvideerida.

4.2 Lahenduse kirjeldus

Lahenduse eesmärk on automatiseerida reklaamijate andmete ja arvete edastamist Directo majandustarkvarasse, samaaegselt järgides analüüsi peatükis tuvastatud DSP rakenduse arhitektuuri ja tarkvaralisi standardeid.

4.2.1 Mooduli struktuur

DSP rakenduses kasutatava modulaarse monoliitarhitektuuri järgimiseks komplekteeritakse kogu loodav lahendus uude moodulisse. Piltlikult on moodul lihtsalt rakenduse failisüsteemis olev kaust, kuhu koondatakse kogu lahenduseks vajaminev kood. Mooduli kaust jaotatakse omakorda alamkaustadeks, kuhu lisatakse spetsiifilisemad failid.

Lahenduse directo-nimelisse mooduli kausta (joonis 8) kuuluvad:

- Klientrakenduse kaust (*client*), mis sisaldab Directo API-ga suhtlemiseks vajalikke komponente, näiteks HTTP (*Hypertext Transfer Protocol*) päringute saatmise eest vastutavat klassi.

- Komponentide kaust (*components*) sisaldab andmete töötlemisega seotud klasse, näiteks DSP kliendiandmete Directo API jaoks vajalikku XML formaati teisendamise eest vastutatav klass.

- Konfiguratsiooni kaust (*config*) hõlmab endas Yii2 raamistiku spetsiifiliseid mooduliga seotud konfiguratsioonifaile.

- „cronjobs“ nimelisse kausta läheb cron ehk *Cron job*, teisisõnu ülesannete ajastaja, mis ettemääratud ajal edastab uued andmed Directo tarkvarasse.

- „intercom“ nimeline kaust sisaldab konfiguratsiooni faili, mis seadistab mooduli sõltuvuste süstimise (*Dependency Injection*).

- Mudelite (*models*) kaust sisaldab andmebaasiga suhtlemiseks vajalikke olemite klasse ning ka Directo API teel edastatavate objektide klasse.

- Viimaseks kaustaks on testide kaust (*tests*), mis sisaldab äriloogikaga seotud ühikteste. Lisaks on mooduli juurkaustas testide käivitamiseks vajalik codeception.yml fail.

- Mooduli juurkaustas on ka Module.php fail, mis on mooduli sisenemispunkt ja alus, mille kaudu saavad moodulivälised klassid moodulisisestele klassidele ligi.

Joonis 8. Mooduli failistruktuur.

4.2.2 Andmebaas

DSP rakenduses salvestatakse kõik reklaamijate andmed ja arved vastavatesse andmebaasi tabelitesse. Antud lahenduse jaoks on vajalik meeles pidada andmeid, mida on vaja Directole edastada, selle asemel, et kohe andmete loomisel või uuendamisel neid edastada. Kohene saatmine koormaks rakendust ning teeks selle kasutamise aeglasemaks. Autori arvates on parimaks lahenduseks luua uus andmebaasi tabel kuhu saatmist vajavad andmed sisestada, sellisel juhul ei muudeta rakenduses eksisteerivat loogikat ega andmebaasi tabeleid. Kuna andmete enda hoiustamiseks juba eksisteerivad andmebaasi tabelid, siis andmeid mitu korda salvestada ei ole mõtet. Selle asemel peaks salvestama tabelisse andmetele viitavad unikaalsed võtmed (*foreign key*), mille kaudu hiljem saab pärida vajalikud andmed. Piltlikult saab uut tabelit nimetada ka järjekorraks (*queue*), kuna uut informatsiooni sinna ei lisata, vaid kasutatakse olemasolevate andmetele viitavaid võtmeid, et neid meeles pidada. Kuna ootamatud erindind (*exception*) ja süsteemi vead on alati võimalikud, peab tabeliga olema võimalus järge pidada ka nende andmete üle, mille esmane saatmine ebaõnnestus.

Sellest tulenevalt luuakse tabel nimega *sync_dsp_to_directo*, koos järgmiste veergudega:

- *id* - automaatselt genereeritud rea unikaalne identifikaator.
- *primary* - edastatava kliendi või arve unikaalne identifikaator.
- *model* - edastatavate andmete tüüp - kliendiandmed või arve.
- *action* – tegevuse tüüp, kas lisatakse uusi andmeid või uuendatakse olemasolevalid.
- *additional_data* – kommentaar lisatavatele andmetele.
- *is_failed* – identifikaator, mille järgi saab ära märkida andmed, mille esmane Directosse saatmine ebaõnnestus.
- *created_at* – rea lisamise kuupäev.
- *updated_at* – rea muutmise kuupäev, enamjaolt muutub see kuupäev kui andmete saatmine ebaõnnestus.

Lisaks loodud andmebaasi tabelile, luuakse DSP rakendusse loodud tabeli kasutamise jaoks domeeniobjekt (*Domain object*) *SyncDspToDirectoData* ning hoidla *SyncDspToDirectoRepository*. Sufiksid *Data* ja *Repository* on terves DSP rakenduses kasutuses, et defineerida domeeniobjekte ja hoidlaid. Igale andmebaasitabeli kirjele vastab üks domeeniobjekt, millel on väljad ja väärtused on identsed kirjega. Domeeniobjektide kasutamine muudab andmebaasiga töötamise objektorienteeritumaks, järgides terve rakenduse objektorienteeritud lähenemist. Andmebaasi tabeliga seoseid ei looda.

4.2.3 Directo API-ga suhtlemine

Kuna loodav lahendus hakkab Directo OÜ poolt koostatud API vahendusel andmeid DSP-st Directo majandustarkvarasse saatma, peab DSP-s olema võimalus Directo API-ga suhtlemiseks.

