TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kert Saarma 175850 IDDR

ELASTICSEARCHI RAKENDAMINE ELISA TTL NÄITEL

Diplomitöö

Juhendaja: Jaanus Pöial
PhD

Tallinn 2020

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: Kert Saarma

18.05.2020

Annotatsioon

Käesoleva diplomitöö eesmärk on luua esialgne lahendus Elisa Teenindaja Töölauale.

Eesmärgiks on luua rakendus, mis suudaks andmebaasis olevate andmete uuendamise peale konsolideerida uuenenud kliendiandmed ning saata need Elasticsearchi otsinugmootorisse. Elasticsearchis olevaid andmeid on võimalik rakenduse kaudu teistel Elisa rakendustel pärida.

Eesmärgi täitmiseks analüüsitakse erinevaid otsingumootoreid ning võrreldakse neid. Koostatakse andmemudel ning süsteemi arhitektuur. Samuti kirjeldatakse lahti rakenduse toimimine ning lisades on toodud välja osaliselt rakenduse kood.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 20 leheküljel, 4 peatükki, 5 joonist, 4 tabelit.

Abstract

Elasticsearch implementation on the example of Elisa

The aim of this thesis is to create a propriatary solution to the Elisa Teenindajate Töölaud.

The aim is to create an application that could consolidate client data when it is updated in the database and send the updated data to Elasticsearch search engine. Furthermore other Elisa applications can retreive data frome Elasticsearch through the application.

Diffrent search engines are analyzed and compared. A datamodel and system architecture are created. The inner workings of the application are explained and in the additions some code of the application is provided.

The thesis is in Estonian and contains 20 pages of text, 4 chapters, 5 figures, 4 tables.

Lühendite ja mõistete sõnastik

API	Rakendustarkvara liides
Desperados	Rakendus, mis on vahelüliks andmebaaside ja Elasticsearchi vahel
Dokument	Andmekogum, mis on võrreldav relatsioonilises andmebaasis tabeli reaga
Elasticsearch	Tekstipõhine otsingumootor
EPD	Andmebaas, kus on Elisa kliendiandmed
Indeks	Andmekogum, mis on võrreldav tavalise relatsioonilises andmebaasis tabeliga
JDBC	Java andmebaasi ühendus
Kild	Horisontaalne jagatis andmetest
Klaster	Kogum omavahel seotud objektidest
Node	Seade või andmepunkt võrgus
ODBC	Avatud andmebaasi ühendus
Solr	Tekstipõhine otsingumootor
Sphinx	Tekstipõhine otsingumootor
SV	Andmebaas, kus on Elisa teleteenuste andmed
TTL	Elisa klienditeenindajate kasutatav Teenindaja Töölaud

Sisukord

Sissejuhatus 9
1 Probleemi kirjeldus ja eesmärk 10
1.1 Ettevõtte tutvustus 10
1.2 Hetkeolukorra kirjeldus 10
1.3 Probleemipüstitus ja eesmärgid 10
1.4 Skoop 11
2 Analüüs 12
2.1 Ümberpööratud indeks 12
2.2 Elasticsearch 13
2.2.1 Dokumendid ja indeksid 14
2.2.2 Informatsiooni otsimine ning analüüsimine 14
2.2.3 Jaotus ja koopiate tegemine 15
2.3 Alternatiivide tutvustus 16
2.3.1 Solr 18
2.3.2 Sphinx 19
2.4 Elasticsarchi ja alternatiivide võrdlus 20
2.5 Lahenduse valik 23
3 Lahendus 24
3.1 Süsteemi funktisonaalsete nõuete kirjeldus 24
3.2 Süsteemi disain ja arhitektuur 24
3.3 Andmemudel 25
3.4 Rakendus 26
3.5 Võimalik edasiarendus 27
4 Kokkuvõte 29
Kasutatud kirjandus 30
Lisa 1 – EpdServiceFactory 33
Lisa 2 – ElasticApi 38

Jooniste loetelu

Joonis 1: Elasticsearchi trend [7] 17
Joonis 2: Sphinxi trend [8] 17
Joonis 3: Solri trend [9] 18
Joonis 4: Süsteemi arhitektuur 25
Joonis 5: Andmemudel 26

Tabelite loetelu

Tabel 1: Indekseerimise näidis 13
Tabel 2: Otsingumootorite võrdlus 21
Tabel 3: Muudatuspäise näidis 26
Tabel 4: Ülesande näidis 27

Sissejuhatus

Käesoleva diplomitöö eesmärk on luua uue Elisa Teenindaja Töölaua (edaspidi TTL) rakenduse üks võtmeosa ning analüüsida, kas valitud lähenemine on kõige parem lahendus. Hetkel puudub Elisa Eesti klienditeenindajatel võimalus otsida reaalajalisi kliendiandmeid tulenevalt Elisa ning Starmani ühinemisel tekkinud erinevatest süsteemidest. Antud probleemi lahendamiseks on valitud Elasticsearchi rakendamine ettevõtte infosüsteemiga. Elisa Eestil puudub varasem kokkupuude antud otsingumootoriga.

Töö esimeses peatükis kirjeldatakse probleemi olemust, Elisa TTL hetkeolukorda ning pannakse paika töö eesmärk ning skoop.

Töö teises peatükis tutvustatakse diplomitöö keskmesse valitud otsingumootorit, kirjeldatakse ning võrreldakse olemasolevaid alternatiivseid lahendusi Elasticserchiga ning põhjendatakse valiku tegemist.

Töö kolmandas peatükis kirjeldatakse programmi funktsionaalseid nõudeid, seletatakse lahti andmemudel, süsteemi arhitektuur, kirjeldatakse lahti lahendus ning tuuakse välja edasised arendusvõimalused.

Autor soovib tänada lõputöö juhendajat koostöö ning juhendamise eest.

1 Probleemi kirjeldus ja eesmärk

Käesolevas peatükis kirjeldatakse põhjalikumalt diplomitöös käsitletavat probleemi, antakse ülevaade hetkeolukorrast ning määratletakse ära diplomitöö skoop.

1.1 Ettevõtte tutvustus

Elisa on suurim erakliendi telekomi- ja TV-teenuste pakkuja ning suuruselt teine interneti püsiühenduse pakkuja Eesti turul. Elisa on enam kui 1000 töötajaga suurettevõte, mille käive oli 2018. aastal 169,7 miljonit eurot. Elisa kuulub Soome ühele suurimale telekommunikatsiooni ettevõttele Elisa OYJ. [1] Elisas kasutavad Teenindaja Töölauda klienditeenindajad, haldurid ning tootejuhid.

1.2 Hetkeolukorra kirjeldus

Hetkel kasutavad Elisa klienditeenindajad Teenindaja Töölaua rakendust, mis kajastab eelmise päeva andmete seisu. Kõiki andmeid, mida klienditeenindajal on vaja kliendi teenindamiseks, ei ole klienditeenindajale ühest konkreetsest kohast kättesaadav ning ta peab erinevatest rakendustest andmeid otsima, et saada terviklik ülevaade. See tuleneb Elisa ning Starmani ühinemisest, millega seoses on kasutusel veel eraldiseisvad süsteemid. Tulenevalt tehnoloogiate erisusest ei ole otsene liidestus olnud võimalik. Hetkel kõige suuremaks puuduseks on, et osad klienditeenindaja poolt otsitud andmed on päev vanad. Otsitud andmete uuendamine toimub kord päevas, seega ei ole klienditeenindajal kõige uuemat ülevaadet andmetest. Diplomitöö lahendus aitab tulevikus vähendada teenindusjuhtumile kuluvat aega ning jätab klienditeenindajal ära üleliigsete toimingute tegemise, saamaks terviklikku ülevaadet kliendist. Elisa Eestil ei ole siiani olnud kokkupuuteid Elasticsearchiga.

