TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

André Stender 185489IADB

Otsingusüsteemi loomine innovatsiooniprojektile Raamatud Liikuma

Bakalaureusetöö

Juhendajad: Margus Hanni
MSc
Jaanus Pöial
PhD

Tallinn 2021

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: André Stender

17.05.2021

Annotatsioon

Käesolev bakalaureusetöö jaguneb kolme eesmärgi vahel. Töö esimene eesmärk on luua serveripoolne rakendus, mis suhtleb väliste raamatukogusüsteemidega ning toetab loodavat otsingusüsteemi. Lisaks kuulub töö esimese eesmärgi juurde luua otsingute teostamiseks minimalistlik kliendirakendus. Töö teine eesmärk on analüüsi tulemusel valida otsingusüsteemi prototüüplahenduse loomiseks sobiv otsingutehnoloogia. Töö lõppeesmärk on kahe esimese eesmärgi tulemusel luua otsingusüsteemi prototüüp.

Serveripoolse rakenduse põhiülesanded on raamatukogusüsteemidest väljaannete pärimine, MARC standardiga struktureeritud väljaannete andmestike lugemine ning suhtlemine otsinguteenusega. Kliendirakendust kasutatakse prototüüplahenduses otsingu teostamiseks ning vastete kuvamiseks. Kliendirakenduse loomisel keskendutakse otsingu funktsionaalsetele omadustele.

Töö teoreetilises osas vaadeldakse mõningaid tehnoloogiaid, mida saab kasutada otsingusüsteemi loomiseks. Tehnoloogia valimiseks teostatakse põhjalikum analüüs, kasutades mitme-kriteeriumi analüüsi meetodeid.

Lõpptulemusena valmib otsingusüsteemi prototüüp, kasutades serveripoolset rakendust, kliendirakendust ning otsingutehnoloogiat. Valmiva prototüüplahenduse komponente kasutakse Raamatud Liikuma teenuse arendamiseks.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 89 leheküljel, 6 peatükki, 66 joonist, 25 tabelit.

Abstract

Creation of Search Engine for Innovation Project Raamatud Liikuma

Present Bachelor's thesis is divided into three goals. The first goal is to create a server-side application that communicates with external library systems and supports the functionalities of the search engine. Also, within the scope of the first goal is to create a simple client-side application that could be used to perform searches. The second goal is to perform an analysis to choose a search engine technology that will be used to build a prototype solution. As a result of the first two goals, the final goal is to create a search engine prototype.

The main tasks of the server-side application are fetching publications from external library systems, reading the data of publications that are structured according to MARC standards, and communicating with the search engine service. The client-side application is used in the prototype solution to perform searches and to display matches. On creating the client-side application, the focus is on the functional features of searching.

The thesis' theoretical part observes some technologies, that could be used to build a search engine. For technology decision-making, a more in-depth analysis is performed using multi-criteria decision analysis methods.

As a result, a search engine prototype is built using a server-side application, a client-side application, and search engine technology. The components of the prototype solution are used to develop the Raamatud Liikuma service.

The thesis is in Estonian and contains 89 pages of text, 6 chapters, 66 figures, 25 tables.

Lühendite ja mõistete sõnastik

.NET	Microsofti arendatud tarkvararaamistik.
AHP	*Analytic hierarchy process.* Analüütilise hierarhia protsess.
API	*Application programm interface.* Rakendusliides.
Auvis	Teavik, milles valdava osa moodustavad heli ja/või pilt ning mida saab vaadata ja/või kuulata eriseadmetega. [1]
Bibliokirje	Reeglipäraselt esitatud bibliograafiaandmete kogum, mis võimaldab kirjeldatavat dokumenti täielikult identifitseerida. [2]
BLL	*Business logic layer.* Äriloogika kiht.
DAL	*Data access layer.* Andmete ligipääsukiht.
DI	*Dependency injection*. Sõltuvuse sisestamine. Tarkvara disaini muster.
DTO	*Data transfer object.* Andmeedastusobjekt.
Eksemplar	Sama trükise iga üksik teisik. [3]
ERD	*Entity relationship diagram.* Olemi-suhte diagramm.
HTTP	*Hypertext Transfer Protocol.* Protokoll hajutatud hüpermeedia süsteemide jaoks. [4]
IDE	*Integrated development environment.* Integreeritud programmeerimiskeskkond.
JSON	*JavaScript Object Notation.* Programmeerimiskeelest sõltumatu kergekaaluline andmevahetusvorming. [5]
Levenshteini kaugus	Näitab mitu korda tuleb ühes sõnes tähte lisada, eemaldada või vahetada selleks, et ühest sõnest saaks teine sõne. [6]
Loose coupling	Nõrk komponentide vaheline sõltuvus.
MARC	*Marchine-readable cataloging.* Masinloetav kataloogimine.
ORM	*Object-relational mapping.* Objekt-relatsiooniline kaardistamine.
OSI	*Open Source Initiative*. Vabavaralise tarkvara kasutamist edendav korporatsioon. [7]
POCO	*Plain old CLR object.* Lihtne objekt .NET-i ühiskäituses, mis ei sõltu välistest raamistikest.
Presentation layer	Esitluskiht.

Repository pattern	Hoidla muster. Tarkvara disaini muster.
REST	*Representational state transfer.* Tarkvaraarhitektuuri stiil hajutatud hüpermeedia süsteemide jaoks. [8]
RL	Raamatud Liikuma.
Tarkvara paindlikkus	Lihtsus muuta süsteemi või komponenti, et seda saaks kasutada teistes rakendustes või keskkondades. [9]
Teavik	Mis tahes materiaalne objekt, millele on talletatud informatsioon, nt trükis, foto, heliplaat, kompaktketas. [10]
UOW	*Unit of work.* Tööühik.
URL	*Universal Resource Locator*. Universaalne nimemall, mida kasutatakse ressursside leidmiseks ja kasutamiseks Interneti kaudu. [11]
Väljaanne	Üldnimetus levitamiseks määratud teose või teaviku kohta. [12]
WPM	*Weighted product model.* Kaalutud korrutamise mudel.
XML	*Extensible Markup Language.* Standardne üldotstarbeline märgistuskeel, mida kasutatakse struktureeritud info jagamiseks infosüsteemide vahel.

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Innovatsiooniprojekti Raamatud Liikuma raames arendatakse kaasaegset tehnoloogilist lahendust raamatukogulaenutuse teenusele. Projekti eesmärk on ühendada Eesti raamatukogude kollektsioonid ühte lihtsasse otsingusse, mis töötab olemasolevate e-kataloogide peal ning võimaldab kasutajatel soovitud raamatuid koju tellida kõikjalt Eestist. [13]

Projekti Raamatud Liikuma tulemusena valmib vaheliides raamatukogude jaoks ning veebikeskkond ja mobiilirakendus, kus kasutaja saab teostada otsingu, esitada tellimuse, jälgida tellimuse staatust, saada teavitusi raamatu saabumisest ja tagastustähtaegadest, loetud raamatuid sõpradele soovitada jne. Raamatukogudesse hangitud teavikud muutuvad mobiilseks ning lugeja koduraamatukoguks saab terve Eesti raamatukoguvõrk. [13]

1.1 Probleem

Eesti raamatukogud on koondunud peamiselt kolme suuremasse raamatukogusüsteemi:

1. Sierra – Sierral põhineb e-kataloog ESTER. Sierrat kasutab 30 raamatukogu, peamiselt suuremad Eesti raamatukogud, sh ülikoolide raamatukogud.
2. RIKS – RIKSi kasutab kokku 510 raamatukogu, sh erialaraamatukogud, kooliraamatukogud, rahvaraamatukogud ja lasteaedade raamatukogud [14].
3. Urram – Urramit kasutab kokku 335 raamatukogu, sh keskraamatukogud, rahvaraamatukogud, kooliraamatukogud ja erialaraamatukogud [15].

RL teenuse arendamise raames luuakse kolme eelnimetatud raamatukogusüsteemi abil uus teavikute ühiskataloog. Uue ühiskataloogi loomisel on RL teenuse tarbeks tarvis andmed kolmest raamatukogusüsteemist kokku koguda. Lisaks on tarvis andmeid jooksvalt uuendada.

Raamatukogudes kataloogitakse ning raamatukogusüsteemides talletatakse bibliokirjeid MARC 21 standardi alusel. Sellest tulenevat tuleb leida viis, kuidas lugeda MARC kujul bibliokirjete andmestikke.

Uuest ühiskataloogist saab tulevikus lõppkasutaja otsida teavikuid ja soovi korral neid endale koju tellida. Teavikute otsimiseks luuakse uuel ühiskataloogil põhinev otsingusüsteem. Lahendus peab olema lõppkasutajasõbralik – see tähendab, et arvestama peab kindlasti, et otsingu aeg oleks lühike ning otsingu vasted oleks asjakohased.

1.2 Eesmärk

Töö koostamisel ajal oli RL projektis käsil etapp I, kus toimus põhiliselt teenuse analüüs, IT-süsteemide kaardistus ning nõuete kogumine. Arendustööd on planeeritud etappidesse II-III.

Töö esimene eesmärk on luua valmis projekti põhi, millega saaks alustada etappidesse II-III planeeritud arendustöid ning mis toetaks ka antud töö lõppeesmärki. Esimese eesmärgi saavutamiseks luuakse REST arhitektuuril põhinev serveripoolne rakendus, mis pärib bibliokirjeid välistest andmeallikatest, loeb ja töötleb MARC kujul kirjete andmestikke ning suhtleb otsinguteenusega. Lisaks luuakse minimalistlik kliendirakendus, kus keskendutakse otsingu funktsionaalsetele komponentidele.

Töö teine eesmärk on valida otsingusüsteemi loomiseks otsingutehnoloogia. Selle käigus tutvutakse lähemalt erinevate alternatiividega ning koostatakse parima alternatiivi leidmiseks põhjalik analüüs.

Töö lõppeesmärk on kahe esimese eesmärgi täitmise tulemusena luua otsingusüsteemi prototüüp.

2 Metoodika

Töös luuakse REST arhitektuurist lähtuvalt serveripoolne rakendus. Serveripoolne rakendus põhineb mitmekihilisel arhitektuuril ning selle loomisel on arvestatud puhta koodi printsiipidega.

Bibliokirjete vaheteenusesse kataloogimiseks kasutatakse Sierra API-t. RIKSi- ja Urrami API-t käesoleva töö koostamise ajal vajalikus mahus ei eksisteerinud ning ei täitnud seeläbi vajaminevaid andmevahetuse nõudeid. RIKSi- ja Urrami API nõuete seadmisel võetakse paljuski aluseks olemasolev Sierra API, mis tähendab, et töös koostatud päringumeetodite näiteid ja nende andmekoosseise saab tulevikus kasutada ka RIKSist ja Urramist andmete pärimiseks.

Andmete pärimine ja uuendamine on oluline osa RL teenusest. Raamatukogusüsteemidest päritud andmed salvestatakse RL-i relatsioonilisse andmebaasi ning otsingusüsteemi indeksisse. Sierrast andmete pärimiseks on kaks erinevat võimalust, millest mõlemal on omad plussid ja miinused. Kahe erineva päringumetoodika võrdlemiseks koostatakse jõudlustestid. Jõudlustestid teostatakse serveripoolses rakenduses ning testide koostamiseks kasutatakse xUnit raamistikku.

Raamatukogusüsteemidest päritud kirjed on struktureeritud MARC standardi alusel. MARC kujul andmestikuga töötamiseks kasutatakse käesolevas töös MARC4J.Net [16] teeki. Teek võimaldab lihtsustatud kujul lugeda ja kirjutada MARC struktuuriga andmestikku. Teek toetab MARC ja MARCXML formaate.

Otsingu demonstreerimiseks luuakse minimalistlik kliendirakendus. Kliendirakenduse loomisel on keskendutud otsingu funktsionaalsetele omadustele.

Otsingusüsteemi loomiseks valitakse välja otsingutehnoloogia. Otsingutehnoloogia valimiseks teostatakse antud töös põhjalikum analüüs. Töös kasutatud analüüsi metoodika on kirjeldatud peatükis 2.2 Mitme-kriteeriumi analüüs. Otsingutehnoloogiate analüüs toimub peatükis 4.3 Otsingutehnoloogiate analüüs.

2.1 Tehnoloogiad ja arendusvahendid

Tehnoloogiate ja arendusvahendite valikul on arvestatud autori ning teiste projektis töötavate arendajate kogemusi ja oskusi. Lisaks on valikute puhul arvestatud, et tehnoloogiate ja arendusvahendite funktsioonid aitaks saavutada RL teenuse eesmärke. Tabelis 1 on toodud välja peamised projektis kasutatud tehnoloogiad ja arendusvahendid. Tabelis ei ole esitatud otsingutehnoloogiat, see selgub alles peale põhjalikumat analüüsi peatükis 4.3 Otsingutehnoloogiate analüüs.

Tabel 1. Projektis kasutatud tehnoloogiad ja arendusvahendid.

Serveripoolne rakendus	.NET (C#)
Andmehoidla	SQL Server
ORM	Entity Framework
Teek MARC andmestiku lugemiseks serveripoolses rakenduses	MARC4J.Net
Testimise raamistik serveripoolses rakenduses	xUnit
Kliendirakendus	TypeScript, React
API testimine	Postman
IDE	JetBrains Rider, Visual Studio Code
Andmebaasi tööriistad	Azure Data Studio

2.2 Mitme-kriteeriumi analüüs

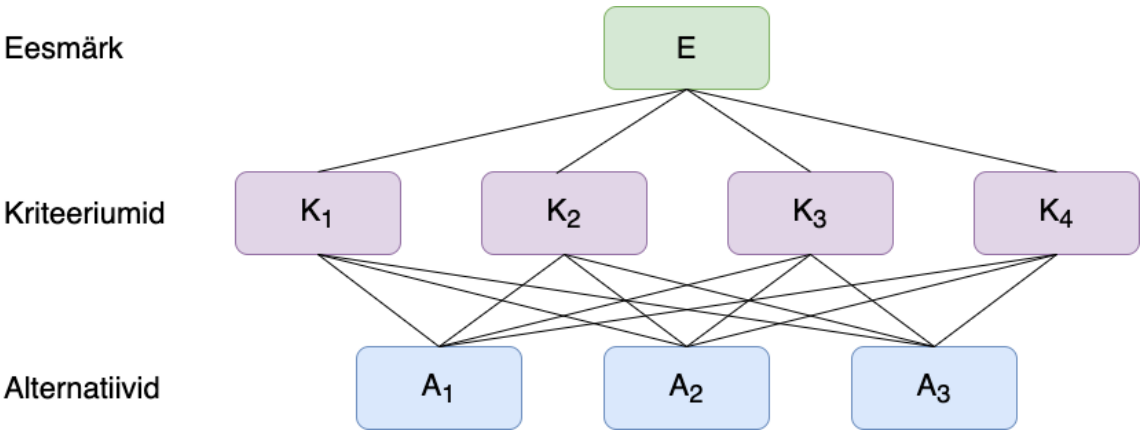
Töös käsitletakse erinevaid otsingutehnoloogiaid ning analüüsitakse tehnoloogiate tugevusi ja nõrkuseid nii üleüldiselt kui ka käesoleva projekti seisukohalt. Otsingutehnoloogiaid analüüsitakse mitme-kriteeriumi analüüsi meetoditega, et anda tehnoloogiatele objektiivne ja arvuline hinnang. Töös kasutatakse analüüsimiseks AHP ja WPM kombineeritud meetodit.

2.2.1 AHP

AHP on meetod otsuste tegemiseks. AHP on disainitud sellisena, et parima valiku väljaselgitamisel võtab see korraga arvesse nii ratsionaalsust kui ka intuitiivsust. Fundamentaalselt koosneb AHP kolmest tasandist [17, pp. 1-2]:

1. eesmärk;
2. kriteeriumid;
3. alternatiivid.

Otsuse eesmärk kirjeldab, mis probleemi üritatakse valikuga lahendada. Otsuse tegemine sõltub kriteeriumitest – need on faktorid, mis omavad otsuse tegemisel tähtsust. Kriteeriumitele antakse kaalud vahemikus 0-1 ning kaalude kogusumma peab olema 1. Kriteeriumi kaal määrab, kui oluline on kriteerium otsuse tegemisel võrreldes teiste kriteeriumitega. Kriteeriumite ja kaalude alusel arvutatakse igale valikule arvuline väärtus. Tulemuseks on arvuliste väärtustega alternatiivid, mis aitavad otsuse langetamisel selgusele jõuda. Joonisel 1 on näide AHP hierarhiast.



Joonis 1. Näide: AHP hierarhia.

2.2.2 WPM

Alternatiivide tulemuste leidmiseks kasutatakse käesolevas töös WPM meetodit. WPM on dimensioonitu mitme-kriteeriumi analüüsimise meetod, mida saab kasutada parima alternatiivi leidmiseks ilma võrdluseid teostamata [18, pp. 8-9]. Käsitletava probleemi raames ei ole tarvis teostada alternatiivide võrdluseid ning parima alternatiivi leidmiseks pakub WPM siinkohal lihtsat ja efektiivset lahendust. WPM on osutunud väga usaldusväärseks meetodiks ning selle efektiivsust on testitud kolme kuni saja kriteeriumi korral [19]. WPM meetodi puhul kasutatakse väärtuste leidmiseks järgnevat valemit [18]:

$$P(A_i) = \prod_{j=1}^{n} (a_{ij})^{w_j} \quad (2.1)$$

Võrrandis 2.1 tähistab $P(A_i)$ alternatiivi tulemust, n kriteeriumite arvu, a_{ij} alternatiivi kriteeriumi väärtust ning w_j kriteeriumi kaalu.

2.2.3 Kriteeriumite kaalude leidmine

Kriteeriumite analüüsimiseks ja kriteeriumite kaalude leidmiseks kasutatakse käesolevas töös AHP meetodit. Selle käigus teostatakse kriteeriumite paarikaupa võrdlused - see tähendab, et ükshaaval võrreldakse igat kriteeriumit iga teise kriteeriumiga ning määratakse arvuline väärtus, mis tähistab, kui palju üks kriteerium on tähtsam või olulisem teisest. Tabelis 2 esitatud Saaty skaala paarikaupa võrdluste tegemiseks [17, p. 6]:

Tabel 2. Saaty skaala.

Intensiivsus	**Definitsioon**	**Selgitus**
1	Võrdselt tähtis	Panus on võrdne eesmärgi saavutamisel
2	Nõrk paremus või tähtsus	
3	Mõõdukas paremus või tähtsus	Kogemuse ja hinnangu põhjal eelistatakse kergelt ühte teisele
4	Mõõdukas-suur paremus või tähtsus	
5	Suur paremus või tähtsus	Kogemuse ja hinnangu põhjal eelistatakse tugevalt ühte teisele
6	Üsna suur paremus või tähtsus	
7	Väga suur paremus või tähtsus	Praktika näitab, et eelistatakse väga tugevalt ühte teisele
8	Väga-väga suur paremus või tähtsus	
9	Ekstreemne paremus või tähtsus	Tõendid kinnitavad suurimat võimalikku eelistust teise üle

Peale paarikaupa võrdlemisi tekib võrdlusmaatriks, mille põhjal saab tuletada kriteeriumite kaalud. Kaalude leidmiseks on erinevaid tehnikaid, käesolevas töös kasutatakse Saaty poolt välja pakutud tehnikat. Esmalt normaliseeritakse võrdlusmaatriks, jagades maatriksi igat elementi vastava veeru summaga. Kriteeriumi kaal saadakse, kui võetakse normaliseeritud võrdlusmaatriksist kriteeriumi rea keskmine väärtus [17, p. 8].

2.2.4 Kriteeriumite alternatiivide väärtuste leidmine ja normaliseerimine

Kriteeriumeid on kahte tüüpi – positiivsed kriteeriumid, mille puhul suurem väärtus on parem ning negatiivsed kriteeriumid, mille puhul väiksem väärtus on parem. Kriteeriumite puhul peab arvestama ka sellega, et need võivad olla omadustelt mittelineaarsed või kvalitatiivsed. Tabelis 3 on esitatud Hwangi ja Yooni poolt pakutud skaala [20, pp. 27-28], mida saab selliste kriteeriumite puhul kasutada alternatiivide väärtuste määramiseks.

Tabel 3. Skaala kriteeriumite alternatiivide väärtuste määramiseks.

	Väga kõrge		**kõrge**		**Keskmine**		**madal**		**Väga madal**
Positiivne kriteerium	9		7		5		3		1
Negatiivne kriteerium	1	2	3	4	5	6	7	8	9

WPM on küll olemuselt dimensioonitu, kuid käesoleva probleemi puhul on negatiivsete kriteeriumite tõttu tarvis teostada ka normaliseerimine. Üks võimalus on normaliseerimiseks kasutada lineaarset transformatsiooni [20, pp. 30-31]. Sellisel juhul, olgu $\max(x_j)$ suurim kriteeriumi alternatiivi väärtus ja x_{ij} otsitava kriteeriumi alternatiivi normaliseerimata väärtus, siis positiivse kriteeriumi alternatiivi normaliseeritud väärtus a_{ij} on:

$$a_{ij} = \frac{x_{ij}}{max(x_j)} \tag{2.2}$$

Olgu $min(x_j)$ väikseim kriteeriumi alternatiivi väärtus ja x_{ij} otsitava kriteeriumi alternatiivi normaliseerimata väärtus, siis negatiivse kriteeriumi alternatiivi normaliseeritud väärtus a_{ij} on:

$$a_{ij} = \frac{min(x_j)}{x_{ij}} \tag{2.3}$$

2.2.5 Otsustusmaatriks

Probleemi visuaalseks esitamiseks ja väärtuste arvutamiseks kasutatakse antud töös otsustusmaatriksit. See on $m \times n$ maatriks, kus element a_{ij} tähistab kriteeriumi alternatiivi väärtust [18, pp. 1-3]. Otsustusmaatriksit saab esitada nii normaliseerimata kui ka normaliseeritud kujul. Käesolevas töös kasutatakse normaliseeritud otsustusmaatriksi esitamiseks eelmises peatükis välja toodud normaliseerimise võtteid (vt 2.2.4 Kriteeriumite alternatiivide väärtuste leidmine ja normaliseerimine). Tabelis 4 on esitatud näide otsustusmaatriksist, kus K tähistab kriteeriumit, w kriteeriumi kaalu ning A alternatiivi.