Directo OÜ töötas välja Adcash OÜ jaoks, ettevõtte nõuete järgi kaks API-t. Üks testimiskeskkonnaks (*Test environment*) ning teine tarbekeskkonnas (*Production environment*), kummagi API-ga ühendamiseks on spetsiaalne URL (*Uniform Resource*

Locator) ja identifitseerimiseks vajalik salavõti. Vältimaks igal ühendusel või keskkonnavahetusel käsitsi URL-i või salavõtme sisestamist, salvestatakse URL ja salavõti mooduli konfiguratsiooni kaustas asuvasse faili, lisaks salvestatakse mõlemasse arenduskeskkonna spetsiifilisse konfiguratsiooni faili vastavad Directo API URL ja salavõti. Mõlemad Directo API-d kasutavad andmete edastamiseks XML märgistuskeelt, millest tulenevalt, edastas teenusepakkuja ka iga andmetüübi spetsiifilise domeeniobjekti struktuuri XML-is. Kliendiandmete XML struktuur on välja toodud lisas 3.

Arenduse mugavdamiseks ja koodikorduse vältimiseks, komplekteeritakse kõik API-ga suhtlemiseks vajalikud klassid mooduli kliendirakenduse kausta.

Kliendirakenduse kausta peamine fail on *Client.php*, kuhu koondatakse Directo API-ga ühenduse loomine ning algelised pärimise meetodid. *Client* klassi rakendatakse *AbstractDirectoResource* klassis, mis on abstraktne klass, mille eesmärk on tegeleda spetsiifilise Directo andmete saatmisega. *AbstractDirectoResource* klassi rakendab kaks klassi: *DirectoCustomerResource*, mis tegeleb kliendiandmete saatmisega ning *DirectoInvoiceResource*, mis tegeleb arvete saatmisega. Lisaks on loodud tehase mustri rakendamiseks klass *DirectoResourceFactory*, mille üheainsa meetodi eesmärk on tagastada parameetrina antud objektile vastav *resource*-tüüpi klass. Viimaks on klass *ResponseHelper*, mis tegeleb Directo API päringult saadud vastuse töötlemisega. Päringu õnnestumise korral tagastab meetod saadud XML-formaadis vastuse sisu. Vastasel juhul tegeleb klass erinditöötlemisega (*Exception handling*) ning tagastab, kas üldise *DirectoRequestException* või spetsiifilise, ebaedukal Directo-poolsel päringu autoriseerimisele ilmnenud vea puhul, *DirectoUnauthorizedException*-tüüpi erindi. Kliendirakenduse klassi *Client.php* programmikood on välja toodud lisas 4.

4.2.4 Domeeniobjektid

Mooduli domeeniobjektid ja andmebaasiga seotud klassid koondatakse *models* kausta.

Directo majandustarkvarasse hakatakse edastama reklaamijate andmeid, millele pääseb DSP rakendusest ligi läbi *AdvertiserData* domeeniobjekti, ning reklaamijate arveid, mille domeeniobjekt on *AdvertiserInvoiceData*. Kuna reklaamijate ja nende arvetega on seotud ka väga palju muud äriloogikat DSP rakenduses, on domeeniobjektidel palju omadusi, mida Directo XML-formaadis olevates objektides ei ole. Selle tõttu ei saa otse DSP domeeniobjekte XML-i teisendada ja Directosse saata, vaid neid peab enne teisendama.

Andmete edastamiseks luuakse eraldi *resource*-sufiksiga objektid, mille väljadeks on varem mainitud XML-formaadis Directo domeeniobjektidelt võetud väljad. Kasutajate andmete jaoks luuakse *DirectoCustomerRecord* ning arvete haldamiseks objektid *DirectoInvoiceRecord*, mis omab arve detaile, ning *DirectoInvoiceRowRecord*, mis omab ühe arverea detaile. Ühe arvega ehk *DirectoInvoiceRecord*-iga saab olla seotud mitu arverida ehk *DirectoInvoiceRowRecord*-it. Klassidele abstraktsiooni lisamiseks luuakse *AbstractDirectoRecord*, mida rakendavad kõik eelmainitud *resource* klassid. Antud olukorras võimaldab lisatud abstraktsioon rakendada liideseid ja tehase meetodeid, et erinevate *resource*-tüüpi objektidega kergemini ümber käia. Eelmainitud klasside failid koondatakse *external* alamkausta. Allpool on toodud *DirectoInvoiceRecord* programmikood (joonis 9).

```
class DirectoInvoiceRecord extends AbstractDirectoRecord
{
    /** @var string */
    public $number;

    /** @var DateTimeImmutable */
    public $date;

    /** @var string */
    public $paymentTerm;

    /** @var string */
    public $currencyCode;

    /** @var string */
    public $customerCode;

    /** @var bool */
    public $isPrepayment;

    /** @var string */
    public $creditInvoiceNumber;

    /** @var DirectoInvoiceRowRecord[] */
    public $rows = [];
}
```

Joonis 9. *DirectoInvoiceRecord* klass.

Mainitud domeeniobjektid rakendavad *DirectoSyncableInterface*-nimelist liidest. Lisaks abstraktsioonile, võimaldab antud olukorras liides eristada domeeniobjekte, mis ei ole Directosse saatmiseks kõlblikud. Eristamine on hiljem vajalik, et kohe andmetöötluse alguses tagastada erind, kui mingil põhjusel üritatakse Directole saata andmetüüpe, millel puudub Directo-poolne toetus, näiteks publitseerijate andmetel või arvetel.