1.3 Probleemipüstitus ja eesmärgid

Diplomitöö eesmärk on luua rakenduse üks osa, mille kaudu oleks võimalik pärida koondatud reaalajalisi kliendiandmeid, Elasticsearchil põhinevas otsingus.

Kliendiandmed peavad olema reaalajas uuendatud ühest või enamast erinevast andmebaasist. Kliendiandmed peavad uuenema jooksvalt ka Elasticsearchi rakenduses. Rakendus peab uuendama ainult muudetud kliendiandmeid. Pikaajaline eesmärk on muuta Elisa klienditeenindaja töö hõlpsamaks ning kiiremaks ning läbi selle tõsta ka klienditeeninduse efektiivsust.

1.4 Skoop

Elisa on üks suurematest telekommunikatsiooni ettevõtetest Eestis ning hetkel puudub Elisa Eestil selline lahendus, mis võimaldaks eesmärgiks seatud probleemi lahendada. Lahenduse analüüs ja skoop on piiritletud Elisa juhtivarhitektide poolt ning arvesse on võetud ärilist huvi ning arenduse mahtu. Lahenduse käigus arvestatakse olemasoleva infosüsteemi arhitektuuriga. Diplomitöö skoop on teha rakendus, mis on piiritletud ühe andmebaasiga, kust loetakse reaalajaliselt muutuvaid andmeid.

2 Analüüs

Analüüsi eesmärk on tutvustada ning võrrelda erinevaid olemasolevaid otsingumootori võimalust pakkuvaid rakendusi. Analüüs selgitab ümberpööratud indeksi loogikat, Elasticsearchis informatsiooni otsimist ning analüüsimist, skaleeritavust ning andmete jaotuvust. Samuti tutvustatakse erinevaid alternatiive ning võrreldakse neid Elasticsearchiga.

2.1 Ümberpööratud indeks

Väga paljud tekstipõhised otsingumootorid kasutavad ümberpööratud indeksi loogikat. Täisteksti otsingusüsteem (ingl *Full-text search*) vajab indeksit, et teha päringuid. Kõige levinum indeks, mida täisteksti otsingus kasutatakse on ümberpööratud indeks. See loeb igat elementi (sõna, number jne) dokumendis ning märgib ära milline dokument sisaldab antud elementi. Ümberpööratud indeks leiab üles kõik unikaalsed elemendid, mis esinevad dokumendis ning tuvastab kõik dokumendid, kus antud elemendid esinevad.

Tabelis on toodud näide, kuidas toimib ümberpööratud indeksis kahe lause indekseerimine.

Tabel 1: Indekseerimise näidis

Dokument		Indeks	
Dokumendi nr	Sisu	Termin	Dokumendi nr
1	Test on vaja kodus täita	test	1
2	Ilm on ilus	on	1,2,3
3	Leib on kodust otsas	vaja	1
		täita	1
		ilm	2
		ilus	2
		leib	3
		kodus	1,3
		otsas	3

Antud näite põhjal on näha, et indekseeritud on kõik dokumentides esinevad sõnad ühekordselt ning on ära märgitud millistes dokumentides antud sõnad leiduvad.

2.2 Elasticsearch

Elasticsearch on analüüsi- ning otsingumootor, mille keskmeks on Elastic Stack. Elastic Stackis olevad rakendused nagu Logstash ja Beats lihtsustavad andmete kogumist,

agregeerimist, rikastamist ning salvestamist Elasticsearchi. Elastic Stacki kuulub samuti rakendus nimega Kibana, mis aitab visualiseerida, uurida, hallata ning monitoorida erinevaid andmeid. Elasticsearch võimaldab andmete indekseerimist, otsimist ning analüüsimist.

Elasticsearchis saab reaalajas otsida ning analüüsida erinevat tüüpi andmeid, nii struktureeritud kui struktureerimata teksti, numbrilisi andmeid või isegi ruumiandmeid. Elasticsearch suudab efektiivselt salvestada ning indekseerida andmeid, mis võimaldab teostada otsinguid vähese aja kuluga. Peale lihtsamate otsingute ja andmete koondamise, on võimalik leida andmetest trende ning mustreid. Elasticsearch on oma loomult lihtsasti skaleeruv ning toetab ka suuri andmemahte. [2]

2.2.1 Dokumendid ja indeksid

Elasticsearch sisaldab endas ka jagatud dokumendi hoidlat. Indeksit võib võtta, kui dokumentide kogumit (relatsioonilise andmebaasi mõistes kui tabel) ja igat dokumenti, kui väljade kogumit (relatsioonilise andmebaasi mõistes kui tabeli ühte rida), mis on omakorda võti-väärtus paarid mis sisaldavad andmeid. Selle asemel, et hoida informatsiooni ridade ja tulpadena, salvestab Elasticsearch andmed järjestatud JSON struktuurina. Dokumendid jagatakse ühte klastrisse kuuluvate sõlmede (ingl *node*) vahel ära ning andmed on igast sõlmest koheselt kättesaadavad. Kui dokument salvestatakse, siis see indekseeritakse ning on koheselt otsitav. Elasticsearch kasutab andmemudelit, mida kutsutakse ümberpööratud indeksiks, mis võimaldab väga kiireid otsinguid. Võimekus kasutada iga välja kohta eraldi andmestruktuuri, see kokku panna ja tagastada otsingul, teeb Elasticsearchist kiire otsingumootori. Elasticsearchil on võimalus olla kontekstivaba (ingl *schemaless*), mis tähendab, et dokumente salvestatakse ilma täpsustamata, kuidas peaks käsitlema igat eraldi välja dokumendis. [3]

2.2.2 Informatsiooni otsimine ning analüüsimine

Elasticsearchi dokumendi otsimise põhivõimekus seisneb selles, et Elasticsearch suudab kasutada kõiki Apache Lucene otsingumootori teegi võimalusi. Apache Lucene on üks vanemaid ning laialdasemalt kasutatavaid otsingumootori teeke. Elasticsearch võimaldab läbi REST API klastreid hallata, andmete indekseerimist ning ka andmete otsimist. Testimise eesmärgil on võimalik päringuid saata isegi läbi konsooli. Elasticsearchi rakendamiseks on tehtud kliendirakendus, mis toetab järgnevaid keeli: Java, JavaScript,

Go, .NET, PHP, Perl, Python ja Ruby. Elasticsearchi REST API toetab struktureeritud päringuid ning täistekstpäringuid. Struktureeritud päringud on sama laadi päringud, mida saab moodustada SQLis. Näiteks saab otsida ’sugu’ ja ’vanus’ välju oma töötaja tabelis ning sorteerida need töölevõtu kuupäev alusel. Täistekst päring leiab kõik dokumendid, mis vastavad antud kriteeriumitele ning tagastab need sorteeritud järjekorras. Tulemuse tagastamisel järjestatakse vasted selle järgi, mis võis olla kõige tõenäolisem vaste. Lisaks on võimalik otsida üksikuid sõnu või fraase. Kõiki neid otsinguvõimalusi saab kasutada Elasticsearchi JSON tüüpi päringukeele (Query DSL) kaudu. Samuti on võimalik kasutada ka SQL (Structured Query language) stiilis päringuid, et otsida ning koondada andmeid otse Elasticsearchis. Elasticsearch toetab JDBC ning ODBC draivereid, mis laiendab kolmanda osapoole rakenduste hulka, mida on võimalik ühendada Elasticsearchiga. Ühe päringu sees võib otsida dokumente, filtreerida tulemusi või teha analüütilisi päringuid ning seda kõike üheaegselt. Elasticsearch võimaldab ka masinõpet, mille läbi saaks monitoorida andmeid. [4]