Tabel 4. Näide: otsustusmaatriks.

	K_1 (w_1)	**K_2 (w_2)**	**K_3 (w_3)**	**K_4 (w_4)**
A_1	a_{11}	a_{12}	a_{13}	a_{14}
A_2	a_{21}	a_{22}	a_{23}	a_{24}
A_3	a_{31}	a_{32}	a_{33}	a_{34}

2.2.6 AHP ja WPM kombineeritud meetod

Käesolevas töös kasutatakse otsingutehnoloogiate analüüsimiseks järgnevaid samme:

1. selgita välja otsuse eesmärk, kriteeriumid ning alternatiivid;
2. leia kriteeriumite kaalud AHP meetodiga;
3. esita kriteeriumite alternatiivide väärtused;
4. esita normaliseerimata ja normaliseeritud otsustusmaatriksid ning arvuta alternatiivide tulemused WPM meetodiga;
5. järeldused.

3 Vaheteenuse loomine

Käesolevas peatükis käsitletakse RL otsingusüsteemi puudutava vaheteenuse loomist. Esimeses kahes alampeatükis on esitatud RL teenuse üldine arhitektuur ning andmemudel, andmaks aimu, millest lähtuvalt luuakse käesoleva töö praktiline lahendus. Lisaks käsitletakse antud peatüki raames serveripoolse rakenduse ja kliendirakenduse loomist.

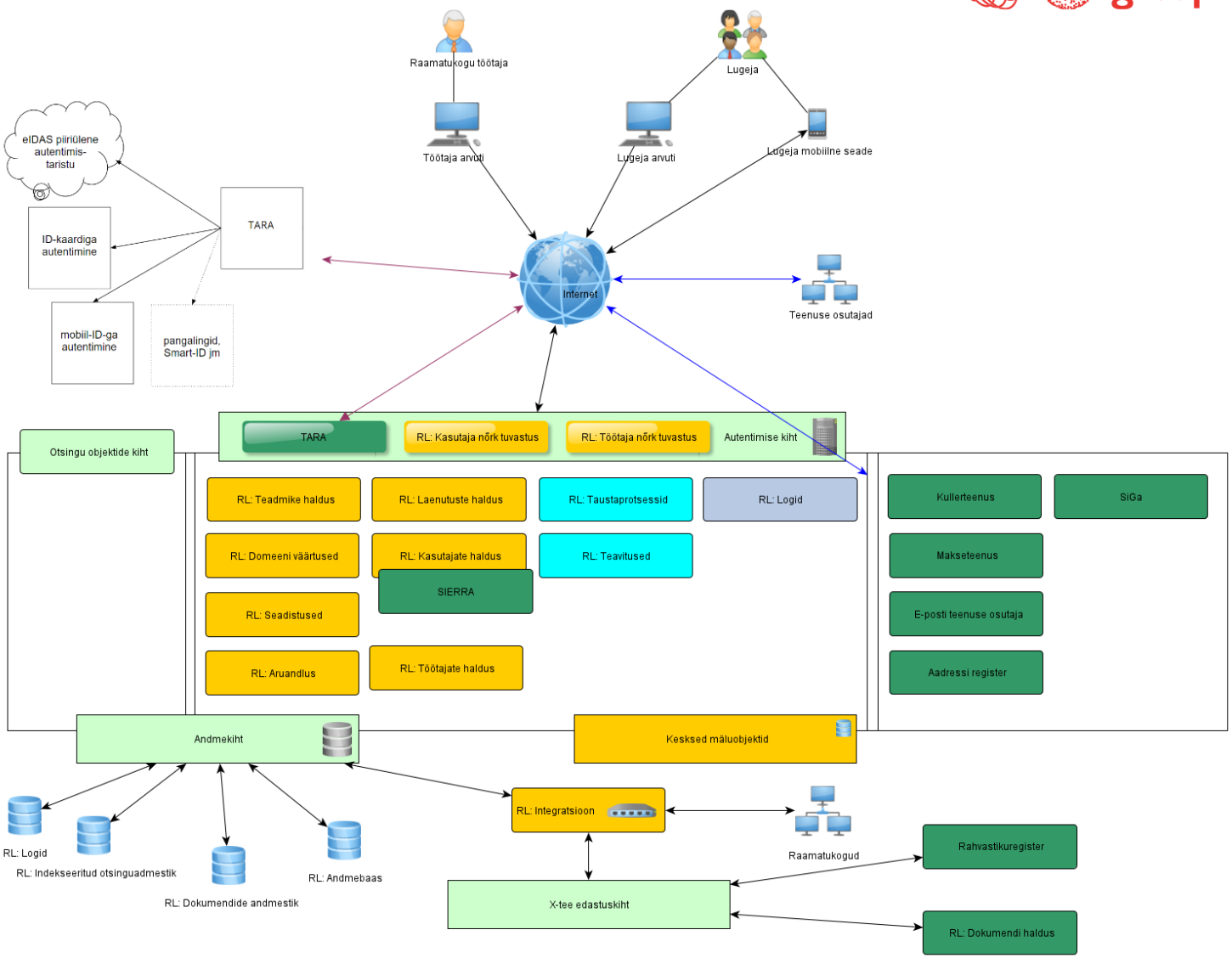
3.1 Teenuse arhitektuur

Liidese põhimõtted ja lahendus tulevad REST veebistandarditel põhinevast arhitektuurist ja HTTP protokollist. Arhitektuurist tulenevalt ei oma erinevad liidesed omavahelisi tugevaid seoseid. Andmete edastamise üldine kontseptsioon on olekuvaba, mis tähendab seda, et osapooled ei tea omavahelisest andmevahetuse võimalikust algandmekoosseisu ehk osapooled ei oma integratsiooni vaates sessioone.

REST arhitektuuril baseeruvaid teenuseid nimetatakse RESTful programmilisteks liidesteks, mis peavad järgima järgmist kuut põhimõtet [21], [22]:

1. ühtne liides;
2. klient-server lahendus;
3. olekuvaba;
4. vahemälu kasutusvõimalus;
5. kihilisus;
6. vajaduspõhine kood.

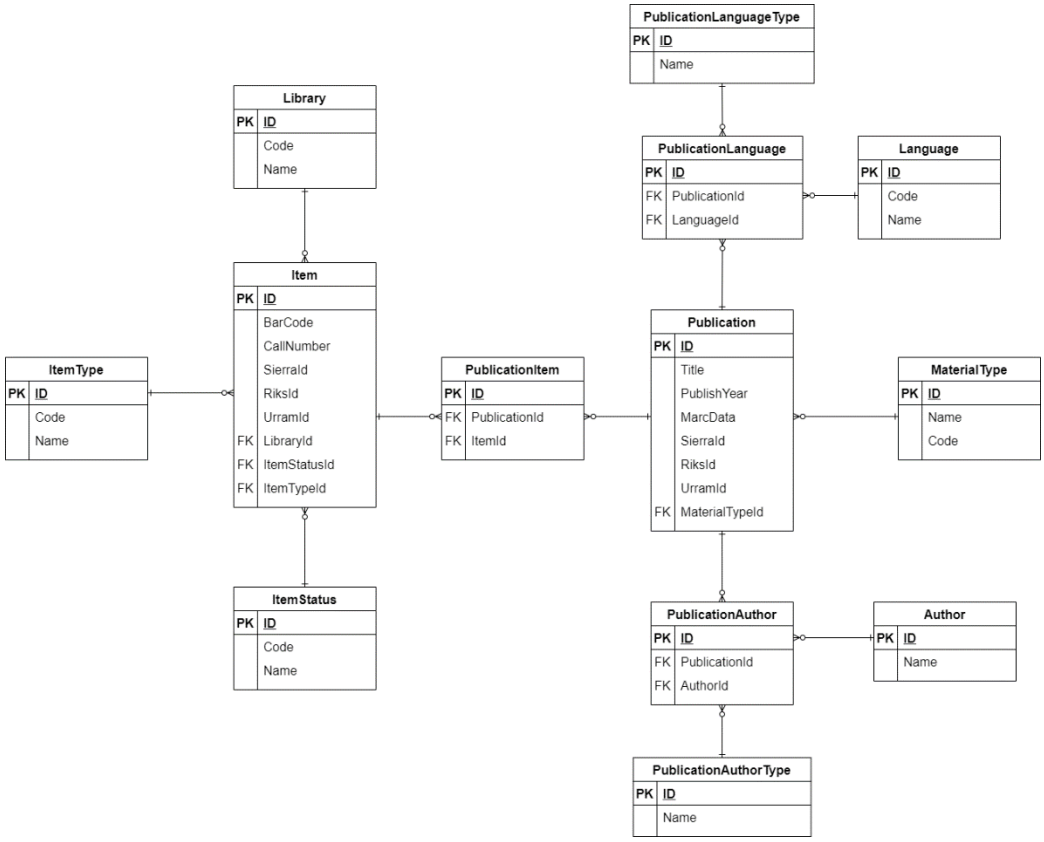
Allpool joonisel 2 on Net Groupi koostatud RL teenuse arhitektuurijoonis. Joonisel tähistab oranž ärilisi komponente, heleroheline tehnoloogilisi komponente, helesinine taustakomponente, hall toekomponente ning tumeroheline liidestuskomponente.



Joonis 2. Net Groupi koostatud RL arhitektuurijoonis.

3.2 Andmemudel

Antud töös esitatud andmemudel kuulub Net Groupile. See on valminud põhinedes tellija projektimeeskonna poolt esitatud nõuetest. Andmemudel ei pruugi olla lõplik ega täielik, sest nõuded võivad ajas muutuda. Lisa 2 all on täiendavalt toodud välja Net Groupi loodud RL kontseptuaalne andmemudel, kuid üksikasjalik ülevaade terviklikust andmemudelist jääb antud töö skoobist välja. Antud töös käsitletakse lähemalt ainult seda osa andmemudelist, mis puudutab otsingusüsteemi. Joonisel 3 on esitatud väljaannete ERD.



Joonis 3. RL väljaannete ERD.

Järgnevalt on tabelis 5 antud ülevaade väljaannete ERD olemitest. Tabelis on toodud olemitele vastavad tabelite nimed ning semantikad. Semantika all on välja toodud olemi tüüp (olem võib olla antud juhul objekt-, subjekt-, teatmik- või seos-tüüpi [23]), olemi tähistus reaalses maailmas, olemi eesmärk ning näide.

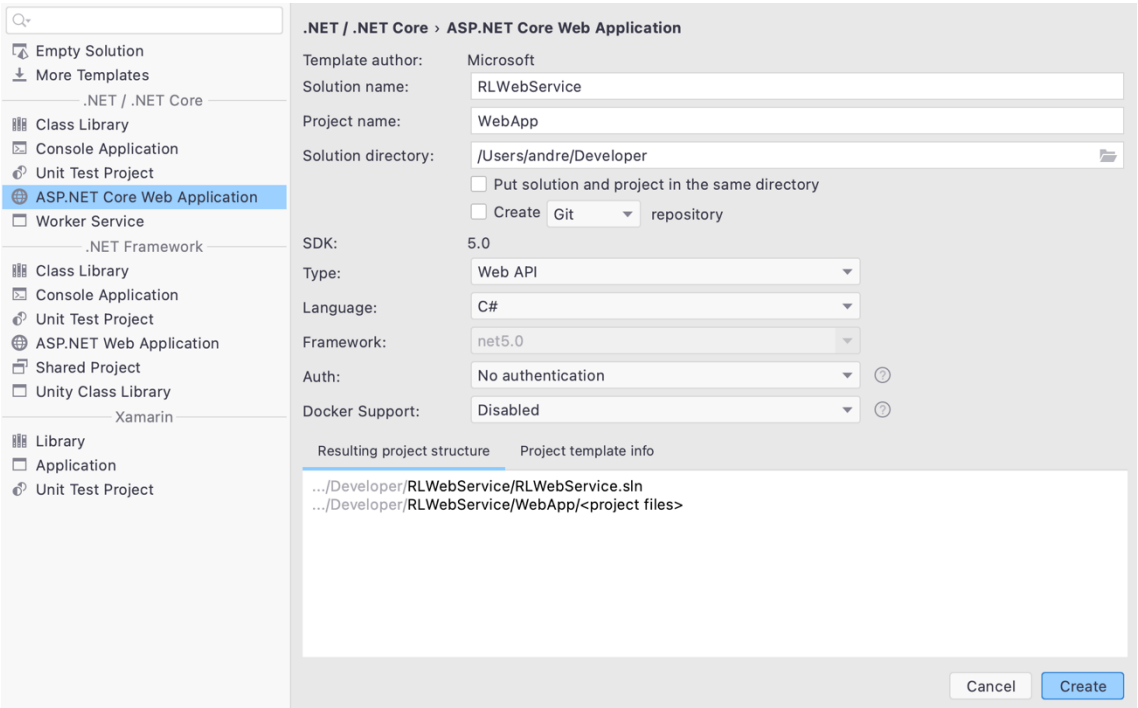
Tabel 5. Olemite semantikad.

Tabeli nimi	**Semantika**
Author	Subjekt-tüüpi olem, mis tähistab (väljaande) autorit. Selle olemi eesmärk on luua autorite teatmik. Autor võib olla näiteks J. K. Rowling.
Item	Objekt-tüüpi olem, mis tähistab eksemplari. Selle olemi eesmärk on luua eksemplaride teatmik. Eksemplar võib olla näiteks „Harry Potter ja Tarkade Kivi“ füüsiline trükis.
ItemStatus	Teatmik-tüüpi olem, mis tähistab Eksemplari staatust. Selle olemi eesmärk on luua eksemplari staatuste teatmik. Eksemplari staatus võib olla näiteks saadaval või ainult raamatukogus kasutamiseks.
ItemType	Teatmik-tüüpi olem, mis tähistab Eksemplari tüüpi. Selle olemi eesmärk on luua eksemplari tüüpide teatmik. Eksemplari tüüp võib olla näiteks raamat või veebiressurss.
Language	Teatmik-tüüpi olem, mis tähistab keelt. Selle olemi eesmärk on luua keelte teatmik. Keel võib olla näiteks eesti keel, vene keel või inglise keel.
Library	Objekt-tüüpi olem, mis tähistab raamatukogu. Selle olemi eesmärk on luua raamatukogude teatmik. Raamatukogu võib olla näiteks Eesti Rahvusraamatukogu.
MaterialType	Teatmik-tüüpi olem, mis tähistab teaviku laadi. Selle olemi eesmärk on luua teaviku laadide teatmik. Teaviku laad võib olla näiteks raamat, ajakiri, kaart või auvis.
Publication	Objekt-tüüpi olem, mis tähistab väljaannet. Selle olemi eesmärk on luua väljaannete teatmik. Väljaanne võib olla näiteks 2000. aastal ilmunud eesti keelne teos „Harry Potter ja Tarkade Kivi“.
PublicationAuthor	Seos-tüüpi olem, mis tähistab seost väljaande ja autori vahel. Selle olemi eesmärk on luua mitu-mitmene seos, sest ühel väljaandel võib olla mitu autorit ning ühel autoril võib olla mitu väljaannet. Olem võib kirjeldada näiteks 2000. aastal ilmunud eesti keelse väljaande „Harry Potter ja Tarkade Kivi“ ning autori J. K. Rowlingu vahelist autorluse seost.
PublicationAuthorType	Teatmik-tüüpi olem, mis tähistab väljaande autori tüüpi. Selle olemi eesmärk on luua väljaande autorite teatmik. Väljaande autori tüüp võib olla näiteks põhiautor.

Tabeli nimi	Semantika
PublicationItem	Seos-tüüpi olem, mis tähistab seost väljaande ja eksemplari vahel. Selle olemi eesmärk on luua mitu-mitmene seos, sest ühel väljaandel võib olla mitu eksemplari ning üks eksemplar võib olla seotud mitme väljaandega. Olem võib kirjeldada näiteks seost 2000. aastal ilmunud eesti keelse väljaande „Harry Potter ja Tarkade Kivi“ ning füüsilise raamatu eksemplari vahel.
PublicationLanguage	Seos-tüüpi olem, mis tähistab seost väljaande ja keele vahel. Selle olemi eesmärk on luua mitu-mitmene, sest väljaanne võib olla mitmes keeles ning ühes keeles võib olla palju väljaandeid. Olem võib kirjeldada näiteks väljaande „Tõde ja õigus“ ning eesti keele vahelist seost.
PublicationLanguageType	Teatmik-tüüpi olem, mis tähistab väljaande keele tüüpi. Selle olemi eesmärk on luua väljaande keele tüüpide teatmik. Väljaande keele tüüp võib olla näiteks põhikeel.

3.3 Serveripoolne rakenduse

Serveripoolse rakenduse arendamiseks kasutatakse Microsofti .NET raamistikku. Uue serveripoolse rakenduse projekti loomiseks kasutab autor JetBrains Rider arenduskeskkonda (Joonis 4). Valitud seadistusega luuakse lihtne veebirakendus, mis toetab töö edasist käiku.



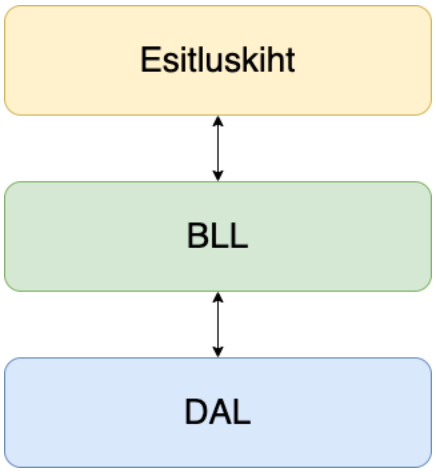
Joonis 4. Serveripoolse rakenduse projekti loomine.

3.3.1 Serveripoolse rakenduse arhitektuur

Teenusepõhisest ülesehitusest tulenevalt baseerub serveripoolne rakendus klassikalisel mitmekihilisel arhitektuuril [24]:

- esitluskiht (inglise k. *presentation layer*);
- äriloogika kiht (inglise k. *business logic layer*, lühend. BLL);
- andmete ligipääsukiht (inglise k. *data access layer*, lühend DAL).

Joonisel 5 on esitatud serveripoolse rakenduse kihid.



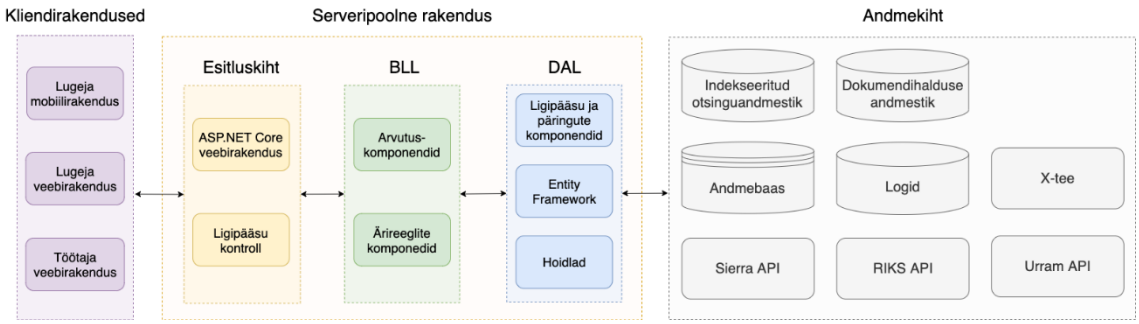
Joonis 5. Serveripoolse rakenduse kihid.

Esitluskiht pakub läbi kasutajaliidese rakendusele ligipääsu ning vastutab andmete esitluse eest. Rakenduse siseselt saab esitluskiht suhelda vaid äriloogika kihiga.

Äriloogika kiht on rakenduse põhiline ja keskne osa mis suhtleb kahe välimise kihiga – esitluskihiga ja DAL kihiga. Äriloogika kihis tehakse kõik vajalikud arvutused, et edastada andmed õigel kujul järgmisele kihile.

Andmete ligipääsukiht vastutab väliste andmeallikatega suhtlemise eest. Välisteks andmeallikateks võivad olla andmebaasid, välised API-d, failid jmt. RL teenuse raames on välisteks andmeallikateks RL andmebaas, logide andmebaas, otsingusüsteemi indekseeritud andmestik, raamatukogusüsteemide API-d ja X-tee. Rakenduse siseselt saab andmete ligipääsukiht suhelda vaid äriloogika kihiga.

Järgnevalt on joonisel 6 esitatud põhjalikum serveripoolse rakenduse mitmekihiline arhitektuurijoonis.



Joonis 6. Serveripoolse rakenduse mitmekihiline arhitektuurijoonis.

3.3.2 Serveripoolse rakenduse disainimustrid

Rakenduse loomisel on lähtutud objektorienteeritusest ning puhta koodi põhimõtetest, eelkõige SOLID printsiipidest [25, pp. 57-59]:

- *Single Responsibility Principle* – üksiku vastutuse printsiip;
- *Open-Closed Principle* – avatud-suletud printsiip;
- *Liskov Subsitution Principle* – Liskovi asendusprintsiip;
- *Interface Segregation Principle* – liideste eraldatuse printsiip;
- *Dependency Inversion Principle* – sõltuvuse inversiooni printsiip.

Dependency injection

Serveripoolses rakenduses on kasutusel DI *(dependency injection*) arhitektuurimuster. Selle abil on võimalik nõrgestada omavahelisi sõltuvusi, parandades sellega tunduvalt koodi hooldatavust. DI 3 põhilist omadust on [26]:

1. objektide komponeerimine;
2. objektide elutsüklite haldamine;
3. objektide peatamine.