Domain alamkausta koondatakse DDD-stiilis olemi klass *DirectoSync*. *DirectoSync* klassil on üks globaalne meetod *addObjectToQueue*. Meetodis kontrollitakse, kas vastava andmetüübi ja *primary* identifikaatori kombinatsiooniga kirje eksisteerib andmebaasis, kui jah, siis uut kirjet ei lisata, vastasel juhul luuakse uus *SyncDspToDirectoData* objekt ja andmebaasi lisatakse vastavate andmetega kirje. Kui uue objekti või kirje loomisel tekib erind, tagastab meetod spetsiifilise *DirectoSyncException*-tüüpi erindi. Allpool on toodud programmikood (joonis 10) *addObjectToQueue* meetodist.

```
public function addObjectToQueue(string $primary, string $action,
string $modelClass, array $additionalData = []): void {
    if ($this->queueItemExists($primary, $action,$modelClass){
        return;
    }

    $syncDspToDirectoData = make(SyncDspToDirectoData::class);
    $syncDspToDirectoData->setAttributes([
        'primary' => $primary,
        'action' => $action,
        'model' => $modelClass,
        'additional_data' =>
$additionalData ? json_encode($additionalData) : null
    ]);

    if (!$syncDspToDirectoData->save()) {
        throw new DirectoSyncException(
          'Directo sync queue save: ' .
$syncDspToDirectoData->getErrorFirst()
        );
    }
}
```

Joonis 10. *AddObjectToQueue* meetod.

Viimaks on *models* kaustas hoidla nimega *SyncDspToDirectoRepository*. Antud hoidla vastutab erinevate *SyncDspToDirectoData* domeeniobjekti pärimistega seotud meetodite eest, näiteks meetod, mis kontrollib, kas vastavate parameetritega domeeniobjekt eksisteerib või mitte.

4.2.5 Andmete teisendamine

Peale Directo domeeniobjektidele vastavate klasside loomist, tuleb DSP rakenduse domeeniobjektid ka vastavateks klassideks ümber teisendada. Lisaks tuleb leida viis objektide teisendamiseks XML-kujule, sest Yii2 raamistik vaikimist sellist teisendamist ei võimalda.

Autori arvates oleks parim lahendus klasside teisendamiseks luua eraldi klassid, mille ainukeseks eesmärgiks saab teisendada DSP domeeniobjekt vastavaks Directo objektiks. Vastavatele klassidele lisatakse *transformer*-sufiks. Luuakse liides *DirectoTransformerInterface*, milles defineeritakse üks meetod *transform*, mille parameetriks on *DirectoSyncableInterface* liidest rakendav objekt, ning mis tagastab *AbstractDirectoRecord* liidest rakendava objekti.

Peale Directo objektide loomist, on vaja teisendada need objektid XML-kujule, et neid saaks Directo API vahendusel saata. Selle jaoks on vaja spetsiaalset parserit, mis suudaks objekti teisendada XML-kujule. Probleemi lahendamiseks võetakse kasutusele juba valmis vabavaraline lahendus. Objekti teisendamiseks XML-kujule, on see esmalt vaja teisendada assotsiatiivseks massiiviks ehk võti-väärtus paaridest koosnevaks massiiviks. Selle massiivi suudab vabavaraline XML-parser korrektsesse XML-formaati teisendada.

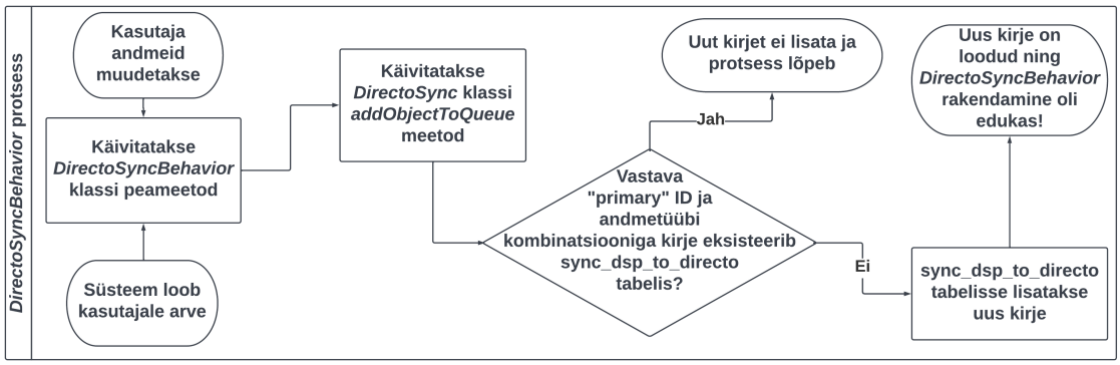
Tingituna asjaolust, et antud bakalaureusetöö lahenduse valmistamisel oli kaasatud terve arendusmeeskond, ei tegelenud töö autor spetsiifiliste domeeniobjektide teisendamise ega XML-formaati teisendamise realiseerimisega, vaid vastutas mõlemale probleemile algelise ja abstraktse lahenduse loomise eest. Selle tõttu erinevate andmetüüpide teisendamiste detailseid lahendusi antud lõputöös ei käsitleta.

4.2.6 Automatiseerimine

Selleks, et automatiseerida kogu Directo tarkvarasse edastamise protsess, tuleb automatiseerida nii *SyncDspToDirectoData* domeeniobjektide loomine kui ka andmete saatmine.