2.2.3 Jaotus ja koopiate tegemine

Elasticsearch on ehitatud nii, et ta skaleeruks kasutaja vajaduste järgi. Selle saavutamiseks on Elasticsearch jagatud erinevate sõlmede vahel. Alati on võimalus sõlmi klastrisse juurde lisada ja läbi selle tõsta klastri andmemahtu. Elasticsearch automaatselt jagab andmed ning päringute mahu võrdselt kõikide olemasolevate sõlmede vahel ära. Elasticsearch indeks on loogiline grupeering ühest või enamast füüsilisest killust, kus iga kild on iseseisev indeks. Dokumendid jagatakse indeksisse ning indeksid jagatakse mitmesse erinevasse kildu ning killud omakorda mitmesse erinevasse sõlme, sellega saavutatakse liiasus, mis rikete korral tagab andmete säilivuse ning aitab optimiseerida suuremate päringute tegemist. Kui Elasticsearchi klaster kasvab või kahaneb jagatakse automaatselt killud ümber.

Kilde on kahte tüüpi: põhilised killud ja koopia killud. Iga dokument indeksis kuulub ühte põhilisse kildu, ning koopiakild on duplikaat põhikillust. Koopiad aitavad ära hoida andmete kadumise ning võimaldavad ka kiiremaid otsingutulemusi. Põhiliste kildude arv indeksis on fikseeritud, kui indeks luuakse, kuid koopia killdude arvu saab vastavalt vajadusele muuta.

Elasticsearchi kiiruse tagamiseks peavad kõik sõlmed ühes klastris eksisteerima samas võrgus, üle erinevate võrkude võib kildude jagamine võrdselt sõlmede vahel võtta liigselt aega. Elasticsearchis on võimalik teha klastritest koopiaid. Tavaliselt tehakse peamisest klastrist koopia andmete säilivuse tagamiseks. Vajadusel on võimalus peamine klaster kiirelt ümber vahetada koopia vastu ning edasised toimingud saavad sujuvalt jätkuda. Lisaks on võimalus teha klastritest koopiaid ka erinevate õigustega, näiteks on klastril ainult lugemisõigus. [5]

2.3 Alternatiivide tutvustus

Elasticsearchiga võrdluseks on valitud kaks alternatiivset otsingumootorit. Valik on tehtud pingerea alusel, valides viiest kõige populaarsemast alternatiivist kaks, mida saaks kasutada sarnaselt Elasticsearchile. Pingerida põhines järgnevatel kriteeriumitel:

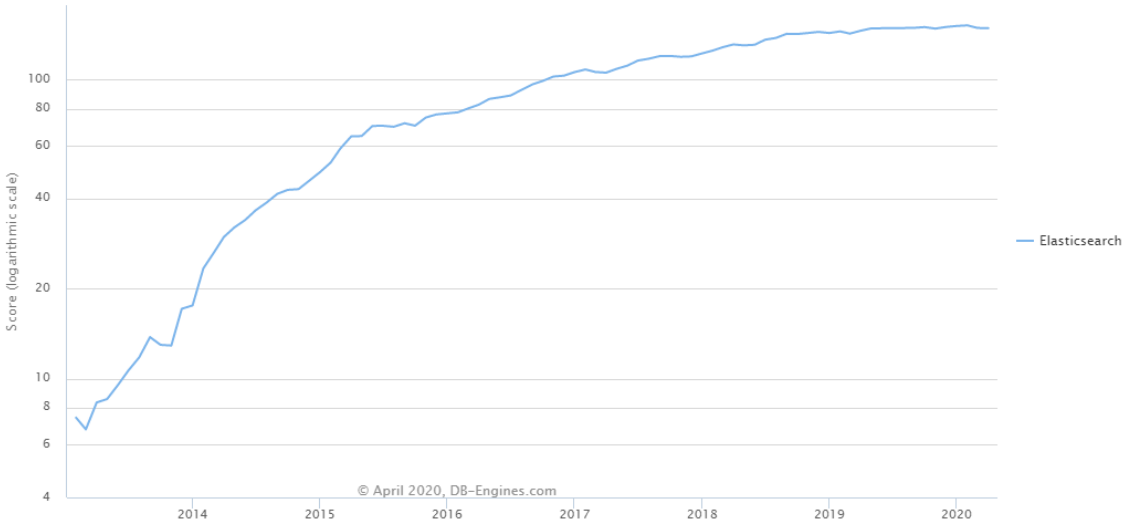
- Süsteemide mainitavus otsingumootorites Google, Bing ning Yandex
- Otsingu populaarsus Google Trendsis
- Tehniliste küsimuste sagedus süsteemi kohta Stack Overflow ja DBA Stack Exchange saitidel
- Töökuulutuste arv, mis mainivad süsteeme lehtedel Indeed ja Simply Hired
- Profesionaalsete profiilide arv, mis mainivad süsteeme keskkonnas LinkedIn ning Upwork
- Populaarsus sotsiaalmeedias Twitter, ehk säutsude arv [6]

Antud kriteeriumite põhjal tekkis pingerida, kus märgiti ära viis kõige populaarsemat:

1. Elasticsearch
2. Splunk
3. Solr
4. MarkLogic
5. Sphinx

Viie kõige populaarsema seast valiti välja esimesel kohal olev Elasticsearch, kolmandal kohal olev Solr ning viiendal kohal olev Sphinx.

Joonistelt on näha otsingumootorite trend ajas, mis on leitud eelmainitud kriteeriumite põhjal (Joonis 1, Joonis 2, Joonis 3). Nendelt saab välja lugeda, et Elasticsearch on hetkel kõige populaarsem, millele järgneb Solr ning siis Sphinx.



Joonis 1: Elasticsearchi trend [7]

Joonis 2: Sphinxi trend [8]

Joonis 3: Solri trend [9]

2.3.1 Solr

Apache Solr, nagu ka Elasticsearch, põhineb Apache Lucene otsingumootoril. Apache Solr on kirjutatud Java keeles. Solr kasutab otsimiseks indekseid, mis koosnevad erinevatest dokumentidest. Nagu ka Elasticsearch kasutab Solr dokumente, mis kujutavad erinevad välju. Solris on võimalik määratleda välja tüüp, mis aitab Solril paremini ning kiiremini otsinguid teostada. Kui lisatakse kirje, eraldatakse kirje väljad ning lisataks need dokumendi kujul indeksisse. Solril on võimekus olla kontekstivaba. Enamasti soovitatakse kasutada kontekstiga versiooni, sest kontekstivaba versiooni puhul peab Solr ise ära arvama andmetüübid ning sellega võivad kaasneda vead. [10]

Solr võimaldab teha filtreerimispäringut, mis on osa otsingupäringust. See annab võimaluse kitsendada otsingutulemust valitud parameetrite alusel. Filtreerimispäring võrdleb indeksit ning vahemällu talletatud tulemusi. Kuna Solr eraldab eraldi vahemälu filtreerimispäringute jaoks, aitab strateegiline filterdamine kiirendada otsingute tulemust.