Rakenduses on kasutusel Microsofti sisse ehitatud DI. Microsofti DI-s on võimalik teenuseid registreerida järgnevate elutsüklitega [27]:

- *Transient* – iga kord, kui teenust küsitakse luuakse uus instants;
- *Scoped* – iga uue kliendi päringu korral luuakse uus instants;
- *Singleton* – kogu rakenduse eluaja jooksul luuakse ja kasutatakse ainult ühte instantsi.

Rakenduses on loodud teenuste registreerimiseks laiendusmeetodid. Laiendusmeetodite eesmärk on kapseldada kokku lähestikku asuvate teenuste registreerimine. Järgneval joonisel 7 on esitatud teenuste registreerimise laiendusmeetod DAL kihi jaoks.

```
public static class DalServiceInstaller
{
    public static IServiceCollection AddDal(
        this IServiceCollection services, IConfiguration configuration)
    {
        services.AddDbContext<AppDbContext>(options =>
               options.UseSqlServer(
                      configuration["ConnectionStrings:AzureSqlEdge"]));
        services.AddScoped<IUnitOfWork, UnitOfWork>();
        services.AddScoped<ISierraRepositoryCollection,
               SierraRepositoryCollection>();
        services.AddScoped<IRiksRepositoryCollection,
               RiksRepositoryCollection>();
        services.AddScoped<IUrramRepositoryCollection,
               UrramRepositoryCollection>();

        return services;
    }
}
```

Joonis 7. Teenuse registreerimise laiendusmeetod.

Hoidla muster

Andmete ligipääsu kihis on kasutusel hoidla muster (inglise k. *repository pattern*). Kontseptuaalselt, hoidla kapseldab andmehoidla objektid ja nendele teostatavad toimingud [28, pp. 322-325]. Antud projektis on lisaks andmebaasi hoidlatele ka teisi hoidlaid, sealhulgas Sierra API hoidla, RIKSi API hoidla, Urrami API hoidla ning otsingusüsteemi API hoidla. Hoidlatest on moodustatud kollektsioonid, et muuta need DI abil lihtsasti ja mugavalt kättesaadavaks äriloogika kihile.

Tööühiku muster

Lisaks on serveripoolses rakenduses kasutusel tööühiku *(UOW – unit of work)* muster. UOW säilitab äri transaktsiooni poolt mõjutatud objektide listi ning koordineerib muudatuste väljakirjutamist ja samaaegsusega soetud probleemide lahendamist. Äri transaktsiooni raames peab UOW arvestust kõige üle, mis võib mõjutada andmebaasi ning kinnituskäskluse edastamisel viiakse muudatused andmebaasi. [28, pp. 184-194]

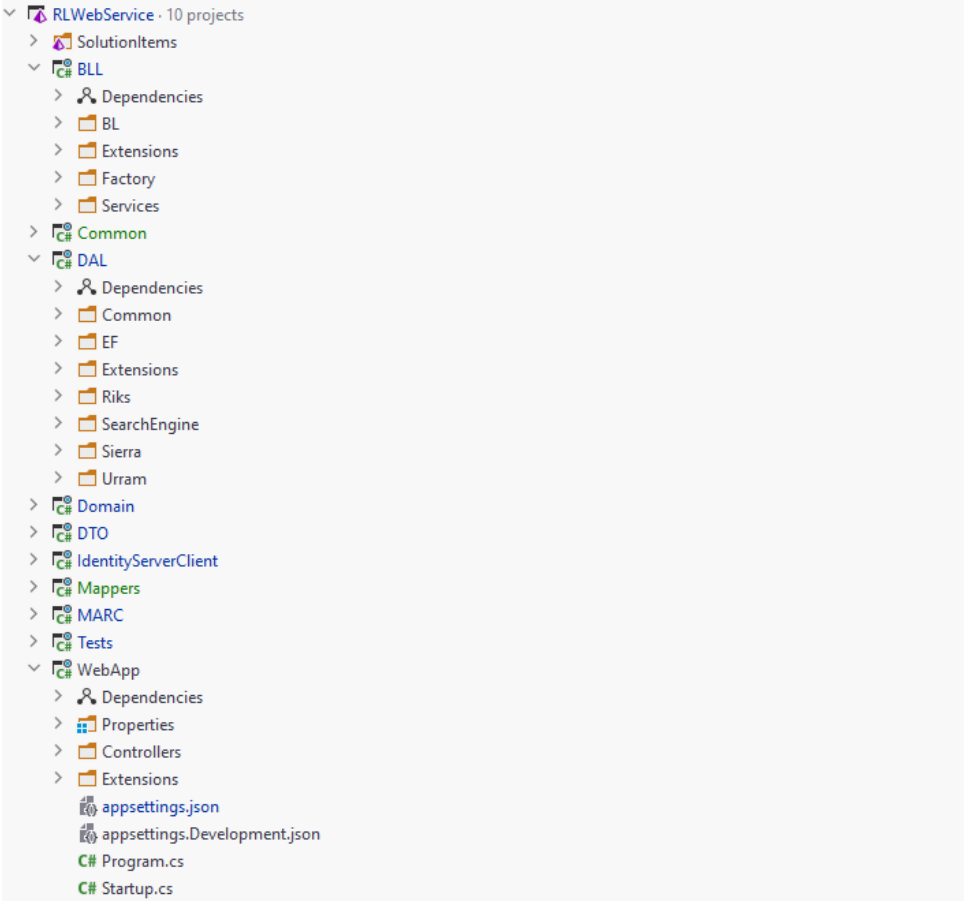
Entity Frameworki sisseehitatud DbContext rakendab hoidla ja UOW mustreid [29]. Sellegipoolest on lisatud projekti DAL kihti täiendavad hoidlad ja UOW, et hoida äriloogika kiht puhtam ning vähendada korduvat koodi.

3.3.3 Projekti struktuur

Projekt on organiseeritud lähtudes mitmekihilisest arhitektuurist. Lahendus koosneb 9-st projektist:

- BLL – äriloogika kiht. Koosneb peamiselt teenustest, äriloogika komponentidest, arvutuskomponentidest ja valideerimismeetoditest;
- Common – sisaldab projektide ühiskasutatavaid komponente, mille hulka kuuluvad näiteks konstandid ja laiendusmeetodid;
- DAL – andmete ligipääsukiht. Koosneb peamiselt hoidlatest;
- Domain – sisaldab andmemudeli klasse;
- DTO – sisaldab DTO klasse;
- IdentityServerClient – abistav projekt arendustöödeks, mis talletab ja uuendab automaatselt arendajate API võtmeid;
- Mappers – sisaldab kaardistusmeetodeid;
- MARC4J.Net – projekt MARC andmestiku töötlemiseks;
- Test – testide projekt. Sisaldab jõudlus- ja integratsiooniteste rakenduse olulisemate funktsionaalsuste testimiseks;
- WebApp – veebirakendus.

Järgneval joonisel 8 on esitatud projekti kataloogistruktuur.



Joonis 8. Projekti kataloogistruktuur.

3.4 Väljaannete päringute jõudlustest

Väljaannete MARC andmestikku on võimalik pärida JSON või XML kujul. Eelistatud on XML, sest siis on võimalik kasutada MARC4J.Net teeki, mis võimaldab lugeda ja lihtsustatud moel liigendada XML kujul MARC andmestikku. JSON kujul MARC andmestiku lugemiseks ja liigendamiseks töö autor ühtegi teeki ei leidnud, kuid võimalus oleks täiendada mõnda olemasolevat vabavaralist teeki. XML kujul pärimise miinuseks on see, et iga väljaande kohta tuleb MARC andmestik küsida juurde eraldi päringuga. JSON-i puhul on võimalus panna MARC andmestik esmasesse päringusse. Tabelis 6 on võetud kokku JSON-i ja XML-i põhise lähenemise plussid ja miinused.

Tabel 6. Päringumetoodikate plussid ja miinused.

Meetod	**Plussid**	**Miinused**
MARC andmestiku pärimine XML kujul	Võimalik kasutada teeki (MARC4J.Net), mis võimaldab lugeda ja lihtsustatud moel liigendada XML kujul MARC andmestikku.	Iga väljaande kohta tuleb MARC andmestik küsida juurde eraldi päringuga. See tähendab, et kui ühe päringuga tagastatakse 2000 väljaannet, siis igale väljaandele eraldi juurde pärida MARC andmestik. Antud juhul teeks see kokku 2001 päringut.
MARC andmestiku pärimine JSON kujul	MARC andmestik on võimalik panna kaasa peamisesse päringumeetodisse. See tähendab, et ühe päringuga on võimalik tagastada 2000 väljaannet, mis sisaldavad MARC andmestikku JSON kujul.	Puudub teek, mis liigendaks MARC andmestikku JSON kujul. Lahendus oleks MARC4J.Net teegile kirjutada juurde laiendusmeetodid, mis liigendaks JSON-i õigetesse andmestruktuuridesse.

3.4.1 Testide kirjeldused

Antud probleemi raames koostab autor 2 jõudlustesti. Jõudlustestide eesmärk on võrrelda kahe erineva päringumetoodika aegasid ja tulemeid. Teste ei teostada rangelt piiritletud tingimustes – testid on teostatud autori lokaalses masinas ning tulemused võivad varieeruda. Antud probleem siiski rangelt piiritletud tingimusi ei nõua ning käesoleva testi metoodika on piisav, et jõuda järeldustele. Tagamõte on saada ettekujutus, kui palju võtab üks päringumetoodika rohkem aega kui teine. Tabelis 7 on esitatud testide kirjeldused.

Tabel 7. Testide kirjeldused.

Testi nimetus	**Testi kirjeldus**
Test nr 1	Sierrast päritakse 50 000 väljaande andmestik. Test on jaotatud tööosadeks. Kokku on 25 tööosa ning üks tööosa teostatakse testis ühe tsüklina. Üks tööosa koosneb järgnevatest peamistest funktsioonidest: ▪ Sierrast 2000 väljaande pärimine (1 päring sisaldab 2000 väljaannet koos JSON kujul MARC andmestikuga). ▪ Päringust tagastatud andmete kaardistamine kujule, mis vastab andmebaasi andmestruktuuridele. ▪ Andmete salvestamine andmebaasi.
Test nr 2	Sierrast päritakse 50 000 väljaande andmestik. Test on jaotatud tööosadeks. Kokku on 25 tööosa ning üks tööosa teostatakse testis ühe tsüklina. Üks tööosa koosneb järgnevatest peamistest funktsioonidest: ▪ Sierrast 2000 väljaande pärimine (1 päring sisaldab 2000 väljaannet). ▪ Sierrast 2000 väljaande MARC andmestiku pärimine XML kujul (1 päringuga on võimalik küsida korraga ainult ühe väljaande MARC andmestik, seega tehakse täiendavad 2000 päringut). ▪ Päringust tagastatud andmete kaardistamine kujule, mis vastab andmebaasi andmestruktuuridele. ▪ Andmete salvestamine andmebaasi.

Andmaks aimu keskkonna võimekusest, on tabelis 8 toodud välja teste läbi viinud masina tehnilised näitajad.

Tabel 8. Jõudlustesti sooritanud masina tehnilised näitajad.

Operatsioonisüsteem	Windows 10 Enterprise
Protsessor	Intel Core i7-7700HQ 2.80GHz
Muutmälu	16 GB DDR4
Videokaart	NVIDIA GeForce 940MX 2 GB
Kõvaketas	256 GB SSD, lugemiskiirus 1500 MB/s, kirjutamiskiirus 760 MB/s
Interneti kiirus	100 Mbit/s

3.4.2 Testide tulemused

Järgnevalt on esitatud testi nr 1 tulemused. Joonisel 9 on näha testi nr 1 väljund. Tabelis 9 on esitatud väljundi põhjal testi nr 1 tulemused.

```
Performance test completed.
MARC data saved in JSON
Total elapsed time: 00:03:53.589
Total requests made to Sierra API: 25
Total publications fetched from Sierra API: 50000
Total publications saved to database: 50000
Total amount of tasks: 25
Max task execution time: 15.75 s
Min task execution time: 8.18 s
Average mean of tasks: 9.34 s
Geometric mean of tasks: 9.25 s
Median of tasks: 9.09 s
```

Joonis 9. Testi nr 1 väljund.

Tabel 9. Testi nr 1 tulemused.

Kirjeldus	**Tulemus**
Testi kulunud aeg kokku	3 minutit ja 53 sekundit
Sierrale saadetud päringute arv kokku	25
Sierrast saadud väljaannete arv kokku	50 000
Andmebaasi salvestatud väljaannete arv kokku	50 000
Tööosade arv kokku	25
Suurim tööosale kulunud aeg	15,75 sekundit
Väikseim tööosale kulunud aeg	8,18 sekundit
Tööosadele kulunud aegade aritmeetiline keskmine	9,34 sekundit
Tööosadele kulunud aegade geomeetriline keskmine	9,25 sekundit
Tööosadele kulunud aegade mediaan	9,09 sekundit

Järgnevalt on esitatud testi nr 2 tulemused. Joonisel 10 on näha testi nr 2 väljund. Tabelis 10 on esitatud väljundi põhjal testi nr 2 tulemused.

```
Performance test completed.
MARC data saved in XML
Total elapsed time: 00:17:05.261
Total requests made to Sierra API: 50025
Total publications fetched from Sierra API: 50000
Total publications saved to database: 50000
Total amount of tasks: 25
Max task execution time: 45.75 s
Min task execution time: 36.31 s
Average mean of tasks: 41.00 s
Geometric mean of tasks: 40.94 s
Median of tasks: 41.16 s
```

Joonis 10. Testi nr 2 väljund.

Tabel 10. Testi nr 2 tulemused.

Kirjeldus	**Tulemus**
Testi kulunud aeg kokku	17 minutit ja 5 sekundit
Sierrale saadetud päringute arv kokku	50 025
Sierrast saadud väljaannete arv kokku	50 000
Andmebaasi salvestatud väljaannete arv kokku	50 000
Tööosade arv kokku	25
Suurim tööosale kulunud aeg	45,75 sekundit
Väikseim tööosale kulunud aeg	36,31 sekundit
Tööosadele kulunud aegade aritmeetiline keskmine	41,00 sekundit
Tööosadele kulunud aegade geomeetriline keskmine	40,94 sekundit
Tööosadele kulunud aegade mediaan	41,16 sekundit

Tulemustest selgub, et JSON-i põhine andmete pärimine on ~4 korda kiirem. Autori hinnangul kaalub JSON-i põhise päringumetoodika väiksem päringute arv ja andmestiku ~4 korda kiirem pärimine üle selle, et MARC andmestiku lugemiseks ja liigendamiseks on vaja kirjutada laiendusmeetodeid. Autor kasutab edasises töös JSON-i põhist päringumetoodikat. Testide lähtekood on esitatud Lisa 3 all. MARC4J.Net teegi jaoks loodud JSON-i liigendamise laiendusklass on esitatud Lisa 4 all.

3.5 Kliendirakendus

Kliendirakenduse arendamiseks kasutatakse TypeScript programmeerimiskeelt ja React raamistikku. Uue React rakenduse loomiseks kasutab autor Node paketihaldurit (Joonis 11).

```
npx create-react-app RLSearchClient
```

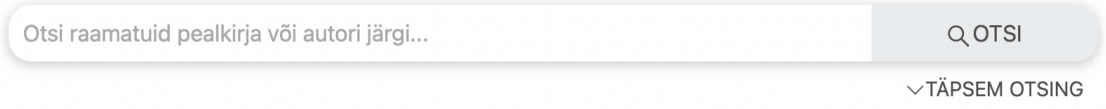
Joonis 11. Uue React rakenduse loomise käsk.

Antud töö kliendirakenduse loomisel on põhiline fookus otsingu funktsionaalsetel komponentidel. Stiili- ja disainielemendid lõpptootesse suure tõenäosusega ei jõua – projekti RL stiiliraamat luuakse alles projekti järgmistes etappides. Käesolev kliendirakendus koosneb põhiotsingust ja täiendavatest otsinguväljadest. Otsingu tulemusena kuvatakse leitud vasteid.

Kliendirakendus võimaldab otsida väljaandeid järgnevate otsinguväljadega:

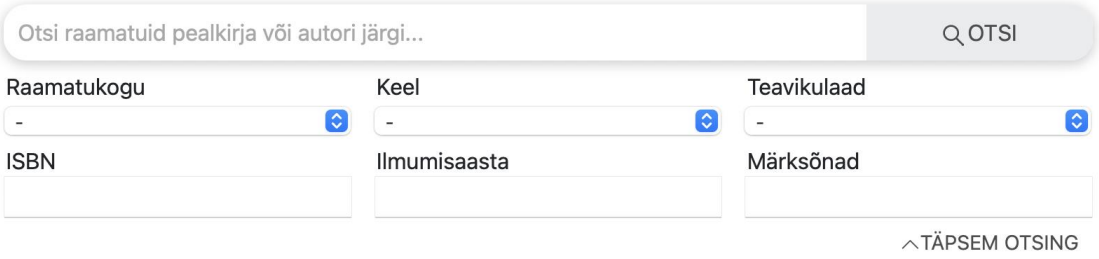
- põhiotsing - võimaldab otsida väljaannet pealkirja ja autori järgi. Põhiotsing toetab ka täpsemat otsingut täpsustussõnade abil;
- raamatukogud;
- ISBN/ISSN/ISMN;
- ilmumisaasta;
- väljaande keeled;
- märksõnad - teose märksõnaks võib olla näiteks kirjanduse liik (nt ilukirjandus) või žanr (nt sõjaromaan);
- teaviku laad.

Järgneval joonisel 12 on kuvatud kliendirakenduse põhiotsing.



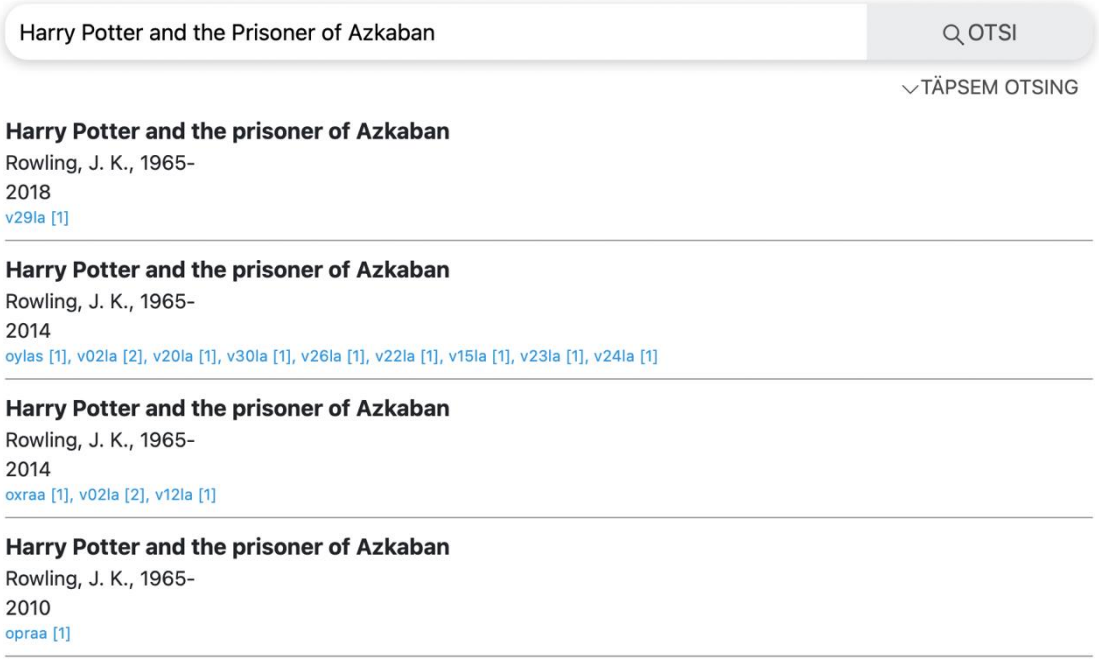
Joonis 12. Kliendirakenduse põhiotsing.

Järgneval joonisel 13 on kuvatud põhiotsing koos täiendavate otsinguväljadega.



Joonis 13. Kliendirakenduse otsing koos täiendavate otsinguväljadega.

Järgneval joonisel 14 on näha, kuidas kuvatakse kliendirakenduses otsinguvasteid.



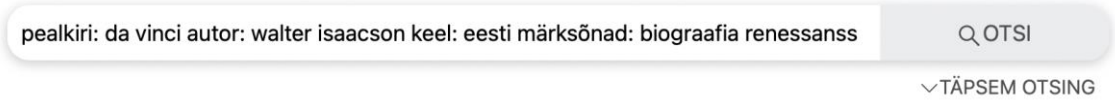
Joonis 14. Kliendirakenduse otsinguvastete kuvamine.

Eelnevalt sai mainitud, et põhiotsing toetab ka täpsemat otsingut. Täpsemaks otsinguks tuleb põhiotsingus määrata otsingutermini ette täpsustussõna. Täpsemat otsingut saab teostada järgnevate täpsustussõnadega:

- pealkiri: *otsingusõne*;
- autor: *otsingusõne*;
- isbn: *otsingusõne*;
- issn: *otsingusõne*;
- ismn: *otsingusõne*;
- aasta: *otsingusõne*;

- raamatukogu: *otsingusõne*;
- keel: *otsingusõne*;
- laad: *otsingusõne*;
- märksõnad: *otsingusõne*.

Järgneval joonisel 15 on toodud näide, millisel kujul saab põhiotsingus kasutada täpsustussõnu.



Joonis 15. Kliendirakenduse põhiotsingu näide koos täpsustussõnadega.

4 Otsingutehnoloogia valimine

Traditsioonilised andmebaasid ei ole üldjuhul optimaalsed täisteksti otsingute teostamiseks. See tähendab, et traditsiooniliste andmebaasidega ei ole tekstipõhised otsingud kiired, eelkõige juhtudel, kus andmemahud muutuvad suureks. Lisaks napib traditsiooniliste andmebaasidega ka teisi kasulikke otsingu funktsioone, näiteks puuduvad nendes üldjuhul keelte analüsaatorid, hägusotsingu tugi või relevantsuse algoritmi seadistamise võimalused.