Esimese sammuna tuleb automatiseerida *SyncDspToDirectoData* objektide loomine, sest nagu varem mainitud, domeeniobjektide loomisel lisatakse automaatselt ka vastavate andmetega kirje andmebaasi ning nende kirjete alusel hakataksegi andmeid Directole edastama. Üks variant selle saavutamiseks oleks koodibaasist otsida üles kõik olukorrad, kus uuendatakse reklaamijate andmeid või luuakse uusi arveid ning täiustada sealset loogikat, kutsudes *DirectoSync* klassi *addObjectToQueue* meetodit. Kuigi selline lahendus oleks kergesti tehtav, ei ole see skaleeritav. Igal juhul, kui koodibaasi lisatakse uus koht, kus uuendatakse reklaamijate andmeid või luuakse arveid, peab meeles pidama ka *SyncDspToDirectoData* loomise loogikat, ja selline lahendus ei ole jätkusuutlik.

Paremaks lahenduseks otsustab autor luua vaatleja mustrit rakendava *DirectoSyncBehavior* klassi, kuhu koondada *SyncDspToDirectoData* loomise loogikaga seotud meetodite kutsumine. Kuna Yii2 raamistikuga on *behavior*-tüüpi klasside rakendamine väga lihtne ning sama metoodikat on ka kasutatud mujal DSP rakenduses. *Behavior*-klassi lisamiseks domeeniobjektile, tuleb see lisada domeeniobjekti klassis olevasse *behavior*-klassidest koosneva massiivi hulka ning selle tulemusena kõik selle klassi objektid omavad sõltuvust *DirectoSyncBehavior* klassist. *DirectoSyncBehavior* pikendab Yii2 raamistikus leiduvat *Behavior*-klassi, mis lisab kogu vaatlemiseks vajaliku loogika. Uues klassis määratakse sündmused (*event*), mille korral *behavior* klassi peameetod käivitatakse, antud olukorras on nendeks domeeniobjekti loomine ja uuendamine, ning funktsionaalsusena lisatakse klassi meetod, mis omakorda käivitab *DirectoSync*-klassi *addObjectToQueue* meetodit. Allpool on toodud voodiagramm (joonis 11) *DirectoSyncBehavior* loogika kohta.



Joonis 11. *DirectoSyncBehavior* voodiagramm.

Viimaks tuleb omavahel siduda ja automatiseerida kogu andmete saatmise protsess ning võttes arvesse DSP rakenduses kasutust leidvaid metoodikaid ja lähenemisi, otsustab autor selle saavutamiseks kasutada cron tööriista. Cron on UNIX-laadsetes süsteemides, nagu Linux ja macOS, leiduv tööriist, mida kasutatakse ettemääratud tegumite perioodiliseks täitmiseks ettemääratud ajal. Üldiselt kasutatakse cron-e andmete varundamiseks (b*ackup*), vananenud andmete kustutamiseks, süsteemi haldamiseks ja muudeks süsteemiülema käskude täitmiseks [33], [34].

Valik on tingitud asjaolust, et tööriista cron on kasutatud ka muude rakenduses leiduvate tegumite, ehk ühest või mitmest käsust koosnevate jadade, automatiseerimiseks.

Luuakse cron nimega *SyncDspToDirectoJob*, mille loogika on järgmine:

1. Cron käivitatakse automaatselt ettemääratud ajal. Käivitades kasutatakse sõltuvuse süstimist, et edastada cron-ile vajalik *DirectoTransformerFactory* klass, millel on tehase meetod *transformer*-tüüpi klasside loomiseks.

2. Järgmiseks päritakse *SyncDspToDirectoRepository*-st maksimaalselt sada kirjet, mida hakatakse Directosse edastama. Päritud kirjed defineeritakse *SyncDspToDirectoData* domeeniobjektide massiivina. Tänu sellisele defineerimisele, suudab Yii2 raamistik automaatselt teisendada andmebaasist päritud andmeread domeeniobjektideks.

3. Peale pärimist hakatakse for-tsüklit kasutades domeeniobjekte läbi käima. Tsükli käsiteldav loogika pannakse erinditöötlemiseks *try-catch* ploki sisse. Kui Directo API-ga suhtlemiseks loodud klientrakendus tagastab *DirectoUnauthorizedException* erindi, siis terve protsessi töötamine peatatakse ning cron käivitatakse käsitsi peale probleemiga tegelemist uuesti. Kui erind tagastatakse muul juhul, tsükkel jätkub.

4. Tsüklist võetud domeeniobjektiga käivitatakse cron-i privaatne meetod *handleAction*, mille parameetriteks on *action*, domeeniobjektilt saadud tegevuse tüüp, ning *directoSyncable*, edastavate andmete tüüp, kas kasutajaandmed või arve.

    - *handleAction* meetod esmalt kasutab *DirectoTransformerFactory*-t, et luua edastavate andmete tüübi järgi õige *transformer*.

    - Seejärel kasutatakse *DirectoResourceFactory*-t, et luua õige andmetüübiga *resource*-tüüpi klass, mis vastutab andmete edastamise eest.

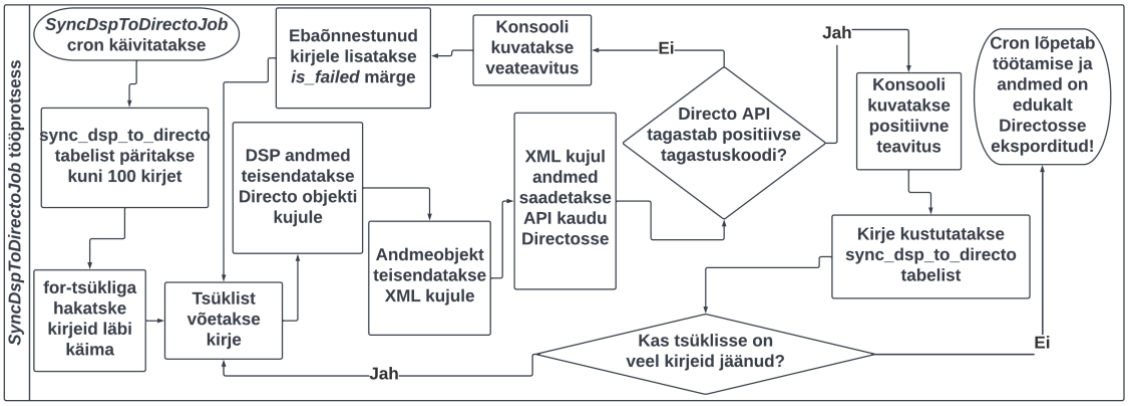
    - Viimaks kutsutakse vastava *resource*-klassi õige meetod välja, mis valitakse *handleAction* meetodile kaasa antud parameetri *action* järgi.