Solril on kaks lisameetodit, mis aitavad leida otsitud vastet kiiremini, kategoriseerimine (ingl *faceting*) ja klasterdamine (ingl *clustering*). Kategoriseerimine on otsingutulemuste organiseerimine kategooriasse, mis põhinevad indeksi terminitel. Näiteks e-poes on informatsioon kategoriseeritud tootjate alusel ning tootja nime all on tooted, mis on antud tootja poolt toodetud. Klasterdamine toimub otsingu käigus, kus tagastatakse väga

sarnane info, aga mida algselt ei otsitud. [11] Näiteks otsin sõna ’tamm’, aga mulle tagastatakse ka sõna ’tammepuu’, kuna see on sarnane minu otsitule.

Solri päringud käivad üle HTTP päringute ning enamasti on tagastatav objekt JSON kujul, kuid võib olla ka XML, CSV või mõni muu formaat. See tähendab, et väga suur hulk kliendirakendusi saab kasutada Solri, alustades veebirakendustest ning lõpetades mobiilirakendustega. Iga platvorm, mis võimaldab HTTP-d, suudab suhelda Solriga. Üle HTTP saab teha viit toimingut: pärida, indekseerida, kustuda, salvestada ja optimiseerida. [12]

Skaleeritavus

Apache Solris on võimalik serveritesse teha klastritest koopiaid. See on kasulik rikete puhul, kuna siis ei lähe andmed kaduma, ja rohkete päringute puhul. Solril puudub keskne server, selle asemel kasutatakse rakendust nimega ZooKeeper, mis haldab kõiki erinevaid servereid.

Sarnaselt Elasticseachile, toimub ka Solris mitme killu jagamine erinevate serverite vahel, mis lihtsustab otsimist ning tagab ka andmete koopiate olemasolu. Kui põhilise killuga peaks midagi juhtuma, valitakse suvaliselt mõni koopiakild uueks põhiliseks killuks. [13]

2.3.2 Sphinx

Sphinx on otsingumootor, mis on tasuta kasutamiseks kõigile. See on kirjutatud C++ keeles ja keskendub otsingu tulemustele ja otsingu ajakohasusele. Peamine rakendus, mille kaudu on võimalik Sphinxiga suhelda, kasutab SphinxQl-te mis on SQLi dialekt. Kõik rakendused, millel on ühilduvus MySQL-iga, ühilduvad ka Sphinxiga. Ühtlasi on Sphinxil võimekus ühenduda ka HTTP ja JSON API kaudu. Sphinx kasutab samuti dokumentide ning indeksite andmestruktuure. Võrreldes teiste võrdluses olevate otsingumootoritega, on Sphinxi dokumentatsioon väga pinnapealne.

Sphinx võimaldab:

- Peaaegu reaalajas võrguvaba suuremahulist indekseerimist.
- Täis ja osalise tekstiotsinguid.
- Otsingu tulemuse järjestust asjakohasuse järgi

Sphinxi indeksid on poolstruktuursed kollektsioonid dokumentidest. Nad on rohkem relatsioonilise andmebaasi tabelite sarnased. Põhiline andmetüüp on täistekstindeks. See on spetsiaalne struktuur, mis aitab otsingutulemusi kiiremini tuvastada ning tagastada. Kontekstipoolt üritab Sphinx kombineerida konteksti ja kontekstivabasid lahendusi. [14]

2.4 Elasticsearchi ja alternatiivide võrdlus

Võrdluses olevad otsingumootorid on väga sarnased, kuid teatud aspektid on siiski erinevad. Järgnev tabel (Tabel 2) võrdleb eelnevalt tutvustatud otsingumootoreid ning nende funktsionaalsust. ’Jah’ tähendab, et antud omadus on rakendusel olemas, ’ei’ tähendab, et rakendusel puudub antud omadus.

Tabel 2: Otsingumootorite võrdlus

Nimi	Elasticseach	Solr	Sphinx
Litsents	ApacheVersion 2, Elastic Licence	ApacheVersion 2	GPL version 2, Eraldi äri litsents
Rakendamiskeel	Java	Java	C++
Määratletud andmetüübid, nagu *float* ja *date*	Jah	Jah	Ei
API ja muud suhtlusmeetodid	Java API RESTful HTTP/JSON API	Java API RESTful HTTP/JSON API	Omaenda protokoll
XML toetus	Ei	Jah	Jah
Serveripoolsed salvestatud protseduurid	Jah	Jah, läbi Java pistikprogrammide	Ei
Andmete säilivuse jaoks koopiate tegemine	Jah	Jah	Ei
Jaotatud otsing	Jah	Jah, kuid vajab lisaseadistust	Jah
Andmebaasi kildude ümberjaotamine	Toimub automaatselt	Vajab välist rakendust nagu näiteks Apache Zookeeper	Ei võimalda
Serverites uuenduste läbiviimine	Saab teha jooksvalt reaalajalises serveris	Peab iga sõlme taaskäivitama	Info puudub
Vahemälu uuendamine	Segmendi kaupa	Kui segment uueneb, tuleb kogu globaalne vahemälu uuendada	Info puudub

Elasticsearch on hetkel turuliider ning tema võimekus ja positsioon otsingumootorite seas on kõige kõrgem. Elasticsearch suudab kõige kiiremini uuendada andmeid, tänu oma osalisele kildude uuendamisele. Lisaks skaleerub Elasticsearch koos andmemahuga. Negatiivse küljena võib välja tuua, et Elasticsearch nõuab rohkem mälu, kuid Elasticsearchi tiim tegeleb optimiseerimisega.

Solri oleks mõistlik kasutada, kui on vaja lugeda suuremaid dokumente või PDF faile. Solril on väga hea kategoriseeritud otsing ning Solr on parim valik, kui tegeletakse staatiliste andmetega, mis väga tihti ei muutu.

Sphinx on väga pikaaegselt turul olnud otsingumootor. Sphinxi positiivsemaks küljeks saab lugeda, et tema otsingud nõuavad väga vähe mälu.

Tasuks ära märkida viimased uuendused, mis otsingumootoritel on välja antud. Elasticsearchi viimane versioon 7.6.2 avalikustati 2020 märtsis, Solri viimane verisoon 8.5.1 avalikustati 2020 aprillis, Sphinxi hetkel viimane versioon 3.1.1 on avalikustatud 2018 oktoobris. Sphinxil ei ole viimase kahe aasta jooksul toimunud uuendusi, seega see ei oleks hetkel väga ajakohane valik.

Lugedes erinevate otsingumootorite dokumentatsiooni leiti, et kõige parem dokumentatsioon on Elasticsearchil, millel on üleüldine info eraldatud väga detailsetest infost ning koodinäidetest. Kõige halvemini oli dokumentatsioon tehtud Sphinxil, kus Sphinxi loojad ise märkisid ära, et nende dokumentatsioon pole väga hea ning nad üritavad tulevikus paremat dokumentatsiooni kirjutada ning hetkel võib esineda nende dokumentatsioonis puudujääke.

Kõikidel otsingumootoritel on nii positiivsed kui ka negatiivsed omadused. Elasticsearchil on alustamine tehtud väga lihtsaks, kuid suuremate projektide jaoks võib see osutuda probleemiks, kuna kasutaja eest on ära peidetud funktsionaalsust, mis toimub automaatselt. Solril on alustamine tehtud aga võrdlemisi keeruliseks ning kohe alguses tuleb väga palju vaeva näha seadistamisega, kuid erinevalt Elasticsearchist, ei pea seda tegema tulevikus. Üheks suureks vaheks Solri ja Elasticsearchi vahel on vahemäluga käitumine. Solri globaalne vahemälu on ülesehitatud nii, et kui tehakse muudatus

andmetes, peab kõiki andmeid uuendama. Elasticsearchis, aga saab uuendada andmeid segmentidena.