Selle osa eesmärk on uurida ja analüüsida erinevaid otsingutehnoloogiaid, tagamõttega muuta rakenduse otsing kiireks ja mugavaks lõppkasutaja jaoks. Nõuded otsingutehnoloogiatele on järgnevad: .NET-i ühilduvus, hägusotsingu tugi, kiire tekstipõhine otsimine ja sobivus suurte andmemahtude jaoks. Lähtudes nendest nõuetest, on autor vaatlemiseks valinud 4 erinevat otsingutehnoloogiat – Lucene, Apache Solr, Elasticsearch ja Azure Cognitive Search. Autor vaatleb otsingutehnoloogiaid kronoloogilises järjekorras, alustades vanimast. Alustatakse Lucene’st, mis on vaadeldavatest kõige madalatasemelisem ning tuumaks ka kõikidele järgnevatele tehnoloogiatele. Järgmisena vaadeldakse Apache Solri, seejärel Elasticsearchi ning viimasena Azure Cognitive Searchi.

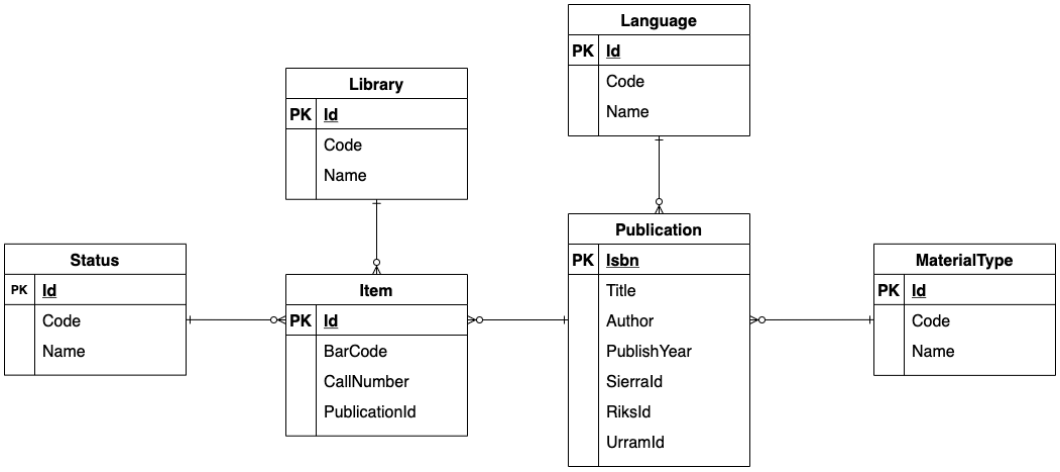
Iga vaadeldava otsingutehnoloogia kohta on loodud lühike ülevaade. Tehnoloogiaga lähemalt tutvumiseks on koostatud ka näidislahendus. Peale näidislahenduse teostamist on toodud välja tehnoloogia plussid ja miinused. Põhjalikum analüüs ja valiku langetamine toimub peatükis 4.3 Otsingutehnoloogiate analüüs.

4.1 Otsingutehnoloogial baseeruva näidislahenduse kirjeldus

Näidislahendus koosneb otsingutehnoloogia ülesseadmisest, kirjete indekseerimisest ning kirjete otsingutest. Lisaks on loodud näidislahenduse jaoks andmemudel.

4.1.1 Näidislahenduse andmemudel

Näidislahenduse andmemudeli alusel koostatakse vajalikud andmestruktuurid. Indekseerimisel normaliseeritakse andmestruktuur siiski üheks pikaks dokumendiks, et muuta otsing võimalikult kiireks. Joonisel 16 on esitatud näidislahenduse ERD.



Joonis 16. Näidislahenduse ERD.

4.1.2 Ülesseadmine

Peale tehnoloogia ülesseadmist peab olema valmidus indeksite loomiseks, kirjete indekseerimiseks ning kirjete pärimiseks.

4.1.3 Kirjete indekseerimine

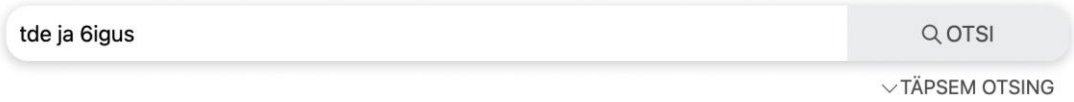
Kirjete indekseerimiseks luuakse rakenduses vajalikud meetodid. Näidislahendustes indekseeritakse ~57000 raamatut.

4.1.4 Kirjete otsing

Kirjete otsingu raames luuakse valmis otsingu päringumeetod ning teostatakse neli erinevat otsingut. Otsingute teostamiseks koostatakse päring domeenispetsiifilises keeles. Otsing peab toetama ka hägusotsingut.

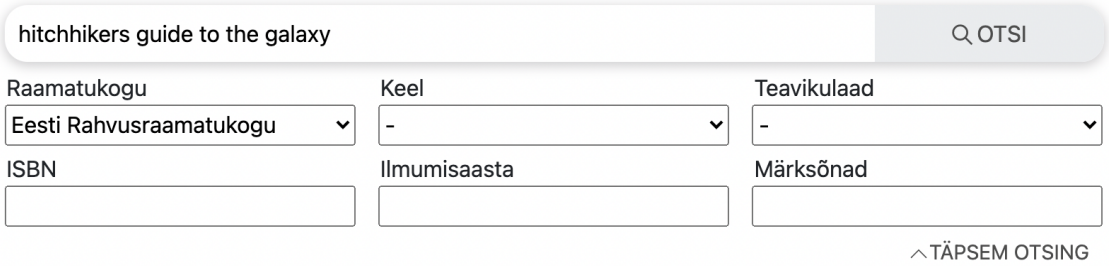
Näidetes otsitakse nelja erinevat raamatut. Kaks esimest otsingut on vigaste otsingusõnadega, kontrollides hägusotsingu funktsiooni. Kaks viimast otsingut

kontrollivad otsingusüsteemi toimetulekut keerulisemate andmestruktuuridega, otsides vasteid andmestruktuuri sisemistest objektidest. Järgnevatel joonistel on esitatud otsingu näited:

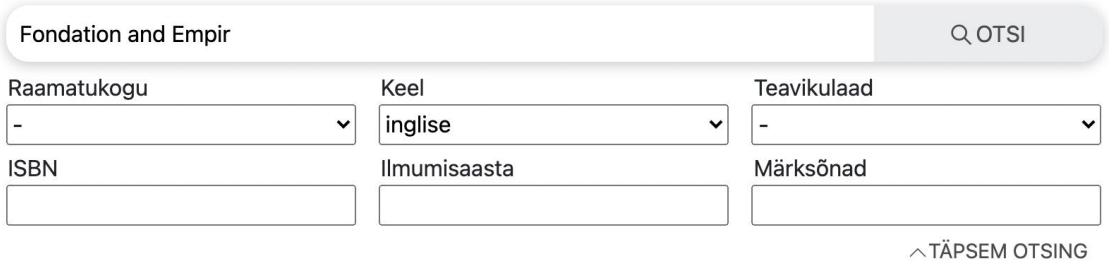


Joonis 17. Näidislahenduse otsingu näide 1: otsitav teos A.H. Tammsaare „Tõde ja õigus“.

Joonis 18. Näidislahenduse otsingu näide 2: otsitav teos J.K Rowling „Harry Potter ja surma vägised“.



Joonis 19. Näidislahenduse otsingu näide 3: otsitav teos Douglas Adams „The hitchhiker's guide to the galaxy“.



Joonis 20. Näidislahenduse otsingu näide 4: otsitav teos Isaac Asimov „Foundation and Empire“.

4.2 Otsingutehnoloogiad

Käesolevas peatükis vaadeldakse lähemalt nelja valitud otsingutehnoloogiat. Iga tehnoloogia kohta on esitatud lühikirjeldus, koostatud näidislahendus ning toodud välja plussid ja miinused.

4.2.1 Lucene

Lucene on vabavaraline otsingumootori teek, mis on originaalselt kirjutatud Javas ning omab Apache 2.0 litsentsi. Lucene ei ole kõrgetasemeline valmislahendus – see on teek, mis võimaldab abistavate meetoditega lisada rakendusele lihtsal moel otsinguvõimalusi. Selle põhilisteks funktsioonideks on teksti indekseerimine ja otsimine ning see teeb seda väga efektiivselt [30]. See tähendab, et kasutaja ei pea muretsema indekseerimise ja otsingu keerulise loogika pärast – sellega tegeleb Lucene. Kasutaja vastutuseks jääb Lucene'i konfigureerimine vastavalt vajadustele. Lisaks Javale on Lucene'i teek ümberkirjutusena olemas ka paljudes teistes programmeerimiskeeltes, sealhulgas: Perl, Python, Ruby, C, C++, PHP ja C# [31]. Mõned kõrgetasemelisemad lahendused, näiteks Elasticsearch, Apache Solr ja Azure Cognitive Search põhinevad just Lucene'l. Lucene'i otsingumootor on kasutusel näiteks järgnevates populaarsetes ettevõtetes ja tarkvarades: LinkedIn, Twitter, IBM, Eclipse, Jira ja Apple [32]. Lucene'i näidislahendus on toodud välja Lisa 5 all.

Lucene'i plussid:

- tasuta;
- vabavaraline litsents;
- avatud lähtekood;
- väga hea jõudlus;
- väga paindlik, sest see on madalatasemeline;
- sisaldab eesti keele analüsaatorit;
- ümberkirjutatud teek on olemas paljudes erinevates programmeerimiskeeltes.

Lucene'i miinused:

- nõrk kasutajatugi;
- puudulik dokumentatsioon;
- nõuab palju seadistamist, sest see on madalatasemeline;
- keerulisemate andmestruktuuride seadistamine on kohmakas;
- käesolev projekt on ülesehituselt distributiivne ning on arvestatud sellega, et ka otsingusüsteem paikneb eraldiseisva teenusena. See tähendab, et Lucene'i rakendamiseks antud projektis tuleks ehitada veel üks REST veebirakendus;

- ümberkirjutud teegid teistes programmeerimiskeeltes on vanemad versioonid. Lucene.NET on uuendustega mitu aastat maas - viimane Lucene.NET'i versioon on 4.8.0 beeta, samas kui originaalne Apache Lucene on juba versioon 8.8.1. Apache Lucene versioon 4.8.0 tuli välja juba aastal 2014 [33].

4.2.2 Apache Solr

Apache Solr on vabavaraline otsingumootor mille tuumikuks on Lucene. Apache Solr on kirjutatud Javas ning see jookseb valmislahendusena eraldiseisva REST veebiserveri peal. Apache Solr veebiserveriga saab suhelda HTTP vahendusel, kasutades JSON-it, XML-i, CSV-d, binaarkoodi või teisi formaate [34]. Apache Solr veebiserveriga suhtlemiseks on kliendi teek olemas järgnevates programmeerimiskeeltes: Java, Python, Ruby, PHP, .NET (C#), Scala, Perl, JavaScript, Clojure, Go, Rust, R, C++, Lua [35]. Apache Solr sisaldab ka SolrCloudi millega on võimalik muuta rakendus distributiivseks. SolrCloudiga saab üles seada serverite klastri, tagades sellega parema jõudluse suurte andmemahtude korral, parema veataluvuse ning parema rakenduse kättesaadavuse [36]. Apache Solri näidislahendus on toodud välja Lisa 6 all.

Apache Solri plussid:

- tasuta;
- vabavaraline litsents;
- avatud lähtekood;
- hea jõudlus;
- distributiivne;
- võimaldab indekseerida paljusid erinevaid formaate, sh JSON, XML, CSV, HTML, PDF, Microsoft Office toodete formaadid jt;
- sisaldab graafilist liidest süsteemi haldamiseks ja seadistamiseks;
- sisaldab eesti keele analüsaatorit (Lucene'i kaudu).

Apache Solri miinused:

- mõningate funktsioonide saavutamine nõuab palju seadistamist;
- indeksi seadistamine on kohmakas;
- keerulisemate andmestruktuuride seadistamine on kohmakas;
- REST API ei toeta mõningaid funktsioone.

4.2.3 Elasticsearch

Elasticsearch on avaliku lähtekoodiga tasuta distributiivne otsingumootor, mis on ehitatud Lucene'i peale [37]. Elasticsearch sisaldab valmislahendusena REST veebiserverit, kus toimub andmete indekseerimine ning päringute töötlemine. Infovahetus veebiserveriga toimub HTTP vahendusel, andmevahetusvorminguks on JSON. Elasticsearchi arendab põhiliselt ettevõte Elastic, mis pakub Elasticsearchi rakendamiseks ka erinevaid pilveteenuseid. Elasticsearch omas pikalt vabavaralist Apache 2.0 litsentsi, aga 2021 jaanuaris see muutus ning nüüd on Elasticsearchil duaalne litsents – Elastic License 2.0 ja SSPL 1.0 [38]. Duaalne litsents tähendab, et kasutajal võimalus valida kumba litsentsi ta kasutab. Elastic License 2.0 ega SSPL 1.0 ei ole kumbki tunnistatud OSI poolt vabavaralisteks litsentsideks [39], [40]. Kliendi teek Elasticsearchi veebiserveriga suhtlemiseks on olemas järgmistel programmeerimiskeeltel: Java, JavaScript, Go, .NET (C#), PHP, Perl, Python, Ruby [37]. .NET'i jaoks on Elasticsearchil 2 kliendi teeki [41]:

1. Elasticsearch.Net – madalatasemeline teek. Elasticsearch.Net'is ei ole seatud rangeid piire millisel kujul kasutaja peab päringuid koostama ning ei oma tähtsust mis kasutaja vastu võetud andmetega teeb.
2. NEST – kõrgetasemeline teek. Selles teegis kasutatakse ära .NET'i mugavaid funktsioone. NEST'iga saab päringuid koostada tugevalt tüübitud domeenispetsiifilises keeles. Sisemiselt põhineb NEST eelmisel madalatasemelisel teegil. Vajadusel on võimalik NEST'iga kasutada ka kõiki madalatasemelise teegi meetodeid.

Elasticsearchi näidislahendus on esitatud Lisa 7 all.

Elasticsearchi plussid:

- tasuta;
- avatud lähtekood;
- hea kasutajatugi;
- hea dokumentatsioon;
- väga hea jõudlus;
- distributiivne;
- palju sisse ehitatud lisafunktsioone;

- sisaldab eesti keele analüsaatorit (Lucene'i kaudu);
- üsna paindlik;
- olemas eraldi tooted muudeks lisafunktsioonideks. Näiteks Kibana andmete visualiseerimiseks ja APM jõudluse seireks;
- mugav .NET'i kliendi teek;
- lihtne ja intuitiivne kasutada.

Elasticsearchi miinused:

- ei oma vabavaralist litsentsi;
- toetab ainult JSON andmevahetusvormingut.

4.2.4 Azure Cognitive Search

Azure Cognitive Search on pilvepõhine otsinguteenus, mis pakub arendajatele API-sid ja tööriistu otsingusüsteemide loomiseks [42]. Azure Cognitive Searchi tuumikuks on samuti Lucene. Arhitektuuriliselt on tegu REST veebiteenusega, mis klassikaliselt paigutatakse andmehoidla ja kliendirakenduse vahele. Azure Cognitive Search veebiteenusega suhtlemiseks on Microsofti poolt kliendi teek olemas järgnevatel programmeerimiskeeltel: .NET (C#), Java, JavaScript ja Python. Azure'i otsinguteenus sisaldab järgnevaid komponente [42]:

- otsingumootor täisteksti otsinguks;
- andmehoidla kasutaja indekseeritud sisu jaoks;
- API-d indekseerimiseks ja andmete pärimiseks;
- tehisintellektil põhinevad lisafunktsioonid, mis võimaldavad luua otsitavat sisu piltidest, struktureerimata tekstidest ja rakenduse failidest;
- integratsioon teiste Azure'i teenustega;
- semantiline otsing.

Azure Cognitive Searchi näidislahendus on esitatud Lisa 8 all.

Azure Cognitive Searchi plussid:

- ühilduvus teiste Azure'i teenustega;
- lihtne ja intuitiivne kasutada;
- hea kasutajatugi;

- hea dokumentatsioon;
- väga hea jõudlus;
- distributiivne;
- hea .NET'i kliendi teek;
- sisaldab eesti keele analüsaatorit (Lucene'i kaudu).

Azure Cognitive Searchi miinused:

- teenus on saadaval ainult Azure'is;
- kõrge hind;
- Lucene'i domeenispetsiifiline keel ei paista korrektselt toimivat.

4.3 Otsingutehnoloogiate analüüs

Käesolevas peatükis kasutatud analüüsi meetodid ja valemid on kirjeldatud ülalpool, 2.2 Mitme-kriteeriumi analüüs, lk 16-20.

4.3.1 Eesmärk, kriteeriumid ja alternatiivid

Analüüsi eesmärk on otsingutehnoloogia valimine. Valitud otsingutehnoloogiat planeeritakse kasutada projekti Raamatud Liikuma otsingusüsteemi loomiseks.

Kriteeriumid otsingutehnoloogia valimisel on järgnevad:

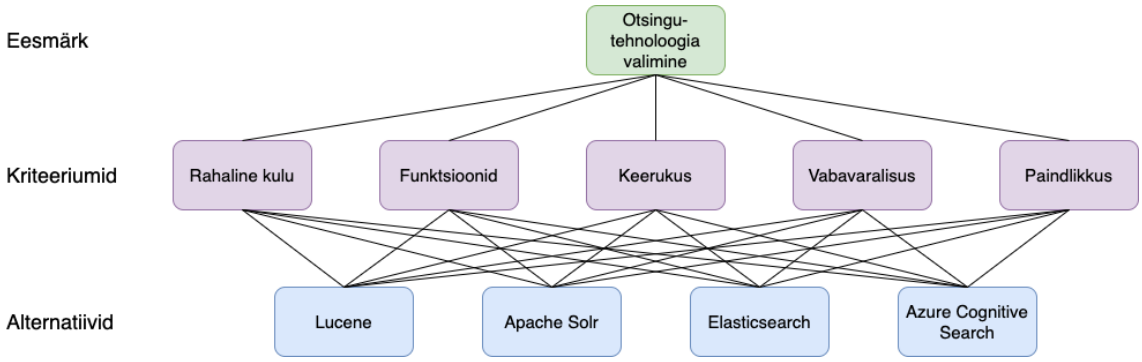
- rahaline kulu – arvestab tehnoloogiate maksumust ja ülalpidamiskulusid;
- funktsioonid – arvestab eelkõige tehnoloogia poolt pakutavaid valmis-funktsioone, mis katavad ärivajadusi;
- keerukus – arvestab tehnoloogia ülesseadmise ja funktsioonide rakendamise keerukust ning dokumentatsiooni ülesehitust ja loetavust;
- vabavaralisus – ei määra selget piiri sellega, kas tarkvara on definitsiooni kohaselt vabavaraline või mitte. See kriteerium sisaldab endas kahte võrdse kaaluga alamkriteeriumit: vabavaralise litsentsi olemasolu ning avatud lähtekoodi olemasolu. See kriteerium on sellepärast, et võtta arvesse mitte-vabavaraliste litsentside piiranguid ning toetada vabavaralist tarkvara ja teadmiste levikut. Kuigi kriteeriumi kaal on väike, on selle eesmärk saada määravaks, kui peaks analüüsi tulemusena tekkima kaks võrdse tulemusega valikut, millest üks on vabavaraline ja teine mitte;

- paindlikkus – arvestab eelkõige tarkvara paindlikkuse definitsiooniga ning majutusvõimalustega.

Alternatiividekseks on valitud 4 otsingutehnoloogiat:

- Lucene;
- Apache Solr;
- Elasticsearch;
- Azure Cognitive Search.

Järgneval joonisel 21 on probleem esitatud hierarhiana.



Joonis 21. Otsingutehnoloogiate hierarhia.

4.3.2 Kriteeriumite kaalude leidmine AHP meetodiga

Kriteeriumite võrdlusmaatriksi saamiseks koostatakse kriteeriumite paarikaupa võrdlemine Saaty skaala alusel (vt Kriteeriumite kaalude leidmine, lk 18). Järgnevas tabelis 11 on esitatud paarikaupa võrdlemise tulemusena saadud võrdlusmaatriks.

Tabel 11. Kriteeriumite võrdlusmaatriks.

	Kulu hind	**Funktsioonid**	**Keerukus**	**Vabavaralisus**	**Paindlikkus**
Kulu hind	1	2	4	6	2
Funktsioonid	$\frac{1}{2}$	1	3	5	2
Keerukus	$\frac{1}{4}$	$\frac{1}{3}$	1	3	$\frac{1}{2}$
Vabavaralisus	$\frac{1}{6}$	$\frac{1}{5}$	$\frac{1}{3}$	1	$\frac{1}{4}$
Paindlikkus	$\frac{1}{2}$	$\frac{1}{2}$	2	4	1
Summa	2,42	4,03	10,33	19	5,75

Normaliseeritud võrdlusmaatriksi saamiseks jagatakse iga maatriksi element vastava veeru summaga. Kriteeriumi kaal saadakse, kui võetakse normaliseeritud võrdlusmaatriksis kriteeriumi rea keskmine väärtus. Järgnevas tabelis 12 on esitatud normaliseeritud võrdlusmaatriks.

Tabel 12. Normaliseeritud kriteeriumite võrdlusmaatriks.