5. Kui vastava meetodi käivitamine oli edukas ning ühtegi erindit ei tagastatud, kustutatakse käsiteldava *SyncDspToDirectoData* domeeniobjektile vastav andmebaasi kirje ning iteratsioonist võetakse järgmine domeeniobjekt ning korratakse eelmainitud, neljandat, sammu.

Vastupidisel juhul tegeletakse erindiga *catch* plokis, kus määratakse domeeniobjekti *is_failed* väli tõeseks, kuvatakse konsooli vastav ebaõnnestumisele viitav sõnum ning võetakse iteratsioonist järgmine domeeniobjekt ja alustatakse taas 4. sammuga.

6. Kui kõik objektid on läbi töödeldud, kuvatakse konsooli tegumi lõppemisele viitav sõnum ning cron lõpetab töötamise.

Croni *SyncDspToDirectoJob* programmikood on välja toodud lisas 5 ning voodiagramm (joonis 12) on esitatud allpool.



Joonis 12. *SyncDspToDirectoJob* voodiagramm.

4.2.7 Testimine

Lisaks manuaalsele testimisele, otsustas arendusmeeskond loodud lahenduse testimiseks kasutada ühiktestimise (*Unit testing*) metoodikat.

Ühiktestimine on tarkvara testimise liik, kus tarkvara komponente testitakse individuaalselt. Individuaalse testimise eesmärk on kindlaks teha, et iga üksik tarkvara komponent töötaks, teisi komponente arvestamata, ootuspäraselt. Üldiselt kirjutatakse ühikteste arendamise käigus, peale esimeste komponentide valmimist [35].

Peamised põhjused ühiktestimiseks on [35]:

- Ühiktestid aitavad arendustsükli varajases faasis vigu tuvastada.
- Aitavad mõista komponendi loogikat.
- Ühiktestid võivad osaliselt täita ka dokumentatsiooni rolli.

Tingituna asjaolust, et antud bakalaureusetöö lahenduse valmistamisel oli kaasatud terve arendusmeeskond, ei tegelenud töö autor lahenduse jaoks testide kirjutamisega. Seetõttu antud lahenduse jaoks kirjutatud testide detaile antud töös ei käsitleta.

5 Tulemused

Lõputöö tulemusena valmis ettevõtte nõuetele vastav tarkvaralahendus, mis automatiseerib reklaamijate andmete ning arvete edastamist Directo majandustarkvarale. Lahendus on avalikult kasutusele võetud ning töötab automaatselt, välist abi vajamata.

Lahenduse loomise käigus esines ka suurem tagasilöök. Esimesel tervikliku lahenduse testimisel ning demonstratsioonil finantsosakonnale, selgus, et klientide andmed ei jõua Directo teenusele. Selgus, et läbirääkimistel teenusepakkujaga oli tekkinud arusaamatus ning arendusmeeskonnale loodud API-l puudus võimekus klientide andmeid saata. Selle tagasilöögi tõttu lükkus lahenduse avalik kasutuselevõtt mitu kuud edasi. Kuigi Directo parandas omapoolsed puudused väga kiiresti, oli samaaegselt töö autorile tekkinud uued ning tähtsamad töökohustused, mis ei võimaldanud autoril lõputöö lahendust edasi arendada.

Edasisteks tegevusteks oleks loodud lahenduse rakendamine teiste Adcashi klientide, ehk reklaami avaldajate jaoks, kelle arvete haldamiseks kasutatakse samuti Directo finantsarvestuse teenust. Reklaami avaldajad jäid esialgse lahenduse skoobist välja kuna reklaami avaldajate arvete haldamise loogika erineb reklaamijatest ning kõiki reklaami avaldajate andmeid ei ole vajagi Directosse sisestada.

Lisaks saaks edasiarendusena loodud lahendusele üldisemaid teste kirjutada, mis testiks lahenduse terviklikku käitumist. Suurem ning põhjalikum testide hulk tõstaks arendusmeeskonna enesekindlust loodud lahenduse osas.

6 Kokkuvõte

Lõputöö eesmärk oli luua ettevõtte Adcash OÜ veebirakendusse lahendus, mis automatiseeriks ettevõtte klientide andmete ning arvete edastamise Directo majandustarkvarale. Andmete edastamise automatiseerimine kaotaks ära vajaduse ettevõtte finantsosakonnal klientide andmeid ning arveid käsitsi Directo majandustarkvarasse sisestada. Saadud ajavõit võimaldaks töötajatel keskenduda teistele, tähtsamatele tööülesannetele.

Töö käigus kaardistati ettevõtte veebirakenduse arhitektuuri- ja arendusmustrid ning selles kasutatavad tehnoloogiad, et leida parim võimalik lahendusmetoodika probleemi lahendamiseks ning samaaegselt säilitades eksisteeriva veebirakenduse terviklikkuse. Lisaks analüüsiti ning kaardistati lahenduse nõuded, mille koostasid ettevõtte töötajad. Olemasoleva süsteemi kaardistamine oli väga kasulik ja kergendas oluliselt lõpliku lahenduse välja mõtlemist.