Erinevuseks Elasticsearchi ja Solri vahel on ka nende funktsionaalsuse juurde arendamine. Solr on vabavaraline, seega kõik saavad kirjutada juurde uust funktsionaalsust, kuid sellega võib kaasneda juba olemasoleva funktsionaalsuse mitte toimimine, kuna puudub keskne kontroll. Elasticsearch on osaliselt vabavaraline, kood on kõigile nähtav ja samuti võivad kasutajad lisada uut funtksionaalsust, kuid selleks, et uuendused ka Elasticsearchi jõuaks, peab keegi Elasticsearchi tiimist kinnitama selle idee ning rakendama selle rakenduses.

Suures osas on Solr ja Elasticsearch väga sarnased. Mõlemad põhinevad Apache Lucene otsingumootoril ning mõlemil on väga sarnased omadused funktsionaalsuse aspektist.

2.5 Lahenduse valik

Elasticsearch sai valitud, sest Elisale sobib Elasticsearchi kiirus ning skaleeritavus. Teine alternatiivne variant oleks olnud Solr, kuid kuna Solril on globaalne vahemälu, mida uuendatakse iga kord, kui andmed muutuvad, tundub see kulukas ning ebavajalik funktsionaalsus. Elisa rakendus peab uuendama andmeid jooksvalt, mitu korda päevas. Ühtlasi on Elasticsearch hetkel kõige populaarsem otsingumootor, mida toetab ka fakt, et see oli ainuke mille nimi tuli kohe meelde, kui mainiti otsingumootori ehitamist. Abiks tuli ka see, et Elisa Soomes on Elasticsearhci juba kasutatud ehk arhitektidel oli võimalus Soome kolleegidega konsulteerida lahenduste osas. Lisaks on Elasticsearchi rakendamine võrdlemisi lihtsaks tehtud.

3 Lahendus

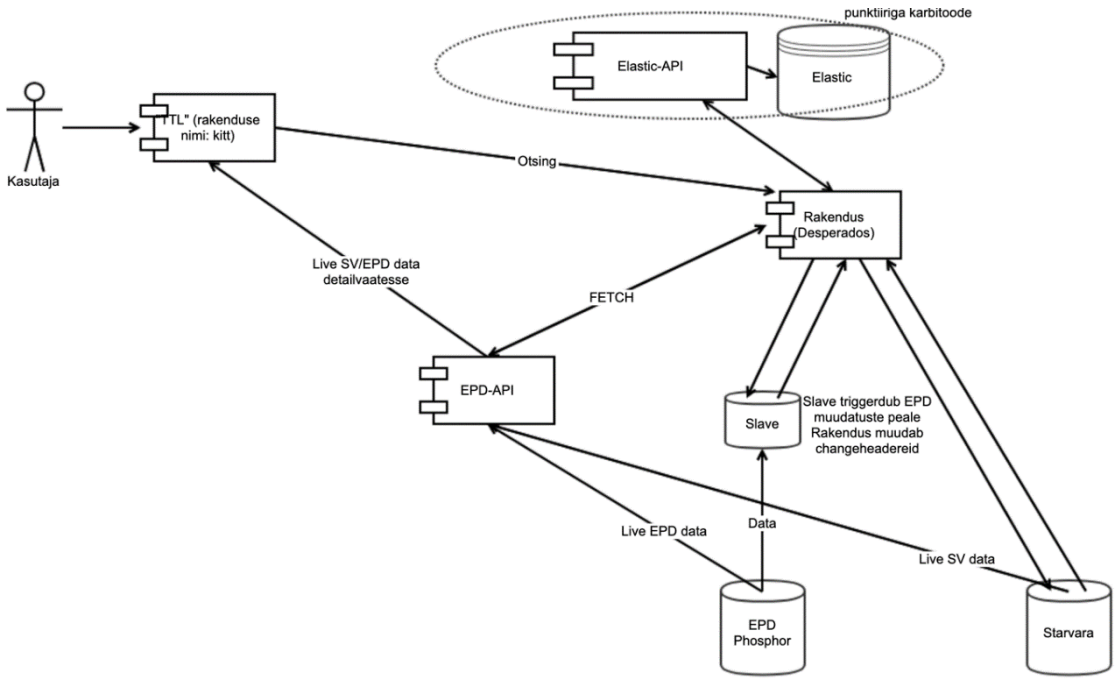
Käesolevas peatükis kirjeldatakse ära süsteemi funktsionaalsed nõuded. Kirjeldatakse lähemalt süsteemi disaini, arhitektuuri ning selgitatakse põhjalikumalt rakenduse toimimist ning edasiarenduse võimalusi.

3.1 Süsteemi funktisonaalsete nõuete kirjeldus

Funktsionaalsed nõuded kirjeldavad, kuidas peaks kavandatav süsteem toimima ning milliseid funktsioone täitma. Tehtav rakendus peab andmebaasist lugema tehtud muudatusi ning vastavalt muudatustele leidma õige kliendi, kelle andmeid muudeti. Rakendus peab leidma kliendi, kelle andmeid muudeti, lahtised lepingud ning ühendama need kokku üheks JSON objektiks. Lisaks peab rakendus saatma loodud JSON objekti Elasticsearchi, kust hiljem on võimalik seda pärida. Kui klient on juba olemas Elasticsearchi andmebaasis, tuleb selle kliendi muutunud andmeid uuendada. Rakendusel peab olema võimekus lugeda muutunud andmeid ühest või enamast andmebaasist. Rakenduse jaoks tuleb luua andmebaasidele päästikud (ingl *triggers*). Ühtlasi peab rakendus võimaldama suhtlust Elasticsearchiga.

3.2 Süsteemi disain ja arhitektuur

Joonisel on näha plaani, kuidas rakendus tulevikus töötama hakkab (Joonis 4). Antud diplomitöös keskenduti Desperadose ja Elasticu rakendamisele. Joonis (Joonis 4) kirjeldab kogu süsteemi ülesehistust ning kirjeldab ka kuidas elemendid omavahel suhtlevad.

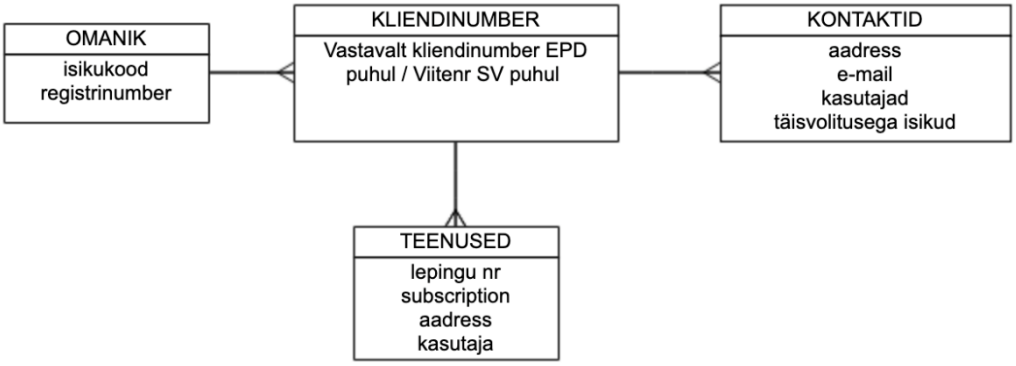


Joonis 4: Süsteemi arhitektuur

EPD phosphor on reaalajaline andmebaas, kus on kõige viimane seis andmetest. Slave-i laetakse andmeid reaalajalisest andmebaasist ning Slave-le on peale ehitatud päästikud, mis tekitavad muudatustest kirjed. Desperadose rakendus loeb antud kirjete põhjal muudatusi ning vastavalt sellele konsolideerib kliendi andmed ning saadab Elasticsearchi muutmiseks või lisamiseks. Läbi Desperadose toimub ka kliendiandmete pärimine.