	Kulu hind	**Funktsioonid**	**Keerukus**	**Vabavaralisus**	**Paindlikkus**	**Kaal**
Kulu hind	0,41	0,50	0,39	0,32	0,35	0,39
Funktsioonid	0,21	0,25	0,29	0,26	0,35	0,27
Keerukus	0,10	0,08	0,10	0,16	0,09	0,11
Vabavaralisus	0,07	0,05	0,03	0,05	0,04	0,05
Paindlikkus	0,21	0,12	0,19	0,21	0,17	0,18

4.3.3 Kriteeriumite alternatiivide väärtused

Kriteeriumite alternatiivide väärtuste määramiseks kasutatakse Hwangi ja Yooni poolt pakutud skaalat (vt Tabel 3, lk 19).

Rahaline kulu

Peamised faktorid rahalise kulu osas on majutuse valikuvõimalused ning tehnoloogia mälukasutus ja sellest tingitud kettaruumi vajadused.

Järgnevas tabelis 13 on esitatud otsingutehnoloogiate indeksi mälukasutus, kus iga indeks sisaldab 57097 dokumenti.

Tabel 13. Indeksite suurused näidislahendustes.

Alternatiiv	**Indeksi suurus**
Lucene	26,31 MB
Apache Solr	31,12 MB
Elasticsearch	41,4 MB
Azure Cognitive Search	30,91 MB

Olgu B indeksi suurus ja n dokumentide arv indeksis, siis otsingutehnoloogia hüpoteetiline vajaminev mälumaht X on:

$$X = \frac{B}{n} \times 1{,}5 \times 7\,000\,000 \times 2 \tag{4.1}$$

Võrrandis 4.1 jagatakse indeksi suurus dokumentide arvuga, et saada keskmine mälukasutus ühe dokumendi kohta. Tulemus korrutatakse 1,5-ga, sest näidislahenduses kasutatav andmemudel on lihtsustatud. Seejärel korrutatakse saadud arv 7 miljoniga, mis on ligikaudne bibliokirjete arv. Lõpuks korrutatakse saadud tulemus 2-ga, et arvestada ka teiste asjaoludega, mis nõuavad mälu ning jätta vaba ruumi ka tulevikuks. Tulemuseks on hüpoteetiline vajaminev mälumaht 7 miljoni bibliokirje indekseerimiseks. Järgnevas tabelis 14 on esitatud iga alternatiivi hüpoteetiline vajaminev mälumaht.

Tabel 14. Alternatiivide hüpoteetiline vajaminev mälumaht.

Alternatiiv	**Vajaminev mälumaht**
Lucene	9,68 GB
Apache Solr	11,45 GB
Elasticsearch	15,23 GB
Azure Cognitive Search	11,37 GB

Järgnevalt vaadeldakse otsingutehnoloogiate majutusvõimalusi. Valikutes on lähtutud teenusepakkuja tootevalikust, toote näitajatest ja sellest, et katta minimaalsete kuludega vajalikud nõuded. Kättesaadavuse parandamiseks ja varukoopiate hoidmiseks vaadatakse majutust korraga kahe erineva teenuspakkuja juures. Erandiks on Azure Cognitive Search, mille majutus on piiratud Azure'ga. Azure Cognitive Search'i kättesaadavuse parandamiseks on lisatud 1 koopia, mis kahekordistab hinna. Tabelis 15 on esitatud otsingutehnoloogiate majutusvõimalused.

Tabel 15. Otsingutehnoloogiate majutusvõimalused.

Valik	**Majutus 1**	**Majutus 2**	**Hind kokku**
Lucene	Contabo, VPS S SSD (4 tuuma, 8 GB RAM, 200 GB SSD), hind: 4,99 €	Netcup, VPS 1000 G8 (2 tuuma, 8 GB RAM, 160 GB SSD), hind: 6,00 €	10,99 €

Valik	Majutus 1	Majutus 2	Hind kokku
Apache Solr	Contabo, VPS S SSD (4 tuuma, 8 GB RAM, 200 GB SSD), hind: 4,99 €	Netcup, VPS 1000 G8 (2 tuuma, 8 GB RAM, 160 GB SSD), hind: 6,00 €	10,99 €
Elasticsearch	Contabo, VPS S SSD (4 tuuma, 8 GB RAM, 200 GB SSD), hind: 4,99 €	Netcup, VPS 1000 G8 (2 tuuma, 8 GB RAM, 160 GB SSD), hind: 6,00 €	10,99 €
Azure Cognitive Search	Azure, Standard S1 (25 GB kettaruumi), 2 koopiat, hind: 2 × 206,845 €	-	413,69 €

Järgnevas tabelis 16 on esitatud rahalise kulu kriteeriumi skaala definitsioonid, kus X tähistab alternatiivi rahalist kulu kuu lõikes.

Tabel 16. Rahalise kulu kriteeriumi skaala definitsioonid

Definitsioon	Väärtus
X < 15 €	1
X > 300 €	9

Rahalise kulu puhul on tegemist negatiivse kriteeriumiga, kus väiksem väärtus annab parema tulemuse. Järgnevas tabelis 17 on esitatud rahalise kulu kriteeriumi alternatiivide väärtused.

Tabel 17. Rahalise kulu kriteeriumi alternatiivide väärtused

Alternatiiv	Väärtus
Lucene	1
Apache Solr	1
Elasticsearch	1
Azure Cognitive Search	9

Funktsioonid

Funktsioonide kriteeriumi puhul on tegemist positiivse kriteeriumiga, kus suurem väärtus annab parema tulemuse. Järgnevas tabelis 18 on esitatud funktsioonide kriteeriumi alternatiivide väärtused.

Tabel 18. Funktsioonide kriteeriumi alternatiivide väärtused

Valik	**Väärtus**
Lucene	3
Apache Solr	5
Elasticsearch	7
Azure Cognitive Search	7

Keerukus

Keerukuse kriteeriumi puhul on tegemist negatiivse kriteeriumiga, kus väiksem väärtus annab parema tulemuse. Järgnevas tabelis 19 on esitatud keerukuse kriteeriumi alternatiivide väärtused.

Tabel 19. Keerukuse kriteeriumi alternatiivide väärtused

Valik	**Väärtus**
Lucene	5
Apache Solr	5
Elasticsearch	2
Azure Cognitive Search	4

Vabavaralisus

Järgnevas tabelis 20 on esitatud vabavaralisuse kriteeriumi skaala definitsioonid.

Tabel 20. Vabavaralisuse kriteeriumi skaala definitsioonid

Definitsioon	**Väärtus**
Ei ole avatud lähtekoodiga ning puudub vabavaraline litsents	1
Avatud lähtekood ning puudub vabavaraline litsents	5
Avatud lähtekood ning omab vabavaralist litsentsi	9

Järgnevas tabelis 21 on esitatud alternatiivide vabavaralised omadused.

Tabel 21. Alternatiivid ja vabavara

	Vabavaraline litsents	**Avatud lähtekood**
Lucene	+	+
Apache Solr	+	+
Elasticsearch	-	+
Azure Cognitive Search	-	-

Vabavaralisuse kriteeriumi puhul on tegemist positiivse kriteeriumiga, kus suurem väärtus annab parema tulemuse. Järgnevas tabelis 22 on esitatud vabavaralisuse kriteeriumi alternatiivide väärtused.

Tabel 22. Vabavaralisuse kriteeriumi alternatiivide väärtused

Valik	**Väärtus**
Lucene	9
Apache Solr	9
Elasticsearch	5
Azure Cognitive Search	1

Paindlikkus

Paindlikkuse kriteeriumi puhul on tegemist positiivse kriteeriumiga, kus suurem väärtus annab parema tulemuse. Järgnevas tabelis 23 on esitatud paindlikkuse kriteeriumi alternatiivide väärtused.

Tabel 23. Paindlikkuse kriteeriumi alternatiivide väärtused

Valik	**Väärtus**
Lucene	8
Apache Solr	6
Elasticsearch	7
Azure Cognitive Search	1

4.3.4 Otsustusmaatriks ja alternatiivide tulemuste leidmine WPM meetodiga

Kriteeriumite alternatiivide väärtuste alusel on koostatud normaliseerimata otsustusmaatriks (Tabel 24). Kriteeriumite juures on esitatud ka kriteeriumi tüüp (positiivne või negatiive) ning kaal.

Tabel 24. Normaliseerimata otsustusmaatriks

	Rahaline kulu (-) (0,39)	**Funktsioonid (+) (0,27)**	**Keerukus (-) (0,11)**	**Vabavaralisus (+) (0,05)**	**Paindlikkus (-) (0,18)**
Lucene (A_1)	1	3	5	9	8
Apache Solr (A_2)	1	5	5	9	6
Elasticsearch (A_3)	1	7	2	5	7
Azure Cognitive Search (A_4)	9	7	4	1	1

Otsustusmaatriksi normaliseerimiseks on kasutatud lineaarset transformatsiooni (vt Kriteeriumite alternatiivide väärtuste leidmine ja normaliseerimine, lk 19). Järgnevas tabelis 25 on esitatud normaliseeritud otsustusmaatriks.

Tabel 25. Normaliseeritud otsustusmaatriks

	Rahaline kulu (0,39)	**Funktsioonid (0,27)**	**Keerukus (0,11)**	**Vabavaralisus (0,05)**	**Paindlikkus (0,18)**
Lucene (A_1)	1	0,43	0,4	1	1
Apache Solr (A_2)	1	0,71	0,4	1	0,75
Elasticsearch (A_3)	1	1	1	0,56	0,875
Azure Cognitive Search (A_4)	0,11	1	0,5	0,11	0,125

Järgnevalt on arvutatud alternatiivide tulemused WPM meetodiga (vt WPM, lk 17):

$$A_1 = 1^{0,39} \times 0{,}43^{0,27} \times 0{,}4^{0,11} \times 1^{0,05} \times 1^{0,18} = 0{,}72$$

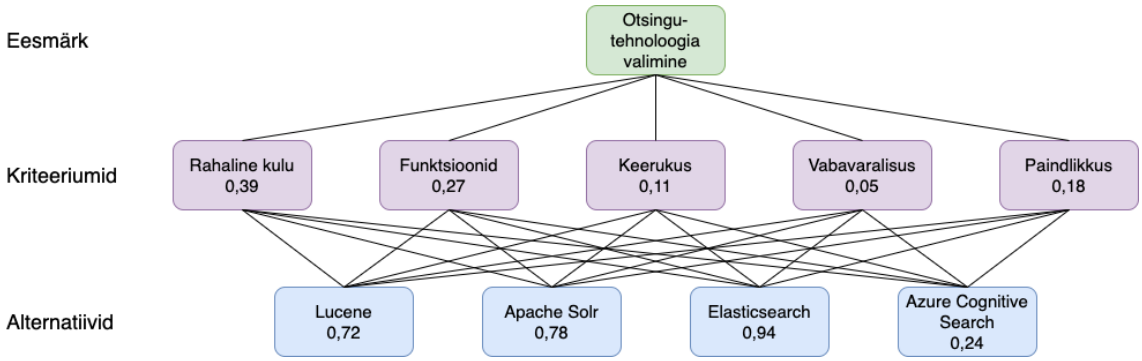
$$A_2 = 1^{0,39} \times 0{,}71^{0,27} \times 0{,}4^{0,11} \times 1^{0,05} \times 0{,}75^{0,18} = 0{,}78$$

$$A_3 = 1^{0,39} \times 1^{0,27} \times 1^{0,11} \times 0{,}56^{0,05} \times 0{,}875^{0,18} = 0{,}94$$

$$A_4 = 0{,}11^{0,39} \times 1^{0,27} \times 0{,}5^{0,11} \times 0{,}11^{0,05} \times 0{,}125^{0,18} = 0{,}24$$

4.3.5 Järeldused

Analüüsi tulemusena selgus, et parimaks alternatiiviks osutus Elasticsearch. Teiseks osutus Apache Solr, kolmandaks Lucene ning neljandaks Azure Cognitive Search. Järgneval joonisel 22 on esitatud analüüsi tulemused hierarhilisel kujul.



Joonis 22. Otsingutehnoloogiate hierarhia koos kaalude ja tulemustega.

Kokkuvõtteks võib öelda, et Elasticsearch ei erine kulu poolest Lucene'st ja Apache Solrist, aga võrreldes viimastega on Elasticsearch funktsioonide poolest rikkam ning selle keerukus on autori hinnangul tunduvalt madalam. Apache Solr on kõigist neljast autori hinnangul kõige kohmakam, kuid sellegipoolest, võrreldes Lucene'ga, katab Apache Solr valmislahendusena suurema osa vajadustest ning sobib paremini teenusepõhise arhitektuuriga. Azure Cognitive Search on küll funktsioonide poolest rikas ja üsna madala keerukusega, kuid sellele said saatuslikuks suur rahaline kulu ja paindumatus. Elasticsearchi kasutatakse töö edasises osas prototüübi loomiseks.

5 Prototüüp

Käesolevas peatükis kasutatakse serveripoolset rakendust, kliendirakendust ning analüüsi tulemusena valitud otsingutehnoloogiat otsingusüsteemi prototüübi loomiseks. Välja tuuakse rakenduse olulisemad kohad ning esitatud on olulisemad osad programmikoodist koos selgitustega.

5.1 Serveripoolse rakenduse liidestamine otsingusüsteemiga

Suhtlus Elasticsearch otsingusüsteemiga toimub serveripoolse rakenduse vahendusel. Serveripoolse rakenduse vastutus on siinkohal saata otsingusüsteemile indekseerimiseks korrastatud kujul väljaandeid ning vahendada kliendirakendusest teostatud otsinguid.

Otsingusüsteemiga mugavaks suhtlemiseks on serveripoolses rakenduses võetud kasutusele Elasticsearchi NEST klienditeek [43]. Peale teegi installeerimist registreeritakse ElasticClient instants DI konteinerisse (Joonis 23). ElasticClient pakub abistavaid meetodeid otsingusüsteemiga suhtlemiseks HTTP vahendusel.

```
var settings = new ConnectionSettings(new Uri("http://localhost:9200"));
var elasticClient = new ElasticClient(settings);
services.AddSingleton<IElasticClient>(elasticClient);
```

Joonis 23. ElasticClient instantsi registreerimine DI konteinerisse.

Serveripoolses rakenduses luuakse otsingusüsteemi väljaande dokumendi andmestruktuurile vastav BibDocument klass (Joonis 24). Väljaannete andmestikud kaardistatakse BibDocument instantsideks, et saata need otsingusüsteemile indekseerimiseks. Lisaks kaardistatakse väljaannete otsingu vasted NEST'iga automaatselt BibDocument instantsideks.

```
public class BibDocument
{
    public string Id { get; set; } = default!;
    public string? Title { get; set; }
    public string? OriginalTitle { get; set; }
    public string? FirstAuthor { get; set; }
    public string? AllAuthors { get; set; }
    public string? RlId { get; set; }
    public string? SierraId { get; set; }
    public string? RiksId { get; set; }
    public string? UrramId { get; set; }
    public string? LanguageCode { get; set; }
    public string? LanguageName { get; set; }
    public string? MaterialTypeCode { get; set; }
    public string? MaterialTypeName { get; set; }
    public int? PublishYear { get; set; }
    public ICollection<Item>? Items { get; set; }
    public ICollection<Library>? Libraries { get; set; }
    public ICollection<string>? Isbns { get; set; }
    public ICollection<string>? Issns { get; set; }
    public ICollection<string>? Ismns { get; set; }
    public ICollection<string>? Keywords { get; set; }
}
```

Joonis 24. BibDocument klass.

Dokumentide indekseerimiseks on loodud meetod IndexManyAsync (Joonis 25). Meetod võtab sisendiks BibDocument kollektsiooni. Kirjed indekseeritakse bibliographies nimelisse indeksisse.

```
public async Task IndexManyAsync(IEnumerable<BibDocument> bibs)
{
    await _elasticClient.IndexManyAsync(bibs, "bibliographies");
}
```

Joonis 25. Meetod IndexManyAsync väljaannete indekseerimiseks.

Otsingu teostamiseks on loodud meetod SearchAsync (Joonis 26). Meetod võtab sisendiks päringusõne ning edastab selle otsingusüsteemile, kus neid otsitakse bibliographies nimelisest indeksist. Otsingusüsteemilt saadud vastus kaardistatakse SearchResponse instantsiks, mis lisaks otsingu vastetele sisaldab ka täiendavaid metaandmeid, nt leitud vastete arv, otsingu aeg ja maksimaalne relevantsuse skoor.

```
public async Task<ISearchResponse<BibDocument>> SearchAsync(string query)
{
    return await _elasticClient.LowLevel
        .SearchAsync<SearchResponse<BibDocument>>("bibliographies", query);
}
```

Joonis 26. Meetod SearchAsync väljaannete otsimiseks.

Eelneva joonise peal on näha, et meetod võtab sisendiks päringusõne. See päringusõne on Elasticsearchi domeenispetsiifilises keeles [44]. NEST'iga on võimalik luua päringuid ka tugevalt tüübitud kujul, kuid parima paindlikkuse saavutamiseks luuakse antud töös Elasticsearch päringuid „käsitsi" sõnede liitmisega.

5.2 MARC andmestiku lugemine

MARC andmestikust on tarvis välja lugeda väljaannete andmed (nt pealkiri, autorite nimed, teaviku laad jt) ning kaardistada need sobivatesse andmestruktuuridesse. MARC andmestiku lugemiseks kasutatakse antud töös MARC4J.Net teeki. Teek võimaldab XML või MARC vormingus sõne või voogu kaardistada IRecord liidesel põhinevaks instantsiks. IRecord põhineb MARC standardil ja pakub bibliokirje andmestiku lugemiseks abistavaid meetodeid.

Antud töös päritakse MARC andmestik JSON kujul ning sellest tingituna sai loodud laiendusmeetod, mis kaardistab JSON kujul MARC andmestiku IRecord liidesel põhinevaks instantsiks (vt Lisa 4 – MARC4J.Net JSON-i laiendusklass, lk 78).

IRecord instantsi kasutatakse kaardistusmeetodis (Joonis 27), kus selle abil luuakse BibDocument instants.

```
public static BibDocument? Map(IRecord? record, ContextDto contextDto)
{
    if (record == null) return null;

    var leader = record.Leader;
    var controlFields = record.GetControlFields();
    var dataFields = record.GetDataFields();

    var materialTypeCode = leader.TypeOfRecord.ToString();
    var materialTypeName = GetMaterialTypeName(materialTypeCode, contextDto);
    var publishYear = GetPublishYear(controlFields);
    var languageCode = GetLanguageCode(controlFields);
    var languageName = GetLanguageName(languageCode, contextDto);
    var isbns = GetIsbns(dataFields);
    var issns = GetIssns(dataFields);
    var ismns = GetIsmns(dataFields);
    var firstAuthor = FirstAuthor(dataFields);
    var allAuthors = GetAllAuthors(dataFields);
    var fullTitle = GetFullTitle(dataFields);
    var originalTitle = GetOriginalTitle(dataFields);
    var keywords = GetKeywords(dataFields);

    var bibDocument = new BibDocument
    {
        Title = fullTitle,
        OriginalTitle = originalTitle,
        FirstAuthor = firstAuthor,
        AllAuthors = allAuthors,
        PublishYear = publishYear,
        MaterialTypeCode = materialTypeCode,
        MaterialTypeName = materialTypeName,
        LanguageCode = languageCode,
        LanguageName = languageName,
        Isbns = isbns,
        Issns = issns,
        Ismns = ismns,
        Keywords = keywords
    };

    if (!IsValidRecord(bibDocument)) return null;

    return bibDocument;
}
```

Joonis 27. BibDocument instantsi kaardistusmeetod.

5.3 Väljaannete indekseerimine

Prototüübi jaoks kasutatakse testandmeid Sierrast. Väljaannete salvestamine otsingusüsteemi toimub serveripoolse rakenduse vahendusel. Väljaannete indekseerimiseks luuakse kontroller IndexDataFromSierra, mis käivitab rakenduses vajalikud protsessid (Joonis 28).

```
[HttpGet("searchEngine/indexDataFromSierra")]
public async Task IndexDataFromSierra()
{
    await _bll.SearchEngineService.IndexDataFromSierra();
}
```

Joonis 28. Sierra väljaannete indekseerimise kontroller serveripoolse rakenduses.

Kontrolleris kutsutakse välja meetod IndexDataFromSierra (Joonis 29). Meetodis seatakse esimese asjana API märgis, mis tagab ligipääsu Sierra API-le. Seejärel päritakse väljaanded, kaardistatakse väljaanded otsingusüsteemi jaoks sobivale kujule ning indekseeritakse otsingusüsteemis. Prototüübi jaoks indekseeritakse kokku ~500 000 väljaannet.

```
public async Task IndexDataFromSierra()
{
    await Sierra.Bibs.SetTokenAsync();

    var contextDto = await GetBibContextDto();

    var startId = await GetSierraStartId();
    var maxCount = startId + INDEX_FROM_SIERRA_MAX_COUNT;
    const int step = MAX_BIB_RESPONSE_COUNT;

    for (var i = startId; i < maxCount; i += step)
    {
        var bibDocuments = (await GetBibDocumentsFromSierraMarcJson(
                        i, i + step, contextDto)).ToList();

        if (bibDocuments.Any())
        {
            await Elasticsearch.Bibs.IndexManyAsync(bibDocuments);
        }
    }
}
```

Joonis 29. Meetod IndexDataFromSierra väljaannete indekseerimiseks.

5.4 Väljaannete pärimine serveripoolses rakenduses

Otsingud toimuvad serveripoolse rakenduse vahendusel. Kliendirakendusest saadetakse otsingupäring serveripoolsele rakendusele, kus selle alusel luuakse uus päring Elasticsearchi domeenispetsiifilises keeles [44] ning edastatakse otsingusüsteemile. Kliendirakendusest päringute vastuvõtmiseks luuakse kontroller SearchBibs (Joonis 30).

```
[HttpGet]
public async Task<SearchResponseDto> SearchBibs([FromQuery]
                                        SearchRequestDto searchRequestDto)
{
    return await _bll.SearchService.SearchBibsAsync(searchRequestDto);
}
```

Joonis 30. Otsingu kontroller serveripoolses rakenduses.