Lõputöö tulemusena loodi terviklik lahendus, mis salvestab uued andmed, teisendab need Directo majandustarkvara jaoks sobivale kujule ning lõpuks edastab need andmed Adcashi veebirakendusest Directo majandustarkvale. Kogu eelmainitud protsess käivitatakse automaatselt kord ööpäevas.

Ettevõtte finantsosakond, kelle palvel lahendus loodi, jäid loodud lahendusega rahule, tänu millele pole neil enam vaja käsitsi reklaamijate andmeid ja arveid Directo majandustarkvarasse sisestada. Lahendusest saadud ajaline võit on keskmiselt tund tööpäevast ja andmete korrektsus on manuaalse sisestamisega võrreldes samuti tõusnud. Siiski igaks juhuks kontrollitakse aegajalt Directo tarkvaras olevaid andmeid, et veenduda nende korrektsuses.

Edasiarenduseks saaks loodud lahenduse põhjal automatiseerida ka ettevõte teiste klientide, reklaami avaldajate, arvete edastamist Directo majandustarkvarale. Samuti saaks loodud lahendusele üldisemaid teste kirjutada, mis kontrolliks lahenduse terviklikku käitumist.

Kasutatud kirjandus

[1] „Adcash,“ [Võrgumaterjal]. Available: https://www.businessofapps.com/ads/adcash/. [Kasutatud 13.03.2022].

[2] „What is safe,“ [Võrgumaterjal]. Available: https://www.atlassian.com/agile/agile-at-scale/what-is-safe. [Kasutatud 24.04.2022].

[3] D. H. F. Rizal, „Software Architecture and Design,“ [Võrgumaterjal]. Available: https://medium.com/the-legend/software-arsitektur-dan-design-90f76288078e. [Kasutatud 31.03.2022].

[4] „What Is Software Architecture?,“ [Võrgumaterjal]. Available: https://www.castsoftware.com/glossary/what-is-software-architecture-tools-design-definition-explanation-best. [Kasutatud 31.03.2022].

[5] S. u. Haq, „Introduction to Monolithic Architecture and MicroServices Architecture,“ [Võrgumaterjal]. Available: https://medium.com/koderlabs/introduction-to-monolithic-architecture-and-microservices-architecture-b211a5955c63. [Kasutatud 31.03.2022].

[6] „Are monolithic software applications doomed for extinction?,“ [Võrgumaterjal]. Available: https://nortal.com/blog/are-monolithic-software-applications-doomed-for-extinction/. [Kasutatud 31.03.2022].

[7] „Programming coupling,“ [Võrgumaterjal]. Available: https://blog.ndepend.com/programming-coupling/. [Kasutatud. 31.03.2022].

[8] P. Gupta, „Understanding the modular monolith and its ideal use cases,“ [Võrgumaterjal]. Available: https://www.techtarget.com/searchapparchitecture/tip/Understanding-the-modular-monolith-and-its-ideal-use-cases. [Kasutatud 31.03.2022].

[9] J. P. Smith, „My experience of using modular monolith and DDD architectures,“ [Võrgumaterjal]. Available: https://www.thereformedprogrammer.net/my-experience-of-using-modular-monolith-and-ddd-architectures/. [Kasutatud 31.03.2022].

[10] „Design patterns,“ [Võrgumaterjal]. Available: https://sourcemaking.com/design_patterns. [Kasutatud 30.03.2022].

[11] „What's a design pattern?,“ [Võrgumaterjal]. Available: https://refactoring.guru/design-patterns/what-is-pattern. [Kasutatud 30.03.2022].

[12] „Factory method,“ [Võrgumaterjal]. Available: https://refactoring.guru/design-patterns/factory-method. [Kasutatud 29.03.2022].

[13] „Inheritance in Object Oriented Programming for Python – An In-Depth Guide for Everyone,“ [Võrgumaterjal]. Available: https://www.analyticsvidhya.com/blog/2020/10/inheritance-object-oriented-programming/. [Kasutatud 30.03.2022].

[14] „Builder pattern,“ [Võrgumaterjal]. Available: https://refactoring.guru/design-patterns/builder. [Kasutatud 29.03.2022].

[15] „Facade pattern,“ [Võrgumaterjal]. Available: https://refactoring.guru/design-patterns/facade. [Kasutatud 29.03.2022].

[16] „Observer pattern,“ [Võrgumaterjal]. Available: https://refactoring.guru/design-patterns/observer. [Kasutatud 29.03.2022].

[17] P.-E. Bergman, „Repository design pattern,“ [Võrgumaterjal]. Available: https://medium.com/@pererikbergman/repository-design-pattern-e28c0f3e4a30. [Kasutatud 29.03.2022].

[18] J. Petrakovich, „Why should you use the repository pattern,“ [Võrgumaterjal]. Available: https://makingloops.com/why-should-you-use-the-repository-pattern/. [Kasutatud 29.03.2022].

[19] S. Oloruntoba, „SOLID: The First 5 Principles of Object Oriented Design,“ [Võrgumaterjal]. Available: https://www.digitalocean.com/community/conceptual_articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design. [Kasutatud 31.03.2022].

[20] K. Kunal, „Top Programming Languages in 2014 – IEEE Spectrum’s Ranking,“ [Võrgumaterjal]. Available: https://superdevresources.com/top-programming-languages-2014/. [Kasutatud 03.28.2022].

[21] S. Mino, „8 Reasons Why PHP Is Still So Important for Web Development,“ [Võrgumaterjal]. Available: https://www.jobsity.com/blog/8-reasons-why-php-is-still-so-important-for-web-development. [Kasutatud 30.03.2022].