3.3 Andmemudel

Andmemudel kirjeldab loodava süsteemi andmeobjektide vahelisi seoseid. Loodud andmemudel on kooskõlastatud ettevõtte arhitektidega. Antud andmemudelil olevad elemendid kujutavad Elasticsearchi saadetavaid andmeid. (Joonis 5)



Joonis 5: Andmemudel

Igale kliendile kuulub unikaalne kliendinumber, mille alusel leitakse üles kõik kasutajad (mudelil element kontaktid), mis kuuluvad antud kliendile (mudelil element kliendinumber) ning ka kõik teenused, mida antud klient hetkel kasutab, sinna alla kuuluvad näiteks internetipakett, mobiilipakett jne. Igal kliendil on üks omanik. Omanik määratletakse kas isikukoodi või registrikoodi alusel. Info kogutakse tabelitest kokku, tehakse üheks JSON objektiks ning suunatakse edasi Elasticsearchi.

3.4 Rakendus

Rakendus on üles ehitatud Elisa-sisesele Coppo 3 raamistikule ning kirjutatud Java keeles. Raamistik töötab ülesannete (ingl *task*) alusel. Reaalajalises andmebaasis luuakse kindlate tabeli muudatuste peale muudatuspäis (ingl *changeheader*), mis näitab millal, kus tabelis ning mis kliendinumbriga seoses muudatus tehti.

Näide muudatuspäise struktuuri kohta:

Tabel 3: Muudatuspäise näidis

Database	Andmebaasi tabeli nimi, kus muudatus toimus
Tablename	Tabel, kus muudatus toimus
Changedrow	Rida, kus muudatus toimus
Operation	Muudatuse liik: Uuendamine, Lisamine
Changed	Millal muudatus läbi viidi
Status	Kas muudatuspäise kohta on ülesanne loodud või mitte
Applicationdata	Kliendinumber, kelle kohta muudatus käis

Iga muudatuspäise kohta genereerib rakendus ülesande, mida täita.

Näide ülesande struktuuri kohta:

Tabel 4: Ülesande näidis

Source_class	Tabeli nimi, kus muudatus toimus
Source_id	Kliendinumber, kelle kohta muudatus käis
Changeheader_id	Millise muudatuspäisega on ülesanne seotud
Task	Mis on ülesande nimi
Status	Kas ülesanne on loodud, sooritatud või ebaõnnestunud

Lisaks on ära määratletud, mitu korda on proovitud jooksutada antud ülesannet, juhul kui on ebaõnnestunud, siis mis põhjusel on ebaõnnestunud, täpsem selgitus ebaõnnestumisest, millal ülesanne loodi, viimane täitmiskuupäev ning õnnestumise kuupäev.

Iga ülesanne algab päringuga Elasticsearchi, tehes kindlaks, kas antud kliendiandmed eksisteerivad juba otsingumootoris või mitte (vt Lisa 2 meetod find()). Kui antud kliendiandmed juba eksisteerivad Elasticsearchis, siis suundutakse antud kliendiandmeid muutma, kui ei eksisteeri, genereeritakse uus kirje. Iga ülesannet lahendades võtab rakendus muudatuspäise küljest kliendinumbri ning otsib kliendinumbri alusel andmebaasist omaniku (vaata lisa 1 meetodit createClient()). Kui omanik on leitud määratletakse ära, kas tegemist on äri- või eraisikuga. Olles selle kindlaks teinud sisestatakse andmed Omaniku objekti, mille külge lisatakse ka kliendinumber, kontaktid ning teenused, mis kuuluvad antud omaniku alla (vaata lisa 1 meetodid setContactsList() ning generateServices()). Kontaktides täpsustatakse kliendi aadress, e-mail, kasutaja ning märgitakse ära, kas kasutaja on täievolituslik isik või mitte (vt Lisa 1 meetod generateContacts()). Teenuste all otsitakse välja kliendinumbri alusel kõik lahtiolevad lepingud ning ükshaaval märgitakse ära lepingu number, tüüp, aadress ning teenuse kasutaja. (vt Lisa 1 meetod generateServices()) Lõpuks pannakse kogutud info kokku ning tehakse ümber JSON tüüpi objektiks ning saadetakse Elasticsearchi, kus seda on võimalik hiljem otsida (vt Lisa 2 meetod postToIndex()).

3.5 Võimalik edasiarendus

Kuna diplomitöö raames sai Desperadose rakendus üles seatud ning sellega seotud üks andmebaas, kust andmeid laetakse, siis tulevikus on plaanis Desperadose külge lisada ka teisi andmebaase. Järgmisena on plaanis liidestada juurde endise Starmani andmebaas.

Hetkel on antud lahendus vaadatud üle vanemarendajate poolt, kes on andnud soovitusi, kuidas muuta antud süsteemi paremaks. Enamasti osutus selleks optimiseerimine kliendi andmete pärimisel. Kindlasti saab optimiseerida kliendi konsolideerimise protsessi, mis hetkel on töötav, kuid võiks olla veelgi vähem ressursinõudlikum. Lisaks on võimalus kirjutada juurde üha keerukamaid ja täpsemaid päringuid.

4 Kokkuvõte

Käesoleva diplomitöö tulemusena sai tehtud rakendus, mis suudab jooksvalt konsolideerida kliendiandmeid ning saata uuenenud andmed Elasticsearchi, kus nad on otsitavad. Põhjus projekti tegemiseks oli uue Elisa Teenindaja Töölaua arendamine, parandamaks klienditeenindajate tööriistu ning muutmaks nende tööd kiiremaks ja efektiivsemaks.

Analüüsi osas selgitati lähemalt Elasticsearchi, mis on antud diplomitöö keskmeks valitud otsingumootor. Tutvustati alternatiivseid otsingumootoreid ning võrreldi neid Elasticsearchiga. Põhjendati ka Elasticsearchi valimist.

Rakendus võimaldab hoida Elasticsearchis reaalajalisi konsolideeritud kliendiandmeid, tehes need kohe päritavaks, elimineerides sellega praeguse probleemi, kus osad kliendiandmed on päev vanad ning asuvad erinevates süsteemides. Loodud rakendus teeb Elisa TTL kasutajate töö lihtsamaks, kuna nad ei pea otsima andmeid erinevatest süsteemidest, vaid saavad pärida reaalajalisi andmeid ühest kesksest rakendusest.

Töö eesmärk oli luua rakendus, mis võimaldab:

- Lugeda andmeid ühest või enamast andmebaasist
- Uuenenud andmete põhjal konsolideerida kliendiandmed
- Uuenenud kliendiandmed saata Elasticsearchi
- Läbi rakenduse suhelda Elasticsearchiga, tehes võimalikuks erinevad päringud

Loodud süsteem on kooskõlas kõigi eelpool mainitud eesmärkidega. Püstitatud probleem on lahendatud ja töö eesmärk on saavutatud.

Järgnevalt on plaanis optimiseerida kliendiandmete konsolideerimist ning lisada teine andmebaas, kust saaks pärida kliendiandmeid. Ühtlasi on plaan luua juurde meetodeid, millega otsinguid Elasticseachist täiendada ja veelgi rohkem täpsustada.

Kasutatud kirjandus

[1] E. Eesti, „Elisa organisatsioonist,“ Elisa, [Võrgumaterjal]. Available: https://www.elisa.ee/et/elisast/organisatsioonist. [Kasutatud Aprill 2020].