Kontroller võtab vastu GET päringu, kus võivad olla järgnevad otsinguparameetrid:

- q – põhiotsingu otsingusõne;
- lib – raamatukogu kood;
- lang – väljaande keele kood;
- type – teaviku laadi kood;
- y – publitseerimise aasta;
- kw – märksõnad;
- isxn – standardnumber.

Kontrolleris kutsutakse välja meetod SearchBibsAsync (Joonis 31). Meetodis luuakse otsinguparameetrite alusel otsingupäring Elasticsearchi domeenispetsiifilises keeles. Loodud otsingupäring saadetakse otsingusüsteemile. Otsingusüsteemilt saadud päringu vastus kaardistatakse sobivale kujule ning tagastatakse meetodi lõpus.

```
public async Task<SearchResponseDto> SearchBibsAsync(SearchRequestDto
                                                      searchRequestDto)
{
    var result = new SearchResponseDto {Total = 0, Bibs = new List<Bib>()};

    if (IsEmptyRequest(searchRequestDto)) return result;

    var query = SearchQueryBuilder.GetBibQueryString(searchRequestDto);
    var response = await Elasticsearch.Bibs.SearchAsync(query);

    result.Total = Math.Max(response.Total, 0);
    result.Bibs = BibMapper.Map(response.Hits.Select(h => h.Source)).ToList();

    return result;
}
```

Joonis 31. SearchBibAsync meetod.

GetBibQueryString meetodis toimub domeenispetsiifilise päringu koostamine. Lisaks leitakse selles meetodis regulaaravaldiste abil täpsustussõnadega märgitud otsingusõned. Järgneval joonisel 32 on esitatud regulaaravaldis, mille abil leitakse töötlemata otsingusõnest autori otsingusõne.

```
(?si)(?<=(\bautor:)|(\bauthor:)).*?(?=(\bpealkiri:)|(\btitle:)
|(\bautor:)|(\bauthor:)|(\baasta:)|(\blibrary:)|(\braamatukogu:)
|(\blibrary:)|(\bkeel:)|(\bkeel:)|(\blaad:)|(\btype:)|(\bmärksõnad:)
|(\bkeywords:)|(\bis(b|s|m)n:)|($))
```

Joonis 32. Regulaaravaldis autori otsingusõne leidmiseks.

Järgneval joonisel 33 on näha eelneva regulaaravaldise abil leitud otsingusõne. Leitud otsingusõne on joonisel märgitud sinise taustaga.

```
keisri hull autor: Jaan Kross isbn: 9985611918
```

Joonis 33. Regulaaravaldise abil leitud otsingusõne autori näitel.

Regulaaravaldiste loomisel lähtuti sellest, et otsitav otsingusõne peab järgnema märgitud täpsustussõnale ning kui leidub veel järgnevaid täpsustussõnu, siis peab otsitav otsingusõne jääma märgitud täpsustussõna ja järgneva täpsustussõna vahele. Täpsustussõnad võivad olla nii eesti- kui ka inglise keeles.

5.5 Väljaannete pärimine kliendirakenduses

Kliendirakenduse otsinguomadused on esitatud peatükis 3.5 Kliendirakendus, lk 35. Käesolevas peatükis on toodud välja, kuidas toimub kliendirakenduses otsingupäringu koostamine ja saatmine serveripoolsele rakendusele.

Kliendirakenduses tehakse otsingu teostamisel GET päring. Otsingupäring luuakse lähtuvalt serveripoolse rakenduse otsingu kontrollerist (vt 5.4 Väljaannete pärimine serveripoolses rakenduses, lk 58).

Otsingu teostamisel kutsutakse välja funktsioon onSearch (Joonis 34). Funktsioonis luuakse otsingu päringusõne, mis lisatakse URL-i. Seejärel teostatakse GET päring. Õnnestunud päringu puhul tagastatakse väljaanded ning otsinguvastete koguarv.

```
const onSearch = async () => {
    const searchQuery = searchQueryBuilder();
    const response = await SearchService.searchBibs(searchQuery);
    setBibs(response.bibs);
    setSearchTotalCount(response.total);
};
```

Joonis 34. Kliendirakenduse funktsioon onSearch.

Funktsiooniga searchQueryBuilder (Joonis 35) lisatakse kasutaja sisendi alusel otsingupäringusse otsinguparameetrid päringusõnena.

```
const searchQueryBuilder = (): string => {
    let arr = [];
    if (searchString?.length > 0) arr.push("q=" + searchString);
    if (library?.length > 0) arr.push("lib=" + library);
    if (language?.length > 0) arr.push("lang=" + language);
    if (materialType?.length > 0) arr.push("type=" + materialType);
    if (isxn?.length > 0) arr.push("isxn=" + isxn);
    if (publishYear?.length > 0) arr.push("y=" + publishYear);
    if (keywords?.length > 0) arr.push("kw=" + keywords);

    if (!(arr.length > 0)) return "";

    return "?" + arr.join("&");
};
```

Joonis 35. Kliendirakenduse funktsioon searchQueryBuilder otsingu päringusõne loomiseks.

Funktsiooniga searchBibs (Joonis 36) saadetakse GET päring ning tagastatakse päringust saadud vastus.

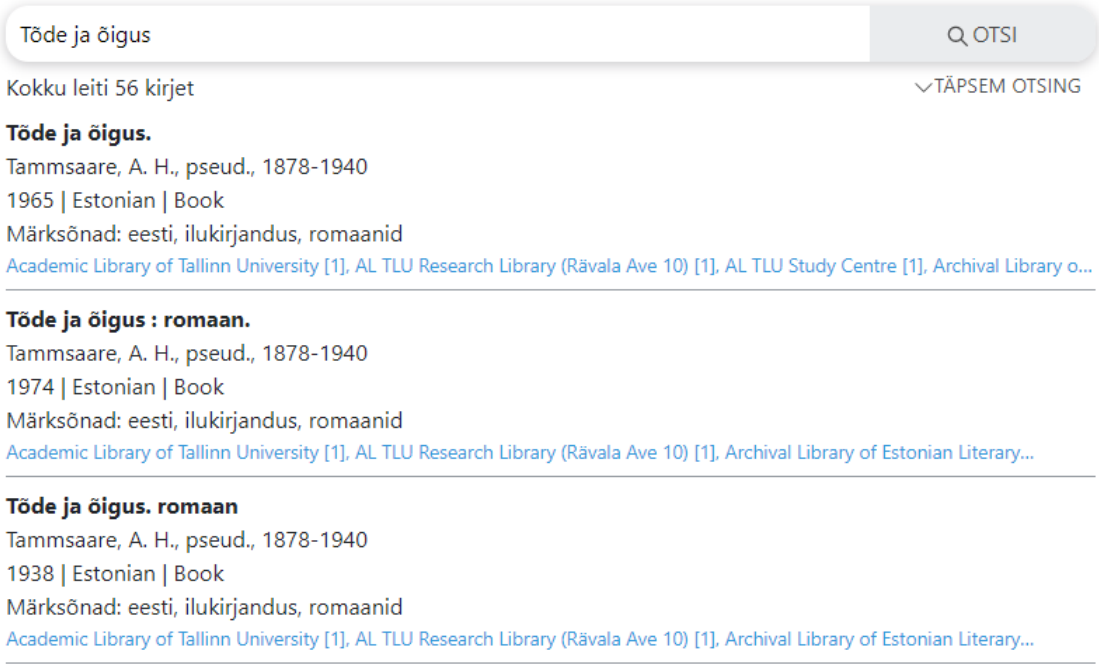
```
export const SearchService = {
    async searchBibs(searchQuery: string): Promise<BibSearchResponseDto> {
        const url = "https://localhost:5001/search" + searchQuery;
        const response: BibSearchResponseDto = await ServiceApi.get(url);

        return response;
    }
};
```

Joonis 36. Kliendirakenduse funktsioon searchBibs.

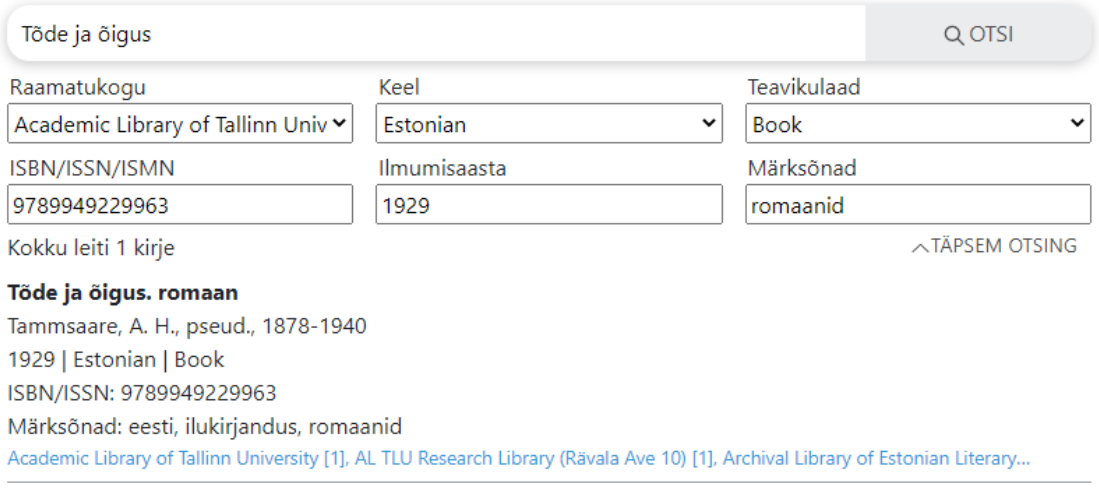
5.6 Otsingu näited

Joonisel 37 on teostatud lihtne otsing, kus põhiotsingut kasutades otsitakse väljaandeid otsingusisendiga „Tõde ja õigus“.



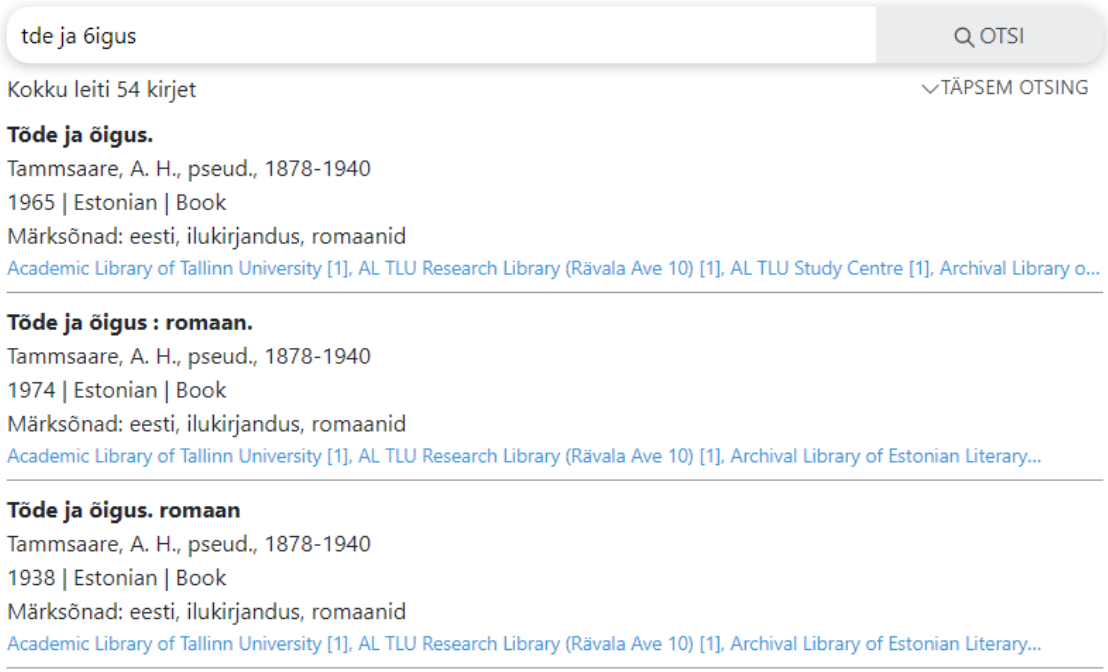
Joonis 37. Otsingu näide: Tõde ja õigus.

Joonisel 38 otsitakse väljaannet „Tõde ja õigus“ täpsemate otsinguparameetritega.



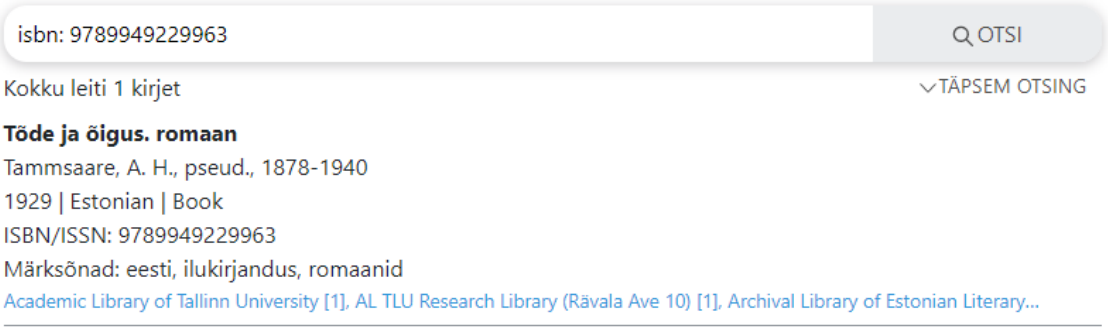
Joonis 38. Otsingu näide: Tõde ja õigus täpsemate otsinguparameetritega.

Joonisel 39 otsitakse väljaannet „Tõde ja õigus“ vigase sisendiga, kontrollides hägusotsingu funktsionaalsust.



Joonis 39. Otsingu näide: Tõde ja õigus vigase kirjapildiga.

Joonisel 40 otsitakse väljaannet standardinumbri järgi, kasutades põhiotsingus standardnumbri täpsustussõna.

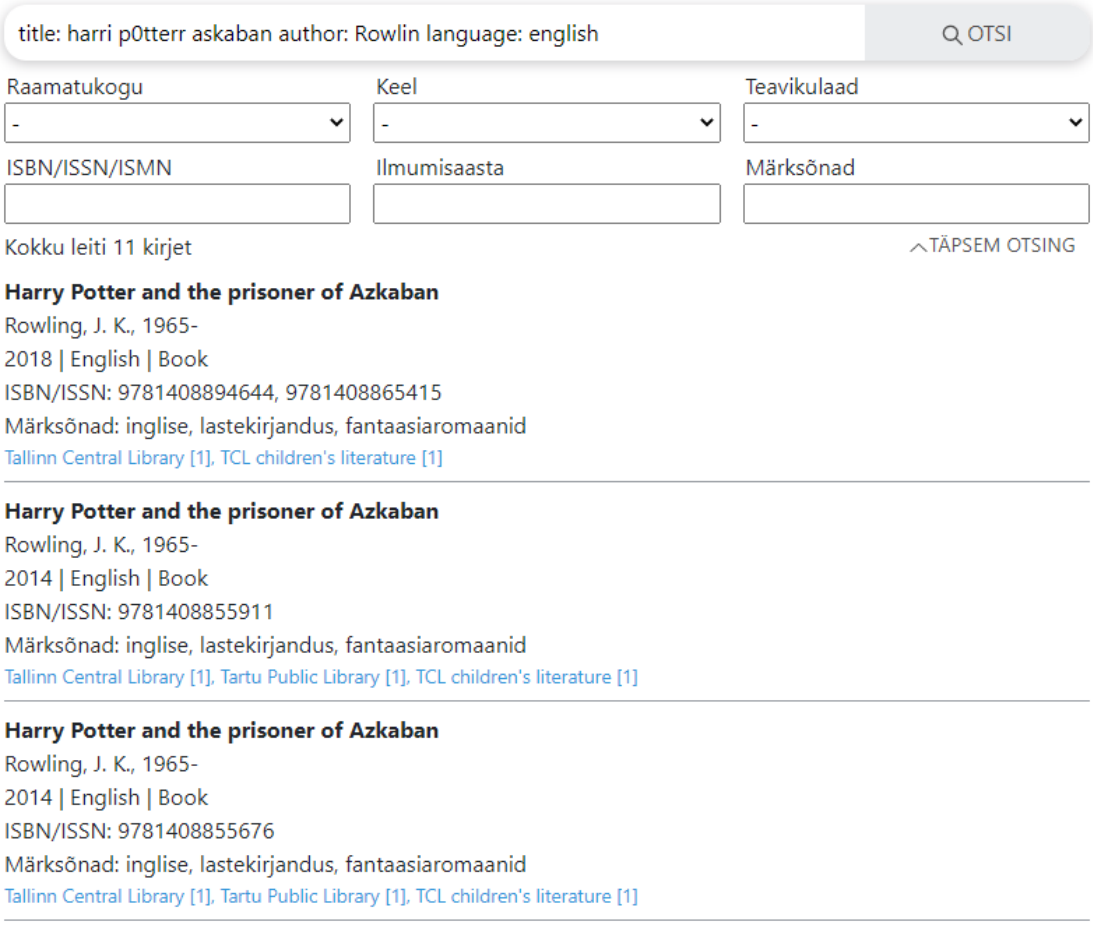


Joonis 40. Otsingu näide: otsing põhiotsingu standardnumbri täpsustussõnaga.

Joonisel 41 otsitakse väljaandeid, kasutades põhiotsingus mitut täpsustussõna.

Joonis 41. Otsingu näide: otsing kasutades põhiotsingus mitut täpsustussõna.

Joonisel 42 otsitakse väljaandeid vigaste sisenditega, kasutades põhiotsingus mitut inglise keelset täpsustussõna.



Joonis 42. Otsingu näide: otsing vigase kirjapildiga, kasutades põhiotsingus mitut inglise keelset täpsustussõna.

6 Kokkuvõte

Käesolev bakalaureusetöö jagunes kolme eesmärgi vahel. Töö esimene eesmärk oli luua serveripoolne rakendus, mille ülesanneteks on pärida andmeid välistest raamatukogusüsteemidest, lugeda MARC standardiga struktureeritud väljaannete andmestikke ning suhelda otsinguteenusega. Lisaks kuulus töö esimese eesmärgi juurde luua väljaannete otsimiseks ja kuvamiseks minimalistlik kliendirakendus. Töö teine eesmärk oli teostada analüüs ja selle tulemusena valida otsingutehnoloogia RL teenuse otsingusüsteemi loomiseks. Töö kolmas eesmärk oli kahe esimese eesmärgi tulemusel luua otsingusüsteemi prototüüp.

Serveripoolne rakendus sai loodud mitmekihilise arhitektuuriga ning lähtudes puhta koodi printsiipidest. Optimaalseima väljaannete päringumetoodika leidmiseks sai teostatud jõudlustest. Jõudlustestiga selgus, et kõige optimaalsem on pärida väljaannete andmestikke JSON kujul. MARC standardiga struktureeritud väljaannete lugemiseks sai kasutusele võetud MARC4J.Net teek. Lisaks sai loodud eelnimetatud teegile laiendusklass, mis võimaldab lugeda JSON kujul MARC andmestikku.

Otsingusüsteemi loomiseks valiti 4 alternatiivi: Lucene, Apache Solr, Elasticsearch ja Azure Cognitive Search. Tehnoloogiatega lähemalt tutvumiseks teostati ka näidislahendused. Valiku langetamiseks teostati mitme-kriteeriumi analüüs, kasutades AHP ja WPM kombineeritud meetodit. Analüüsi tulemusel osutus valitud tehnoloogiaks Elasticsearch.

Kasutades esimese osas loodud serveripoolset rakendust ja kliendirakendust ning teises osas analüüsi tulemusel valitud otsingutehnoloogiat, sai edukalt loodud otsingusüsteemi prototüüp. Valminud otsingusüsteem teostab otsinguid kiirelt, toetab hägusotsingut ning võimaldab eesti- ja inglise keelsete täpsustussõnade kasutamist.

Valminud prototüüplahenduse komponente on võimalik edukalt rakendada projektis Raamatud Liikuma. Lisaks saab töö praktilisi lahendusi kasutada uute otsingusüsteemide loomiseks või olemasolevate süsteemide täiustamiseks. Töös kasutatud AHP ja WPM

kombineeritud meetodit on võimalik kasutada sarnaste probleemide lahendamisel, kus valiku langetamine nõuab põhjalikumat analüüsi ning sõltub mitmest kriteeriumist.

Kasutatud kirjandus

[1] Eesti Rahvusraamatukogu, „auvis,“ 2018. [Võrgumaterjal]. Loetud aadressil: https://termin.nlib.ee/view/7020. [Kasutatud 15. mai 2021].

[2] Eesti Rahvusraamatukogu, „bibliokirje,“ 2018. [Võrgumaterjal]. Loetud aadressil: https://termin.nlib.ee/view/2837. [Kasutatud 4. mai 2021].

[3] Eesti Rahvusraamatukogu, „eksemplar,“ 2018. [Võrgumaterjal]. Loetud aadressil: https://termin.nlib.ee/view/613. [Kasutatud 6. mai 2021].

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach ja T. Berners-Lee, „Hypertext Transfer Protocol -- HTTP/1.1,“ juuni 1999. [Võrgumaterjal]. Loetud aadressil: https://tools.ietf.org/html/rfc2616#section-21. [Kasutatud 14. märts 2021].

[5] D. Crockford, „JSON,“ [Võrgumaterjal]. Loetud aadressil: https://www.json.org/json-en.html. [Kasutatud 14. märts 2021].

[6] P. E. Black, „Levenshtein distance,“ 15. mai 2019. [Võrgumaterjal]. Loetud aadressil: https://xlinux.nist.gov/dads/HTML/Levenshtein.html. [Kasutatud 16. mai 2021].

[7] Open Source Initiative, „About the Open Source Initiative,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://opensource.org/about. [Kasutatud 14. märts 2021].

[8] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures,* Irvine: University of California, 2000.