[22] A. Spittel, „What is a Web Framework, and Why Should I use one?,“ [Võrgumaterjal]. Available: https://welearncode.com/what-are-frontend-frameworks/. [Kasutatud 28.03.2022].

[23] „PHP framework Google trends,“ [Võrgumaterjal]. Available: https://trends.google.com/trends/explore?date=2013-01-01%202015-12-31&q=yii,laravel,symfony. [Kasutatud 28.03.2022].

[24] „History of Yii,“ [Võrgumaterjal]. Available: https://en.rmcreative.ru/blog/the-history-of-yii-framework/. [Kasutatud 28.03.2022].

[25] „What is Yii,“ [Võrgumaterjal]. Available: https://www.yiiframework.com/doc/guide/2.0/en/intro-yii#what-is-yii. [Kasutatud 03.29.2022].

[26] „What is a database,“ [Võrgumaterjal]. Available: https://www.oracle.com/database/what-is-database/. [Kasutatud 30.03.2022].

[27] „What is a relational database,“ [Võrgumaterjal]. Available: https://www.oracle.com/database/what-is-a-relational-database/. [Kasutatud 30.03.2022].

[28] M. Laiho, D. A. Dervos ja K. Silpiö, SQL andmetehingud - õppetöö juhend, DBTech VET Teachers project, 2013.

[29] „MariaDB ametlik leht,“ [Võrgumaterjal]. Available: https://mariadb.org/about/. [Kasutatud 30.03.2022].

[30] J. Fournier, „softizy,“ [Võrgumaterjal]. Available: https://www.softizy.com/blog/mariadb-10-1-mysql-5-7-performances-ibm-power-8/. [Kasutatud 30.03.2022].

[31] „SQL definition,“ [Võrgumaterjal]. Available: https://www.techtarget.com/searchdatamanagement/definition/SQL. [Kasutatud 30.03.2022].

[32] „Tarkvara arendusnõuded,“ [Võrgumaterjal]. Available: https://web.htk.tlu.ee/digitaru/testimine/chapter/tarkvara-arendusnouded/. [Kasutatud 24.04.2022].

[33] „A beginners guide to cron jobs,“ [Võrgumaterjal]. Available: https://ostechnix.com/a-beginners-guide-to-cron-jobs/. [Kasutatud 01.04.2022].

[34] „Cron definitsioon,“ [Võrgumaterjal]. Available: https://akit.cyber.ee/term/8853-cron. [Kasutatud 04.04.2022].

[35] T. Hamilton, „Unit testing tutorial,“ [Võrgumaterjal]. Available: https://www.guru99.com/unit-testing-guide.html. [Kasutatud 07.04.2022].

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

Mina, Siim Mõtshärg

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

16.05.2022

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

Lisa 2 – Firmasisene eepose kirjeldus

Back-end Services and APIs (BSA)

bsa1-pi4. Automatic advertiser data export from Admin to Finance tool (Directo)

Mission type: ENABLER

Risk Matrix Level: **Bucket #1**

Hypothesis	Automated data transfer would reduce the manual work of the finance department.
Scope definition	PUT data (we send to Direct): • Invoice • Invoice items • Customer details GET data (We can check what data is saved in Direct) • Invoice, invoice items • Customer details
Acceptance criteria	• New invoices are sent to the DIRECTO automatically together with customer information and invoice items • Created mapping between Admin DB and Direct elements (invoice types, used payment methods, VAT regions, etc) • Customer balances updated in Directo accordingly • Matching Adcash and Directo data by using GET data (for validation and verification of data) • Changes highlighted to finance department for manual corrections (for example invoice details changed after it was already sent to Directo).

Lisa 3 – Directo kliendiandmete XML struktuur

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
          xmlns:sql="urn:schemas-microsoft-com:mapping-schema">
  <xsd:element name="customers"  sql:is-constant="true">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="customer" sql:relation="in_2_kliendid"
maxOccurs="unbounded"  minOccurs="0" >
          <xsd:complexType>
            <xsd:attribute name="session_id"  sql:field="x"
type="xsd:string"/>
             <xsd:attribute name="code"  sql:field="kood" type="xsd:string"/>
             <xsd:attribute name="name"  sql:field="nimi" type="xsd:string"/>
             <xsd:attribute name="object"  sql:field="objekt"
type="xsd:string"/>
             <xsd:attribute name="address1"  sql:field="aadress1"
type="xsd:string" />
             <xsd:attribute name="address2"  sql:field="aadress2"
type="xsd:string" />
             <xsd:attribute name="address3"  sql:field="aadress3"
type="xsd:string" />
             <xsd:attribute name="country"  sql:field="asumaa"
type="xsd:string"/>
             <xsd:attribute name="contact"  sql:field="kontakt"
type="xsd:string"/>
             <xsd:attribute name="email"  sql:field="email"
type="xsd:string"/>
             <xsd:attribute name="regno"  sql:field="regnr" type="xsd:string"
/>
             <xsd:attribute name="vatregno"  sql:field="kmregnr"
type="xsd:string" />
             <xsd:attribute name="type"  sql:field="klient_tyyp"
type="xsd:int" />
             <xsd:attribute name="vatzone"  sql:field="maa" type="xsd:int" />
             <xsd:attribute name="dealtype"  sql:field="tehinguliik"
type="xsd:string" />
             <xsd:attribute name="destination"  sql:field="sihtriik"
type="xsd:string" />
             <xsd:attribute name="vatcode"  sql:field="kmkood" type="xsd:int"
/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Lisa 4 – Client.php programmikood

```
class Client extends BaseObject
{
    private const GET = 'GET';
    private const PUT = 'PUT';
    private const KEY = 'key';
    private const DEFAULT_REQUEST_METHOD_VALUE = 1;

    /**
     * @var HttpClient
     */
    protected $httpClient;

    /**
     * @var string
     */
    public $url = '';

    /**
     * @var string
     */
    public $apiKey = '';

    /**
     * @var string
     */
    public $requestMethod = '';

    /**
     * @var ResponseHelper
     */
    private $responseHelper;

    public function __construct(HttpClient $httpClient, array $config = [])
    {
        parent::__construct($config);
        $this->httpClient = $httpClient;
        $this->responseHelper = make(ResponseHelper::class);
    }

    /**
     * @throws DirectoRequestException
     */
    public function get(array $body): ?SimpleXMLElement
    {
        $body = $this->prepareRequestBody(self::GET, $body);
        return $this->responseHelper->handleResponse(
$this->sendRequest($body));
    }
```

```
    /**
     * @throws DirectoRequestException
     */
    public function upsert(array $body): ?SimpleXMLElement
    {
        $body = $this->prepareRequestBody(self::PUT, $body);
        return $this->responseHelper->handleResponse($this-
>sendRequest($body));
    }

    private function sendRequest(array $body)
    {
        return $this->httpClient->send(
            $this->httpClient->createRequest($this->requestMethod, $this-
>url, ['body' => $body])
        );
    }

    private function prepareRequestBody(string $method, array &$body):
array
    {
        $body[$method] = self::DEFAULT_REQUEST_METHOD_VALUE;
        $body[self::KEY] = $this->apiKey;
        return $body;
    }
}
```