[2] Elastic, „Elasticsearch introduction,“ Elastic, [Võrgumaterjal]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/elasticsearch-intro.html. [Kasutatud Aprill 2020].

[3] Elastic, „Data in: documents and indices,“ Elastic, [Võrgumaterjal]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/documents-indices.html. [Kasutatud Aprill 2020].

[4] Elastic, „Information out: search and analyze,“ Elastic, [Võrgumaterjal]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-analyze.html. [Kasutatud Aprill 2020].

[5] Elastic, „Scalability and resilience: clusters, nodes and shards,“ Elastic. [Võrgumaterjal]. [Kasutatud Aprill 2020].

[6] DB-Engines, „Method of calculating the scores of the DB-Engines Ranking,“ DB-Engines, [Võrgumaterjal]. Available: https://db-engines.com/en/ranking_definition. [Kasutatud Aprill 2020].

[7] DB-Engines, „DB-Engines Ranking - Trend of Elasticsearch Popularity,“ Aprill 2020. [Võrgumaterjal]. Available: https://db-engines.com/en/ranking_trend/system/Elasticsearch. [Kasutatud Aprill 2020].

[8] DB-Engines, „DB-Engines Ranking - Trend of Sphinx Popularity,“ Aprill 2020. [Võrgumaterjal]. Available: https://db-engines.com/en/ranking_trend/system/Sphinx. [Kasutatud Aprill 2020].

[9] DB-Engines, „DB-Engines Ranking - Trend of Solr Popularity,“ Aprill 2020. [Võrgumaterjal]. Available: https://db-engines.com/en/ranking_trend/system/Solr. [Kasutatud Aprill 2020].

[10] Solr, „Overview of Documents, Fields, and Schema Design,“ Apache Software Foundation, 25 03 2020. [Võrgumaterjal]. Available: https://lucene.apache.org/solr/guide/8_5/overview-of-documents-fields-and-schema-design.html#overview-of-documents-fields-and-schema-design. [Kasutatud Aprill 2020].

[11] A. S. Foundation, „Overview of Searching in Solr,“ Apache Software Foundation, 25 03 2020. [Võrgumaterjal]. Available: https://lucene.apache.org/solr/guide/8_5/overview-of-searching-in-solr.html#overview-of-searching-in-solr. [Kasutatud Aprill 2020].

[12] A. S. Foundation, „Introduction to Client APIs,“ Apache Software Foundation, 25 03 2020. [Võrgumaterjal]. Available: https://lucene.apache.org/solr/guide/8_5/introduction-to-client-apis.html#introduction-to-client-apis. [Kasutatud Aprill 2020].

[13] A. S. Foundation, „SolrCloud,“ Apache Software Foundation, 25 03 2020. [Võrgumaterjal]. Available: https://lucene.apache.org/solr/guide/8_5/solrcloud.html#solrcloud. [Kasutatud Aprill 2020].

[14] A. Aksyonoff, „Sphinx 3,“ Solr, [Võrgumaterjal]. Available: http://sphinxsearch.com/docs/sphinx3.html. [Kasutatud Aprill 2020].

[15] L. S. Sterling, The Art of Agent-Oriented Modeling, London: The MIT Press, 2009.

[16] C. G. a. Z. Tong, „Elasticsearch: The Definitive Guide,“ O'Reilly Media, 2015.

[17] S. T. Inc., „Commerical licensing,“ Sphinx Technologies Inc., [Võrgumaterjal]. Available: http://www.sphinxsearch.com/services/embedding/. [Kasutatud Aprill 2020].

[18] N. Jariwala, „Aureate,“ Aureate Labs., 03 March 2020. [Võrgumaterjal]. Available: https://aureatelabs.com/magento-2/which-search-engine-i-can-choose-among-the-sphinx-solr-elasticsearch/. [Kasutatud Aprill 2020].

[19] J. Nemer, „Elasticsearch vs. CloudSearch: AWS Cloud Search Choices,“ Cloud Academy Inc., 21 February 2020. [Võrgumaterjal]. Available:

https://cloudacademy.com/blog/elasticsearch-vs-cloudsearch/. [Kasutatud Aprill 2020].

[20] R. Kuć, „Solr vs. Elasticsearch: Performance Differences & More. How to Decide Which One is Best for You,“ Sematext Group, 18 July 2019. [Võrgumaterjal]. Available: https://sematext.com/blog/solr-vs-elasticsearch-differences/. [Kasutatud 2020 Aprill].

[21] Lucidworks, „Full Text Search Engines vs. DBMS,“ 29 July 2019. [Võrgumaterjal]. Available: https://lucidworks.com/post/full-text-search-engines-vs-dbms/. [Kasutatud 2020 Aprill].

[22] DB-Engines, „System Properties Comparison Elasticsearch vs. Solr vs. Sphinx,“ DB-Engines, [Võrgumaterjal]. Available: http://develop.sunshiny.co.kr/attach/Elasticsearch_vs_Solr_vs_Sphinx_Comparison.html. [Kasutatud Aprill 2020].

[23] R. K. M. R. S. C. Bharvi Dixit, Elasticsearch: A Complete Guide, Birmingham: Packt Publishint Ltd., 2017.