[9] „IEEE Standard Glossary of Software Engineering Terminology,“ *IEEE Std 610.12-1990,* pp. 1-84, 1990.

[10] Eesti Rahvusraamatukogu, „teavik,“ 2018. [Võrgumaterjal]. Loetud aadressil: https://termin.nlib.ee/view/1307. [Kasutatud 6. mai 2021].

[11] Eesti Keele Instituut, „IT terministandardi sõnastik, URL,“ 2021. [Võrgumaterjal]. Loetud aadressil: http://www.eki.ee/dict/its/index.cgi?Q=url&F=M&C06=et&C10=1. [Kasutatud 9. mai 2021].

[12] Eesti Rahvusraamatukogu, „väljaanne,“ 2018. [Võrgumaterjal]. Loetud aadressil: https://termin.nlib.ee/view/1012. [Kasutatud 6. mai 2021].

[13] Eesti Rahvusraamatukogu, „Paneme raamatud liikuma,“ 26. juuni 2019. [Võrgumaterjal]. Loetud aadressil: https://www.nlib.ee/et/uudised/paneme-raamatud-liikuma. [Kasutatud 25. aprill 2021].

[14] Deltmar OÜ, „Raamatukogud,“ [Võrgumaterjal]. Loetud aadressil: https://www.webriks.ee/blog/?page_id=943. [Kasutatud 29. aprill 2021].

[15] Urania COM OÜ, „URRAM-it kasutavad raamatukogud,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://www.urania.ee/?page_id=180. [Kasutatud 29. aprill 2021].

[16] „MARC4J.Net,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://github.com/mxurshid/MARC4J.Net. [Kasutatud 8. mai 2021].

[17] T. L. Saaty ja L. G. Vargas, Models, Methods, Concepts & Applications of the Analytic Hierarchy Process - Second Edition, New York: Springer, 2012.

[18] E. Triantaphyllou, Multi-Criteria Decision Making Methods: A Comparative Study, Dordrecht: Springer, 2000.

[19] H. Supriyono ja C. P. Sari, „Developing decision support systems using the weighted product method for house selection,“ 2018. [Võrgumaterjal]. Loetud aadressil: https://doi.org/10.1063/1.5042905. [Kasutatud 6. aprill 2021].

[20] C.-L. Hwang ja K. Yoon, Multiple Attribute Decision Making Methods and Applications - A State-of-the-Art Survey, Berlin: Springer, 1981.

[21] „What is REST,“ 2020. [Võrgumaterjal]. Loetud aadressil: https://restfulapi.net. [Kasutatud 25. aprill 2021].

[22] R. T. Fielding, „REST APIs must be hypertext-driven,“ [Võrgumaterjal]. Loetud aadressil: https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven. [Kasutatud 25. aprill 2021].

[23] P. Raspel, „Subjekt-tüüpi olemite seostamise üldised seaduspärasused,“ [Võrgumaterjal]. Loetud aadressil: https://enos.itcollege.ee/~priit/1.%20Andmebaasid/1.%20Loengumaterjalid/. [Kasutatud 26. aprill 2021].

[24] Microsoft, „Common web application architectures,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures. [Kasutatud 6. mai 2021].

[25] R. C. Martin, Clean Architecture: A Craftman's Guide to Software Structure and Design, Pearson, 2017.

[26] M. Seemann, Dependency Injection in .NET, Shelter Island: Manning, 2012.

[27] Microsoft, „Dependency injection in .NET,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection#service-lifetimes. [Kasutatud 17. aprill 2021].

[28] M. Fowler, Patterns of Enterprise Application Architecture, Boston: Addison-Wesley, 2003.

[29] Microsoft, „Implement the infrastructure persistence layer with Entity Framework Core,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/dotnet/architecture/microservices/microservice-ddd-cqrs-patterns/infrastructure-persistence-layer-implementation-entity-framework-core. [Kasutatud 25. aprill 2021].

[30] M. McCandless, E. Hatcher ja O. Gospodnetić, Lucene in Action, Second Edition, Manning, 2010.

[31] The Apache Software Foundation, „Apache Lucene Core,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://lucene.apache.org/core/. [Kasutatud 8. märts 2021].

[32] The Apache Software Foundation, „PoweredBy,“ 2019. [Võrgumaterjal]. Loetud aadressil: https://cwiki.apache.org/confluence/display/LUCENE/PoweredBy. [Kasutatud 13. märts 2021].

[33] GitHub, „lucene-solr,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://github.com/apache/lucene-

solr/releases?after=history%2Fbranches%2Flucene-solr%2Flucene_solr_4_7. [Kasutatud 1. aprill 2021].

[34] Apache Software Foundation, „A Quick Overview,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://solr.apache.org/guide/8_8/a-quick-overview.html. [Kasutatud 21. märts 2021].

[35] Apache Software Foundation, „IntegratingSolr,“ 2019. [Võrgumaterjal]. Loetud aadressil: https://cwiki.apache.org/confluence/display/solr/IntegratingSolr#IntegratingSolr-Lua. [Kasutatud 21. märts 2021].

[36] Apache Software Foundation, „SolrCloud,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://solr.apache.org/guide/6_6/solrcloud.html. [Kasutatud 21. märts 2021].

[37] Elasticsearch B.V., „What is Elasticsearch?,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://www.elastic.co/what-is/elasticsearch. [Kasutatud 14. märts 2021].

[38] Elasticsearch B.V., „FAQ on 2021 License Change,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://www.elastic.co/pricing/faq/licensing. [Kasutatud 14. märts 2021].

[39] The OSI Board of Directors, „The SSPL is Not an Open Source License,“ 19. jaanuar 2021. [Võrgumaterjal]. Loetud aadressil: https://opensource.org/node/1099. [Kasutatud 14. märts 2021].

[40] Open Source Initiative, „Licenses by Name,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://opensource.org/licenses/alphabetical. [Kasutatud 14. märts 2021].

[41] Elasticsearch B.V., „Introduction,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://www.elastic.co/guide/en/elasticsearch/client/net-api/current/introduction.html. [Kasutatud 14. märts 2021].

[42] Microsoft, „What is Azure Cognitive Search?,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://docs.microsoft.com/en-us/azure/search/search-what-is-azure-search. [Kasutatud 16. märts 2021].

[43] Elasticsearch B.V., „NEST - High level client,“ [Võrgumaterjal]. Loetud aadressil: https://www.elastic.co/guide/en/elasticsearch/client/net-api/current/nest.html.

[44] Elasticsearch B.V., „Query DSL,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html. [Kasutatud 9. mai 2021].

[45] K. Schwaber ja J. Sutherland, „The Scrum Guide - The Definitive Guide to Scrum: The Rules of the Game,“ November 2020. [Võrgumaterjal]. Loetud aadressil: https://www.scrumguides.org/scrum-guide.html. [Kasutatud 25. veebruar 2021].

[46] Elasticsearch B.V., „Lifetimes,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://www.elastic.co/guide/en/elasticsearch/client/net-api/current/lifetimes.html. [Kasutatud 27. märts 2021].

[47] Apache Software Foundation, „Solr Cores and solr.xml,“ 2021. [Võrgumaterjal]. Loetud aadressil: https://solr.apache.org/guide/8_8/solr-cores-and-solr-xml.html. [Kasutatud 7. aprill 2021].

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

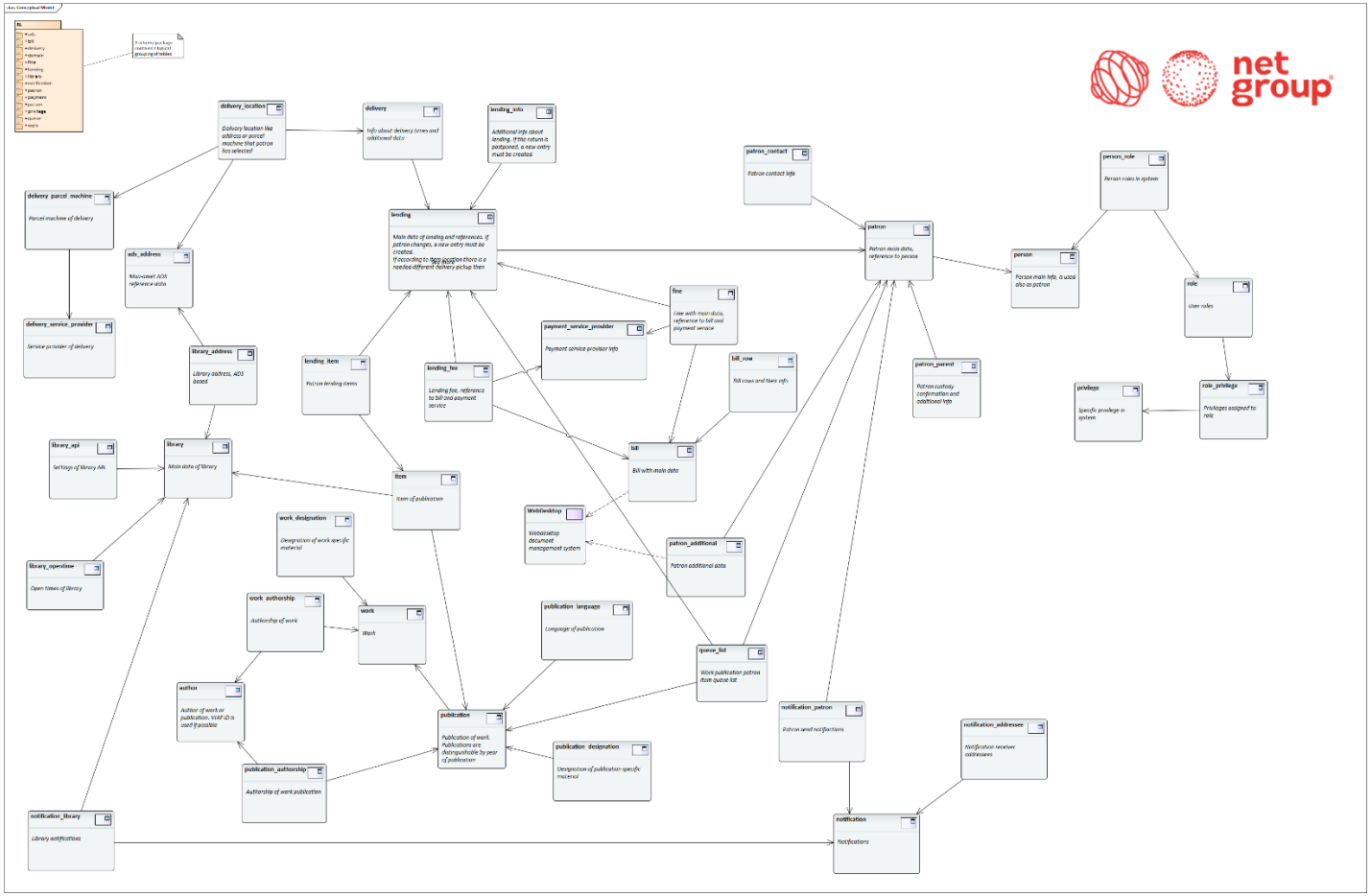
Mina, André Stender

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose Otsingusüsteemi loomine innovatsiooniprojektile Raamatud Liikuma, mille juhendajad on Margus Hanni ja Jaanus Pöial
   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.

17.05.2021

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 – Kontseptuaalne andmemudel



Joonis 43. Net Groupi koostatud kontseptuaalne andmemudel RL teenusele.

Lisa 3 – Testid

```
[Fact]
public async Task TestGetAndSavePublicationsWithMarcJson()
{
    var stopwatch = new Stopwatch();
    stopwatch.Start();

    // Set token before asynchronous tasks
    await _sierraRepo.Books.SetTokenAsync();

    const int startId = 1000001;
    const int maxCount = startId + 50000;
    const int step = 2000;

    var totalRequests = 0;
    var taskExecutionTimes = new List<TimeSpan>();
    var allFetchedBibs = new List<Bib>();
    var allSavedPublications = new List<Publication>();

    // One loop is considered as one task. One task consists of:
    //     - Fetching 2000 publications with JSON MARC data in one request.
    //     - Saving Publications and related data to DB. Related entities are:
    //     Languages, MaterialTypes and Countries.
    for (var i = startId; i < maxCount; i += step)
    {
        var taskStopWatch = new Stopwatch();
        taskStopWatch.Start();

        var (fetchedBibs, savedPublications) =
            await TestGetAndSavePublicationsAndItemsWithMarcJsonTask(i, step);

        totalRequests++;
        allFetchedBibs.AddRange(fetchedBibs);
        allSavedPublications.AddRange(savedPublications);

        taskStopWatch.Stop();
        taskExecutionTimes.Add(taskStopWatch.Elapsed);
    }

    stopwatch.Stop();

    WriteTestOutput("JSON", stopwatch, taskExecutionTimes, totalRequests,
        allFetchedBibs, allSavedPublications);
}

[Fact]
public async Task TestGetAndSavePublicationsWithMarcXml()
{
    var stopwatch = new Stopwatch();
    stopwatch.Start();

    // Set token before asynchronous tasks
    await _sierraRepo.Books.SetTokenAsync();

    const int startId = 1000001;
    const int maxCount = startId + 50000;
    const int step = 2000;

    var totalRequests = 0;
```

```
    var taskExecutionTimes = new List<TimeSpan>();
    var allFetchedBibs = new List<Bib>();
    var allSavedPublications = new List<Publication>();

    // One loop is considered as one task. One task consists of:
    //     - Fetching 2000 publications in one request.
    //     - Fetching XML MARC data for publications in separate requests.
    //     - Saving Publications and related data to DB. Related entities are:
    //     Languages, MaterialTypes and Countries.
    for (var i = startId; i < maxCount; i += step)
    {
        var taskStopWatch = new Stopwatch();
        taskStopWatch.Start();

        var (requests, fetchedBibs, savedPublications) =
            await TestGetAndSavePublicationsAndItemsWithMarcXmlTask(i, step);

        totalRequests += requests;
        allFetchedBibs.AddRange(fetchedBibs);
        allSavedPublications.AddRange(savedPublications);

        taskStopWatch.Stop();
        taskExecutionTimes.Add(taskStopWatch.Elapsed);
    }

    stopwatch.Stop();

    WriteTestOutput("XML", stopwatch, taskExecutionTimes, totalRequests,
        allFetchedBibs, allSavedPublications);
}

private async Task<Tuple<IEnumerable<Bib>, IEnumerable<Publication>>>
    TestGetAndSavePublicationsWithMarcJsonTask(int startId, int step)
{
    // Existing DB data
    var languages = _uow.Languages.GetAllAsync(false).Result.ToList();
    var materialTypes = _uow.MaterialTypes.GetAllAsync(false).Result.ToList();
    var countries = _uow.Countries.GetAllAsync(false).Result.ToList();

    var endId = startId + step - 1;
    var bibResponse = await _sierraRepo.Books
        .RangeWithAllFieldsAsync(startId, endId);

    var bibs = bibResponse.Entries?.ToList() ?? new List<Bib>();
    var publications = PublicationMapper.MapWithoutNulls(bibs);
    var sierraIds = bibs.Select(x => x.Id);
    var existingPublicationSierraIds =
        await _uow.Publications.GetExistingPublicationSierraIds(sierraIds);
    var publicationsToSave = publications
        .Where(x => !existingPublicationSierraIds.Contains(x.SierraId))
        .ToList();

    foreach (var publication in publicationsToSave)
    {
        // Add publication language
        var language = publication.Language;
        var existingLanguage = languages.FirstOrDefault(x =>
            x.Code == language?.Code &&
            x.Name == language.Name);
        publication.Language = existingLanguage ?? language;
        if (existingLanguage == null && language != null)
        languages.Add(language);

        // Add publication material type
        var materialType = publication.MaterialType;
        var existingMaterialType = materialTypes.FirstOrDefault(x =>
            x.Code == materialType?.Code &&
            x.Value == materialType.Value);
```

```
            publication.MaterialType = existingMaterialType ?? materialType;
            if (existingMaterialType == null && materialType != null)
                materialTypes.Add(materialType);

            // Add publication country
            var country = publication.Country;
            var existingCountry = countries.FirstOrDefault(x =>
                x.Code == country?.Code &&
                x.Name == country.Name);
            publication.Country = existingCountry ?? country;
            if (existingCountry == null && country != null)
                countries.Add(country);
        }

        var savedPublications = await _uow.Publications
            .AddRangeAsync(publicationsToSave);

        await _uow.SaveChangesAsync();

        return new Tuple< IEnumerable<Bib>, IEnumerable<Publication>>
            (bibs, savedPublications);
    }

    private async Task<Tuple<int, IEnumerable<Bib>, IEnumerable<Publication>>>
        TestGetAndSavePublicationsWithMarcXmlTask(int startId, int step)
    {
        var requests = 0;

        // Existing DB data
        var languages = _uow.Languages.GetAllAsync(false).Result.ToList();
        var materialTypes = _uow.MaterialTypes.GetAllAsync(false).Result.ToList();
        var countries = _uow.Countries.GetAllAsync(false).Result.ToList();

        var bibResponseTaskList = new List<Task<BibResponse>>();
        var marcDataTasks = new Dictionary<string, Task<string?>>();

        var endId = startId + step - 1;
        var bibResponseTask = _sierraRepo.Books
            .RangeWithoutMarcAsync(startId, endId);
        requests++;
        bibResponseTaskList.Add(bibResponseTask);

        for (var j = startId; j < startId + step; j++)
        {
            var marcResponseTask = _sierraRepo.Books.GetMarcXmlAsync(j);
            requests++;
            marcDataTasks.Add(j.ToString(), marcResponseTask);
        }

        await Task.WhenAll(bibResponseTaskList);
        await Task.WhenAll(marcDataTasks.Values);

        var bibs = bibResponseTaskList
            .SelectMany(x => x.Result.Entries?.ToList() ?? new List<Bib>())
            .ToList();
        var publications = PublicationMapper.MapWithoutNulls(bibs);
        var sierraIds = bibs.Select(x => x.Id);
        var existingPublicationSierraIds =
            await _uow.Publications.GetExistingPublicationSierraIds(sierraIds);
        var publicationsToSave =
            publications.Where(x =>
!existingPublicationSierraIds.Contains(x.SierraId)).ToList();

        foreach (var publication in publicationsToSave)
        {
            marcDataTasks
            .TryGetValue(publication.SierraId ?? "", out var marcData);
            publication.MarcData = marcData?.Result ?? null;
```

```
            // Add publication language
            var language = publication.Language;
            var existingLanguage = languages.FirstOrDefault(x =>
                x.Code == language?.Code &&
                x.Name == language.Name);
            publication.Language = existingLanguage ?? language;
            if (existingLanguage == null && language != null)
             languages.Add(language);

            // Add publication material type
            var materialType = publication.MaterialType;
            var existingMaterialType = materialTypes.FirstOrDefault(x =>
                x.Code == materialType?.Code &&
                x.Value == materialType.Value);
            publication.MaterialType = existingMaterialType ?? materialType;
            if (existingMaterialType == null && materialType != null)
             materialTypes.Add(materialType);

            // Add publication country
            var country = publication.Country;
            var existingCountry = countries.FirstOrDefault(x =>
                x.Code == country?.Code &&
                x.Name == country.Name);
            publication.Country = existingCountry ?? country;
            if (existingCountry == null && country != null)
             countries.Add(country);
        }

        var savedPublications = await
             _uow.Publications.AddRangeAsync(publicationsToSave);

        await _uow.SaveChangesAsync();

        return new Tuple<int, IEnumerable<Bib>,
IEnumerable<Publication>>(requests, bibs, savedPublications);
    }

    private void WriteTestOutput(string dataFormat, Stopwatch stopwatch,
IReadOnlyCollection<TimeSpan> taskExecutionTimes,
        int totalRequests, ICollection allFetchedBibs, ICollection
allSavedPublications)
    {
        var ts = stopwatch.Elapsed;
        var maxExecutionTime = taskExecutionTimes
             .Select(x => x.TotalSeconds).Max();
        var minExecutionTime = taskExecutionTimes
             .Select(x => x.TotalSeconds).Min();
        var taskAverageInSeconds = taskExecutionTimes
             .Select(x => x.TotalSeconds).Average();
        var taskGeometricMeanInSeconds = GeometricMean(taskExecutionTimes
             .Select(x => x.TotalSeconds).ToArray());
        var taskMedianInSeconds = Median(taskExecutionTimes
             .Select(x => x.TotalSeconds).ToArray());

        _testOutputHelper.WriteLine(@"Performance test completed.");
        _testOutputHelper.WriteLine($@"MARC data saved in {dataFormat}");
        _testOutputHelper.WriteLine(
            $@"Total elapsed time:
{ts.Hours:00}:{ts.Minutes:00}:{ts.Seconds:00}.{ts.Milliseconds}");
        _testOutputHelper.WriteLine($@"Total requests made to Sierra API:
{totalRequests}");
        _testOutputHelper.WriteLine($@"Total publications fetched from Sierra API:
{allFetchedBibs.Count}");
        _testOutputHelper.WriteLine($@"Total publications saved to database:
{allSavedPublications.Count}");
        _testOutputHelper.WriteLine($@"Total amount of tasks:
{taskExecutionTimes.Count}");
```

```
    _testOutputHelper.WriteLine($@"Max task execution time:
{maxExecutionTime:F2} s");
    _testOutputHelper.WriteLine($@"Min task execution time:
{minExecutionTime:F2} s");
    _testOutputHelper.WriteLine($@"Average mean of tasks:
{taskAverageInSeconds:F2} s");
    _testOutputHelper.WriteLine($@"Geometric mean of tasks:
{taskGeometricMeanInSeconds:F2} s");
    _testOutputHelper.WriteLine($@"Median of tasks: {taskMedianInSeconds:F2}
s");
}

private static double GeometricMean(double[] arr)
{
    var n = arr.Length;
    double sum = 0;

    for (var i = 0; i < n; i++)
    {
        sum = sum + Math.Log(arr[i]);
    }

    sum /= n;

    return Math.Exp(sum);
}

private static double Median(double[] arr)
{
    if (arr == null || arr.Length == 0)
        throw new Exception("Median of empty array is undefined.");

    Array.Sort(arr);

    var n = arr.Length;
    var mid = n / 2;
    var median = (n % 2 != 0) ? arr[mid] : (arr[mid] + arr[mid - 1]) / 2;
    return median;
}
```

Joonis 44. Jõudlustestide programmikood.