Lisa 5 – SyncDspToDirectoJob.php programmikood

```php
<?php

namespace app\modules\directo\cronjobs;

use app\components\interfaces\directo\DirectoSyncableInterface;
use app\cronjobs\AbstractCronJob;
use app\models\data\mysql\SyncDspToDirectoData;
use app\models\enum\SyncDspToDirectoAction;
use app\modules\directo\client\exceptions\DirectoRequestException;
use app\modules\directo\client\exceptions\DirectoUnauthorizedException;
use app\modules\directo\client\resources\DirectoResourceFactory;
use app\modules\directo\components\transformers\DirectoTransformerFactory;
use app\modules\directo\models\repository\SyncDspToDirectoRepository;
use RuntimeException;
use Yii;

/**
 * Class SyncDspToDirectoJob
 * @package app\modules\directo\cronjobs
 */
class SyncDspToDirectoJob extends AbstractCronJob
{
    /**
     * Default amount of entries to sync on one run.
     */
    const LIMIT = 100;

    /**
     * @var DirectoTransformerFactory
     */
    private $transformerFactory;

    public function __construct(DirectoTransformerFactory
$transformerFactory, array $config = [])
    {
        parent::__construct($config);

        $this->transformerFactory = $transformerFactory;
    }

    /**
     * @throws \Throwable
     */
    public function run()
    {
        $this->log('Start fetching the items in the queue');

        /** @var SyncDspToDirectoData[] $items */
        $items =
make(SyncDspToDirectoRepository::class)->getAdvertisersToSync();
```

```
        $this->log('Amount of items to be synced: ' . count($items));
        $successfulItems = [];
        $failedItems = [];

        foreach ($items as $syncDspToDirectoData) {
            $action = $syncDspToDirectoData->getAction();
            try {
                $model = $syncDspToDirectoData->getEntity();
                $this->handleAction($action, $model);
                $syncDspToDirectoData->delete();

                $successfulItems[] =
$syncDspToDirectoData->getEntityId();
                $this->log("Item successfully synced -
id={$syncDspToDirectoData->getId()}, " .
                    "model={$syncDspToDirectoData->getModelClass()},
primary={$syncDspToDirectoData->getEntityId()}"
                );
            } catch (DirectoUnauthorizedException $e) {
                throw $e;
            } catch (DirectoRequestException $e) {
                $failedItems[] = $syncDspToDirectoData->getEntityId();
                $this->log("Directo error when processing queue item -
id={$syncDspToDirectoData->getId()}, " .
                    "model={$syncDspToDirectoData->getModelClass()},
primary={$syncDspToDirectoData->getEntityId()}, " .
                    "exceptionMessage={$e->getMessage()},
exceptionCode={$e->getCode()}."
                );
                Yii::error($e);
            } catch (\Throwable $e) {
                 $syncDspToDirectoData->is_failed = 1;
                 $syncDspToDirectoData->save();

                $failedItems[] = $syncDspToDirectoData->getEntityId();
                $this->log("DSP error when processing queue item -
id={$syncDspToDirectoData->getId()}, " .
                    "model={$syncDspToDirectoData->getModelClass()},
primary={$syncDspToDirectoData->getEntityId()}, " .
                    "exceptionMessage={$e->getMessage()}, exceptionCode={$e-
>getCode()}."
                );
                Yii::error($e);
            }
        }
        $this->log('Finished processing ' . count($items) . ' items in
total!');
        $this->log(count($failedItems) . ' items failed!');
        $this->log(count($successfulItems) . ' items successfully synced to
Directo!');
    }

    /**
     * @throws DirectoRequestException
     */
    private function handleAction(string $action, DirectoSyncableInterface
$directoSyncable): void
    {
```

```
        $transformer = $this->
transformerFactory->get($directoSyncable);
        $directoRecord = $transformer->transform($directoSyncable);


$resource = DirectoResourceFactory::getResource($directoRecord);

        switch ($action) {
            case SyncDspToDirectoAction::UPSERT:
                $resource->upsert($directoRecord);
                return;
            default:
                throw new RuntimeException(
"Could not find action: {$action}");
}
```