Lisa 1 – EpdServiceFactory

```
public class EpdServiceDataFactory {

	private static final Logger LOGGER =
LoggerFactory.getLogger(EpdServiceDataFactory.class);



	@Transactional(database = Db.EPD)
	public Owner createClient(Integer customerId) throws SQLException,
IOException {
		Owner owner = new Owner();

		ContactInfo contactInfo =
getWantedEpdServiceByAsiakasNumero(customerId);



		if (contactInfo.getCompanyCode() != null) {
			owner.setRegNr(contactInfo.getCompanyCode());
		} else {
			owner.setSsn(contactInfo.getSsn());
		}
		Integer aindex = contactInfo.getAindex();
		owner.setCustomerList(generateCustomerList(customerId, aindex));
		printJson(owner);
		insertIntoElastic(printJson(owner));
		return owner;
	}



	public List<Customer> generateCustomerList(Integer customerId, Integer
aindex) throws SQLException {
		List<Customer> customerList = new ArrayList<>();
```

```
            Customer customer = new Customer();
            customer.setCustomerId(customerId.toString());
            customer.setSource("EPD");
            customer.setContactsList(generateContacts(customerId));
            customer.setAllServicesList(generateServices(customerId,
aindex));
            customerList.add(customer);
            return customerList;
      }

      @Transactional(database = Db.EPD)
      public List<Contacts> generateContacts(Integer customerId) throws
SQLException {
            ContactInfo contactInfo = new ContactInfo();
            ContactInfo[] allUsers =
contactInfo.findOpenByUserId(DaoOperations.getConnection(), customerId);
            List<Contacts> contactsList = new ArrayList<>();
            for (ContactInfo contct : allUsers) {
                  if(contct.getType().contains("owner")) {
                        Contacts contacts = new Contacts();
                        contacts.setUserName(contct.getFirstName() + " " +
contct.getSurname());
                        contacts.setEmail(contct.getEmail());
                        contacts.setAddress(contct.getAddress());

      contacts.setCompleteUser(isCompleteUser(contactInfo.getAindex()));
                        contactsList.add(contacts);
                  }
            }
            return contactsList;
      }

      private Boolean isCompleteUser(Integer contactinfoId) throws
SQLException {
            Mandate[] mandates =
Mandate.findActiveAndFutureByContactInfoId(DaoOperations.getConnection(),
contactinfoId);
```

```
		String s = "COMPLETE";
		for (Mandate mand: mandates) {
			if (s.equals(mand.getType()))
				return true;
		}
		return false;
		}

	@Transactional(database = Db.EPD)
	public List<AllServices> generateServices(Integer customerId, Integer
aindex) throws SQLException {

		List<AllServices> AllServicesList = new ArrayList<>();
		Contract contrac = new Contract();
		Contract[] contracts =
contrac.findUnclosedByUserId(DaoOperations.getConnection() ,customerId);
		ServiceSupport serviceSupport = new
ServiceSupport(DaoOperations.getConnection());

		for (Contract contract : contracts) {
			AllServices AllServices = new AllServices();

			HashMap<String,List<Integer>> serviceHashMap =
serviceSupport.findAllUnclosedServicesInContract(
				contract.getAindex(),false);
			if (serviceHashMap.get("gsm") != null) {
				List<Integer> list = serviceHashMap.get("gsm");
				for (Integer service : list) {
					Gsm fetchedGsm =
Gsm.fetchByPrimaryKey(DaoOperations.getConnection(), service);

	AllServices.setGsmNumber(fetchedGsm.getGsmNumber());
					ContactInfo info =
ContactInfo.fetchByPrimaryKey(DaoOperations.getConnection(),
fetchedGsm.getUser());
					AllServices.setAddress(info.getAddress());
```

```
	AllServices.setContractNumber(fetchedGsm.getContractId());
				AllServices.setSubscription("gsm");
				AllServices.setUser(info.getFirstName() + "
" + info.getSurname());
				AllServicesList.add(AllServices);
			}

		}
		if (serviceHashMap.get("subscriber") != null) {
			List<Integer> list =
serviceHashMap.get("subscriber");
			for (Integer service :list) {
				Subscriber fetch =
Subscriber.fetchByPrimaryKey(DaoOperations.getConnection(), service);
				ContactInfo info =
ContactInfo.fetchByPrimaryKey(DaoOperations.getConnection(),
fetch.getUserId());
				AllServices.setAddress(info.getAddress());

	AllServices.setContractNumber(fetch.getContractId());
				AllServices.setSubscription("subscriber");
				AllServices.setUser(info.getFirstName() + "
" + info.getSurname());
				AllServicesList.add(AllServices);
			}

		}
		if (serviceHashMap.get("voipsubscription") != null) {
			List<Integer> list =
serviceHashMap.get("voipsubscription");
			for (Integer service : list) {
				VoipSubscription fetch =
VoipSubscription.fetchByPrimaryKey(DaoOperations.getConnection(), service);
				ContactInfo info =
ContactInfo.fetchByPrimaryKey(DaoOperations.getConnection(),
fetch.getContactInfo());
```

```
                                AllServices.setAddress(info.getAddress());

      AllServices.setContractNumber(fetch.getContractId());

      AllServices.setSubscription("voipsubscription");
                                AllServices.setUser(info.getFirstName() + "
" + info.getSurname());
                                AllServicesList.add(AllServices);
                          }

                    }
                    if (serviceHashMap.get("gsmorders") != null) {
                          List<Integer> list =
serviceHashMap.get("gsmorders");
                          for (Integer service : list) {
                                GsmOrder fetch =
GsmOrder.fetchByPrimaryKey(DaoOperations.getConnection(), service);
                                ContactInfo info =
ContactInfo.fetchByPrimaryKey(DaoOperations.getConnection(),
fetch.getCustomerId());
                                AllServices.setAddress(info.getAddress());

      AllServices.setContractNumber(fetch.getContractId());
                                AllServices.setSubscription("gsmorders");
                                AllServices.setUser(info.getFirstName() + "
" + info.getSurname());
                                AllServicesList.add(AllServices);
                                }
                          }
                    }
             return AllServicesList;
             }

      public ContactInfo getWantedEpdServiceByAsiakasNumero(Integer
asiakasNumero) {
             try {
```

```
                    return
ContactInfo.fetchBillingCustomerByType(DaoOperations.getConnection(),
asiakasNumero);
             } catch (SQLException e) {
                    throw new TechnicalException("Could not fetch opened epd
service with subscriberId: " + asiakasNumero, e);
             }
       }

       private String printJson (Owner owner) throws JsonProcessingException
{

             ObjectMapper Obj = new ObjectMapper();
             return  Obj.writeValueAsString(owner);
       }

       private void insertIntoElastic(String string) throws IOException {
             ElasticAPI elasticAPI = new ElasticAPI();
             elasticAPI.postToIndex("testing",string);
       }
}
```

Lisa 2 – ElasticApi

```
public class ElasticAPI {
       private static final Logger LOGGER =
LoggerFactory.getLogger(ElasticAPI.class);

       RestHighLevelClient client = new RestHighLevelClient(
             RestClient.builder(
                    new HttpHost("localhost", 9200, "http")));

       boolean response;
```

```
	{
		try {
			response = client.ping(RequestOptions.DEFAULT);
		} catch (IOException e) {
			e.printStackTrace();
		}
	}


	public Boolean doesExsist(GetRequest getRequest) throws IOException {
		return client.exists(getRequest, RequestOptions.DEFAULT);
	}

	@GET
	public String get() throws IOException {
		client.ping(RequestOptions.DEFAULT);
		return "hay desperados";
	}

	@GET
	@Path("{indexName}/by-id/{id}")
	public String findById(@PathParam("indexName") String indexName,
@PathParam("id") String id) throws IOException {
		GetRequest request = new GetRequest("testing", id); //indexname
set to testing for obvious reasons
		GetResponse documentFields = client.get(request,
RequestOptions.DEFAULT);
		LOGGER.info("get: {}", documentFields);
		return documentFields.toString();
	}

	@GET
	@Path("{indexName}/{query}")
	public String find(@PathParam("indexName") String indexName,
@PathParam("query") String query) throws IOException {
```

```
            SearchRequest searchRequest = new SearchRequest(indexName);
            SearchSourceBuilder searchSourceBuilder = new
SearchSourceBuilder();

            MatchQueryBuilder matchQueryBuilder = new
MatchQueryBuilder("ssn", query);
            matchQueryBuilder.fuzziness(Fuzziness.AUTO);
            matchQueryBuilder.prefixLength(3);
            matchQueryBuilder.maxExpansions(10);

            searchSourceBuilder.query(matchQueryBuilder);

            HighlightBuilder highlightBuilder = new HighlightBuilder();
            HighlightBuilder.Field highlightName =
                  new HighlightBuilder.Field("ssn");
            highlightName.highlighterType("unified");
            highlightBuilder.field(highlightName);

            searchSourceBuilder.highlighter(highlightBuilder);

            searchRequest.source(searchSourceBuilder);
            SearchResponse search = client.search(searchRequest,
RequestOptions.DEFAULT);
            LOGGER.info("got {} from {}: {}", search.getHits(), indexName,
search);
            return search.toString();
      }

      @POST
      @Path("{indexName}")
      public String postToIndex(@PathParam("indexName") String indexName,
String json) throws IOException {
            IndexRequest request = new IndexRequest("testing"); //set to
testing for obvious reasons
            request.source(json, XContentType.JSON);
```

```
            IndexResponse index = client.index(request,
RequestOptions.DEFAULT);
            LOGGER.info("indexed {}: {}", indexName, index);
            return index.toString();
      }

      @DELETE
      @Path("{indexName}")
      public String deleteIndex(@PathParam("indexName") String indexName)
throws IOException {
            DeleteIndexRequest request = new DeleteIndexRequest(indexName);
            AcknowledgedResponse delete = client.indices().delete(request,
RequestOptions.DEFAULT);
            LOGGER.info("deleted {}: {}", indexName,
delete.isAcknowledged());
            return String.valueOf(delete.isAcknowledged());
      }
}
```