Lisa 4 – MARC4J.Net JSON-i laiendusklass

```
public class MarcJsonReader
{
    private const string LEADER = "leader";
    private const string FIELDS = "fields";
    private const string IND1 = "ind1";
    private const string IND2 = "ind2";
    private const string SUBFIELDS = "subfields";

    private MarcFactory MarcFactory { get; }

    public MarcJsonReader ()
    {
        MarcFactory = MarcFactory.Instance;
    }

    public IRecord Read(string json)
    {
        var jObject = JObject.Parse(json);

        var record = MarcFactory.NewRecord();
        SetLeader(record, jObject);
        SetFields(record, jObject);

        return record;
    }

    private static ILeader GetLeader(JObject jObject)
    {
        var jToken = jObject[LEADER];
        if (jToken == null)
        {
            throw new MarcException("Leader was not found.");
        }

        return new Leader(jToken.ToString());
    }

    private static void SetLeader(IRecord record, JObject jObject)
    {
        var leader = GetLeader(jObject);
        record.Leader = leader;
    }

    private void SetControlField(IRecord record, string tag, JToken value)
    {
        string stringValue = value.Value<string>();
        var dataField = MarcFactory.NewControlField(tag, stringValue);
        record.AddVariableField(dataField);
    }
```

```
    private void SetFields(IRecord record, JObject jObject)
    {
        var fields = jObject[FIELDS]?.Children<JObject>() ??
                 new JEnumerable<JObject>();

        foreach (var field in fields)
        {
            foreach (var (tag, value) in field)
            {
                switch (value)
                {
                    case JValue:
                        SetControlField(record, tag, value);
                        break;
                    case JObject:
                    {
                        var ind1 = (char) (value[IND1]
                          ?.FirstOrDefault() ?? ' ');
                        var ind2 = (char) (value[IND2]
                          ?.FirstOrDefault() ?? ' ');
                        var subFields = value[SUBFIELDS] ?? new JArray();

                        var subFieldStrings = new List<string>();
                        foreach (var jToken in subFields)
                        {
                            var subJObject = jToken as JObject;
                            var subTag = subJObject?.Properties()
                                  .Select(x => x.Name).FirstOrDefault();
                            if (subTag == null)
                            {
                                throw new
                                  MarcException("Tag cannot be empty.");
                            }

                            var subTagContent = subJObject?.Properties()
                                  .Select(x => x.Value)
                                  .FirstOrDefault() ?? "";
                            subFieldStrings.Add(subTag);
                            subFieldStrings.Add((string) subTagContent!);
                        }

                        var dataField = MarcFactory.NewDataField(
                          tag, ind1, ind2, subFieldStrings.ToArray());
                        record.AddVariableField(dataField);
                        break;
                    }
                }
            }
        }
    }
}
```

Joonis 45. MARC4J.Net JSON-i laiendusklass.

Lisa 5 – Lucene'i näidislahendus

Käesoleva lisa all on esitatud Lucene'i näidislahendus, mis hõlmab ülesseadmist, indeksi loomist, kirjete indekseerimist ning kirjete pärimist.

Ülesseadmine

Lucene'i teek on saadaval NuGet paketihoidlas ning teegi installeerimiseks on võimalik kasutada dotnet käsurealiidest (Joonis 46).

```
dotnet add package Lucene.Net --version 4.8.0-beta00014
```

Joonis 46. Lucene: dotneti käsk teegi installeerimiseks.

Lucene'i teegi abil on võimalik luua instantsid, mis võimaldavad indeksisse kirjutamist ning indeksist lugemist (Joonis 47). Kirjutajat kasutatakse edaspidi kirjete indekseerimiseks ning lugejat kirjete pärimiseks..

```
private IndexWriter GetWriter()
{
    var dir = FSDirectory.Open(IndexDir);
    var analyzer = new StandardAnalyzer(LuceneVersion48);
    var indexConfig = new IndexWriterConfig(LuceneVersion48, analyzer);
    var writer = new IndexWriter(dir, indexConfig);

    return writer;
}

private IndexReader GetReader(IndexWriter writer)
{
    var reader = writer.GetReader(applyAllDeletes: true);
    return reader;
}
```

Joonis 47. Lucene: meetodid indeksi kirjutaja ja lugeja tagastamiseks.

Kirjete indekseerimine

Kirjete indekseerimiseks on loodud meetod AddOrUpdateDocuments (Joonis 48). Duplikaatide vältimiseks otsitakse esmalt identifikaatori alusel üles olemasolevad dokumendid ning kustutatakse ära. Kogu tegevus toimub transaktsiooni sees ning meetodi lõpus kutsutakse välja ka kinnitus-käsklus.

```
public void AddOrUpdateDocuments<T>(IEnumerable<T> items)
{
    items = items.ToList();

    var query = new BooleanQuery();
    BooleanQuery.MaxClauseCount = 10000;

    foreach (var item in items)
    {
        var hasIdProperty = IdProperty.HasIdProperty<T>();
        if (hasIdProperty)
        {
            var id = IdProperty.GetIdValue<T, string>(item);
            query.Add(new TermQuery(new Term("Id", id)), Occur.SHOULD);
        }
    }

    var documents = CreateDocuments(items);

    using var writer = GetWriter();

    writer.DeleteDocuments(query);

    writer.AddDocuments(documents);

    writer.Commit();
    writer.Dispose();
}
```

Joonis 48. Lucene: meetod kirjete indekseerimiseks.

Kirjete pärimine

Kirjete pärimiseks on loodud meetod Search (Joonis 49). Meetodis luuakse sõnedest päring Lucene'i domeenispetsiifilises keeles. Meetodi sisendiks antud otsingusõna tükeldatakse eraldi sõnadeks ning iga sõna lõppu lisatakse „~" märk, mis määrab, et seda sõna otsitaks häguselt, Levenshteini kaugusega kuni 2 ühikut. Otsingu vasteid otsitakse pealkirja ja autori nime alusel. Otsinguga tagastatakse 5 kõige kõrgema skooriga leitud vastet.

```
public IEnumerable<T> Search<T>(string searchString)
{
    using var writer = GetWriter();
    using var reader = GetReader(writer);

    var analyzer = new StandardAnalyzer(LuceneVersion48);
    var searchFields = new[] {"Title", "Author"};
    var parser = new MultiFieldQueryParser(
        LuceneVersion48,
        searchFields,
        analyzer
    );
    var terms = searchString.Split(" ");
    terms = terms.Select(term => term + "~").ToArray();
    var query = new BooleanQuery();
    foreach (var term in terms)
    {
        query.Add(parser.Parse(term), Occur.SHOULD);
    }

    var searcher = new IndexSearcher(reader);

    var hits = searcher.Search<T>(query, 5);

    var result = new List<T>();

    foreach (var scoreDoc in hits.ScoreDocs)
    {
        var doc = searcher.Doc(scoreDoc.Doc);
        var item = doc.ToObject<T>();
        result.Add(item);
    }

    reader.Dispose();
    writer.Dispose();
    return result;
}
```

Joonis 49. Lucene: meetod hägusotsinguks.

Lisa 6 – Apache Solri näidislahendus

Käesoleva lisa all on esitatud Apache Solri näidislahendus, mis hõlmab ülesseadmist, indeksi loomist, kirjete indekseerimist ning kirjete pärimist.

Ülesseadmine

Apache Solri allalaadimiseks on erinevaid võimalusi. Käesolevas näidislahenduses laetakse Apache Solr alla Dockeri tõmmisfailina (Joonis 50).

```
docker pull solr
```

Joonis 50. Apache Solr: Dockeri käsk tõmmisfaili allalaadimiseks.

Esmakordseks konteineri käivitamiseks kasutatakse docker run käsku (Joonis 51). Lisaks luuakse joonisel oleva käsuga ka publication_core nimeline tuumik. Tuumik tähistab Apache Solris indeksit ja sellega seonduvaid konfiguratsiooni faile [36]. Käivitatud konteiner asub aadressil localhost:8983.

```
docker run -d -p 8983:8983 --name solr solr solr-precreate publication core
```

Joonis 51. Apache Solr: Dockeri käsk klastri loomiseks ja käivitamiseks.

Apache Solr REST veebiserveriga suhtlemiseks registreeritakse DI konteinerisse HTTP kliendi instants (Joonis 52).

```
services.AddHttpClient("solr", c =>
{
    c.BaseAddress = new Uri("http://localhost:8983/solr/");
});
```

Joonis 52. Apache Solr: HTTP kliendi instantsi registreerimine DI konteinerisse.

Kirjete indekseerimine

Kirjete indekseerimiseks on loodud meetod AddRangeAsync (Joonis 53). Kui indekseerimisel leitakse sama identifikaatoriga dokument, siis see kustutatakse ära ning selle asemele salvestatakse uus.

```
public async Task AddRangeAsync<T>(string index, IEnumerable<T> publications)
{
    var idPropertyName = IdProperty.GetIdPropertyName<T>();
    var urlString =
$"{index}/update/json/docs?commit=true&f=$FQN:/**&f=id:/{idPropertyName}";
    var documents = CreateDocuments(publications);
    var json = JsonConvert.SerializeObject(documents);
    var content = new StringContent(json, UnicodeEncoding.UTF8,
"application/json");

    await _httpClient.PostAsync(urlString, content);
}
```

Joonis 53. Apache Solr: meetod kirjete indekseerimiseks.

Kirjete pärimine

Kirjete pärimiseks on loodud meetod FuzzySearchAsync (Joonis 54). Apache Solr kasutab päringute teostamiseks Lucene'i domeenispetsiifilist keelt ning sellest tingituna luuakse meetodis sõnedest päring Lucene'i domeenispetsiifilises keeles. Meetodi sisendiks antud otsingusõna tükeldatakse eraldi sõnadeks ning iga sõna lõppu lisatakse „~" märk, mis määrab, et seda sõna otsitaks häguselt, Levenshteini kaugusega kuni 2 ühikut. Otsingu vasteid otsitakse pealkirja ja autori nime alusel. Otsinguga tagastatakse 10 kõige kõrgema skooriga leitud vastet.

```
public async Task<IEnumerable<Publication>> FuzzySearchAsync(string searchString)
{
    var searchStrings = searchString
        .Split(" ")
        .Select(s => s + "~").ToList();
    var joinTitleString = string
        .Join(" ", searchStrings.Select(s => "Title:" + s));
    var joinAuthorString = string
        .Join(" ", searchStrings.Select(s => "Author:" + s));
    var query = $"{joinTitleString} {joinAuthorString}";

    var urlString = $"publication_core/query?q={query}&rows=10&fl=Json&wt=json.nl";
    var itemResponse = await
_httpClient.GetFromJsonAsync<QueryResponse>(urlString);
    var jsons = itemResponse?.Response?.Docs
        ?.Select(doc => doc.Json) ?? new List<string>();

    var publications = GetDocuments<Publication>(jsons!);

    return publications;
}
```

Joonis 54. Apache Solr: meetod hägusotsinguks.

Lisa 7 – Elasticsearchi näidislahendus

Käesoleva lisa all on esitatud Elasticsearchi näidislahendus, mis hõlmab ülesseadmist, indeksi loomist, kirjete indekseerimist ning kirjete pärimist.

Ülesseadmine

Elasticsearchi allalaadimiseks on erinevaid võimalusi. Käesolevas näidislahenduses laetakse Elasticsearch alla Dockeri tõmmisfailina (Joonis 55).

```
docker pull docker.elastic.co/elasticsearch/elasticsearch:7.12.0
```

Joonis 55. Elasticsearch: Dockeri käsk tõmmisfail allalaadimiseks.

Esmakordseks konteineri käivitamiseks kasutatakse docker run käsku (Joonis 56). Joonisel oleva käsuga luuakse ühe-sõlmeline Elasticsearchi klaster. Käivitatud konteiner asub aadressil localhost:9200.

```
docker run -p 9200:9200 -p 9300:9300 -e "discovery.type=single-node"
docker.elastic.co/elasticsearch/elasticsearch:7.12.0
```

Joonis 56. Elasticsearch: Dockeri käsk klastri loomiseks ja käivitamiseks.

Elasticsearchi REST veebiserveri suhtlemiseks kasutatakse antud näidislahenduses Elasticsearchi NEST kliendi teeki. Teek on saadaval NuGet paketihoidlas ning teegi installeerimiseks on võimalik kasutada dotnet käsurealiidest (Joonis 57).

```
dotnet add package NEST
```

Joonis 57. Elasticsearch: dotneti käsk NEST kliendi teegi installeerimiseks.

Peale teegi installeerimist registreeritakse ElasticClient instants DI konteinerisse (Joonis 58). ElasticClient pakub abistavaid meetodeid otsingusüsteemiga suhtlemiseks HTTP vahendusel.

```
var settings = new ConnectionSettings(new Uri("http://localhost:9200"));
var elasticClient = new ElasticClient(settings);
services.AddSingleton<IElasticClient>(elasticClient);
```

Joonis 58. Elasticsearch: ElasticClient instantsi registreerimine DI konteinerisse.

Kirjete indekseerimine

Kirjete indekseerimiseks on loodud meetod IndexManyAsync (Joonis 59). Kirjed indekseeritakse bibliographies nimelisse indeksisse ning juhul, kui sellist indeksit veel ei eksisteeri siis see luuakse. Kui indekseerimisel leitakse sama identifikaatoriga dokument, siis see kustutatakse ära ning selle asemele salvestatakse uus.

```
public async Task IndexManyAsync(IEnumerable<BibDocument> bibs)
{
    await _elasticClient.IndexManyAsync(bibs, "bibliographies");
}
```

Joonis 59. Elasticsearch: meetod kirjete indekseerimine.

Kirjete pärimine

Kirjete pärimiseks on loodud meetod FuzzySearchAsync (Joonis 60). Päringu koostamiseks on kasutatud NEST'i tugevalt tüübitud domeenispetsiifilist keelt. Päringuga otsitakse kirjeid bibliographies nimelisest indeksist. Otsingu vasteid otsitakse pealkirja ja autori nime alusel. Otsing toimub häguselt, Levenshteini kaugusega kuni 2 ühikut. Otsinguga tagastatakse 5 kõige kõrgema skooriga leitud vastet.

```
public async Task<IEnumerable<Publication>> FuzzySearchAsync(string searchString)
{
    var result = await _elasticClient.SearchAsync<Publication>(s => s
        .Index("bibliographies")
        .From(0)
        .Size(5)
        .Query(q => q
            .MultiMatch(c => c
                .Fields(f => f
                    .Field(p => p.Title)
                    .Field(p => p.Author)
                )
                .Query(searchString)
                .Fuzziness(Fuzziness.Auto)
            )
        )
    );

    var publications = result.Documents;
    return publications;

}
```

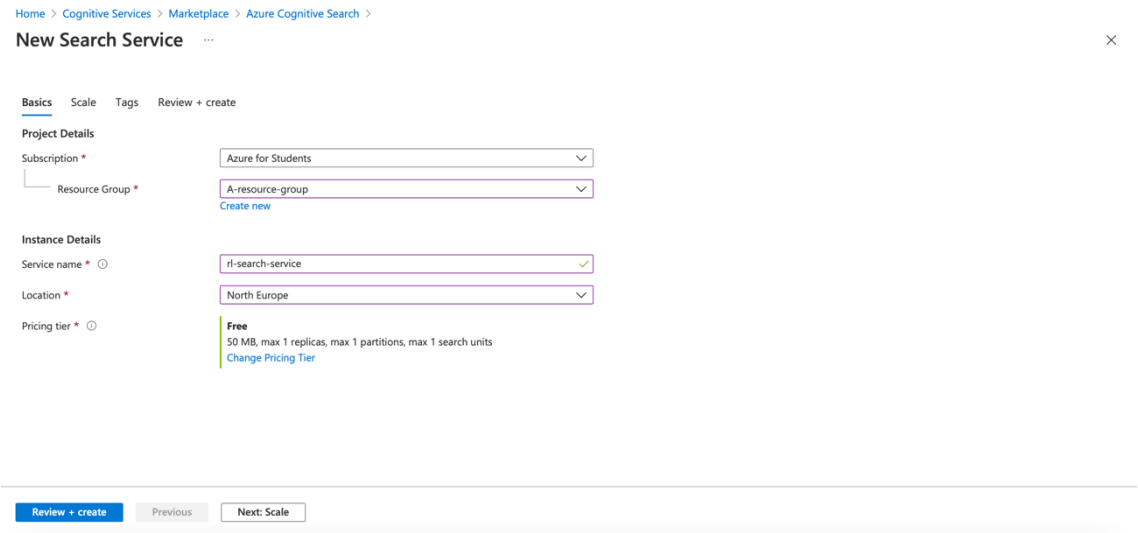
Joonis 60. Elasticsearch: meetod hägusotsinguks.

Lisa 8 – Azure Cognitive Searchi näidislahendus

Käesoleva lisa all on esitatud Azure Cognitive Searchi näidislahendus, mis hõlmab ülesseadmist, indeksi loomist, kirjete indekseerimist ning kirjete pärimist.

Ülesseadmine

Azure Cognitive Search kuulub Microsoft Azure pilveteenuste hulka ning selle kasutamiseks on esmalt tarvis registreerida Azure'is konto. Peale konto registreerimist ja sisse logimist tuleb otsida teenust nimega Azure Cognitive Search ning alustada uue teenuse loomist (Joonis 61). Azure pakub oma otsinguteenuse testimiseks tasuta võimalust, mis sisaldab 50 MB kettaruumi. Näidislahenduse jaoks on see piisav.



Joonis 61. Azure Cognitive Search: teenuse loomine.

Otsinguteenusega suhtlemiseks kasutatakse antud näidislahenduses Azure.Search.Documents teeki. Teek on saadaval NuGet paketihoidlas ning teegi installeerimiseks on võimalik kasutada dotnet käsurealiidest (Joonis 62).

```
dotnet add package Azure.Search.Documents
```

Joonis 62. Azure Cognitive Search: kliendi teegi installeerimine.

Peale teegi installeerimist registreeritakse DI konteinerisse 2 instantsi (Joonis 63). SearchIndexClient instants on mõeldud indeksite haldamiseks. SearchClient instants on mõeldud otsingupäringute jaoks.

```
var searchIndexClient = new SearchIndexClient(
    new Uri("https://rl-search-service.search.windows.net"),
    new AzureKeyCredential(
        Configuration["ApiKeys:RlAzureCognitiveSearchServiceApiKey"])
);
services.AddSingleton(searchIndexClient);

var searchClient = new SearchClient(
    new Uri("https://rl-search-service.search.windows.net"),
    "publications",
    new AzureKeyCredential(
        Configuration["ApiKeys:RlAzureCognitiveSearchServiceApiKey"])
);
services.AddSingleton(searchClient);
```

Joonis 63. Azure Cognitive Search: kliendi teegi registreerimine.

Kirjete indekseerimine

Uue indeksi loomiseks on tehtud meetod CreateOrUpdateIndexAsync (Joonis 64). Meetodiga luuakse uus bibliographies nimeline indeks. Indeksi dokumendi väljad luuakse klassi Publication alusel.

```
public async Task CreateOrUpdateIndexAsync()
{
    var fieldBuilder = new FieldBuilder();
    var searchFields = fieldBuilder.Build(typeof(Publication));

    var index = new SearchIndex("bibliographies", searchFields);

    await _searchIndexClient.CreateOrUpdateIndexAsync(index);
}
```

Joonis 64. Azure Cognitive Search: meetod indeksi loomiseks.

Kirjete indekseerimiseks on loodud meetod IndexDocumentsAsync (Joonis 65). Indekseerimisel on määratud ka tingimus, et kui mõne kirje indekseerimisel peaks tekkima viga, siis ei salvestata mitte midagi maha. Kui indekseerimisel leitakse sama identifikaatoriga dokument, siis see kustutatakse ära ning selle asemele salvestatakse uus.

```
public async Task IndexDocumentsAsync(IEnumerable<Publication> publications)
{
    var batch = IndexDocumentsBatch.Upload(publications);

    var options = new IndexDocumentsOptions { ThrowOnAnyError = true };

    await _searchClient.IndexDocumentsAsync(batch, options);
}
```

Joonis 65. Azure Cognitive Search: meetod andmete indekseerimiseks.

Kirjete pärimine

Kirjete pärimiseks on loodud meetod FuzzySearchAsync (Joonis 66). Azure Cognitive Searchiga on võimalik kasutada Lucene'i domeenispetsiifilist keelt, märkides SearchOptions instantsis QueryType = SearchQueryType.FULL. FuzzySearchAsync meetodis tükeldatakse sisendiks antud otsingusõna eraldi sõnadeks ning iga sõna lõppu lisatakse „~" märk, mis määrab, et seda sõna otsitaks häguselt, Levenshteini kaugusega kuni 2 ühikut. Otsingu vasteid otsitakse pealkirja ja autori nime alusel. Otsinguga tagastatakse 5 kõige kõrgema skooriga leitud vastet.

```
public async Task<IEnumerable<Publication>> FuzzySearchAsync(string searchString)
{
    var searchStrings = searchString
        .Split(" ")
        .Select(s => s + "~").ToList();

    var query = string.Join(" | ", searchStrings);

    var options = new SearchOptions
    {
        Size = 5,
        QueryType = SearchQueryType.Full,
        SearchFields = { "Title", "Author"}
    };

    SearchResults<Publication> searchResult = await
_searchClient.SearchAsync<Publication>(query, options);

    var publications = searchResult.GetResults().Select(x => x.Document);

    return publications;
}
```

Joonis 66. Azure Cognitive Search: meetod hägusotsinguks.