TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Artjom Vysotski IAIB222937
Artur Volnikov IAIB223240

Avaliku sektori andmete kogumine ja töötlemine suure keelemudeli tarbeks

Bakalaureusetöö

Juhendaja: Evelin Halling
PhD

Tallinn 2025

Autorideklaratsioon

Kinnitame, et oleme 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.

Autorid: Artjom Vysotski ja Artur Volnikov

19.05.2025

Lühikokkuvõte

Käesolev bakalaureusetöö keskendub avaliku sektori andmete automatiseeritud kogumise ja töötlemise süsteemi väljatöötamisele suure keelemudeli Bürokratt tarbeks. Töö eesmärgiks oli luua tõhus lahendus, mis võimaldab süstemaatiliselt koguda andmeid erinevatelt avaliku sektori veebilehtedelt (nt emta.ee, terviseamet.ee), neid puhastada ja struktureerida keelemudeli treenimiseks sobivasse formaati, välistades seejuures uudised ja muud mittevajalikud andmed.

Metoodika hõlmas Riigi Infosüsteemi Ameti (RIA) poolt ette valmistatud andmeallikate nimekirja analüüsi, andmete kogumise ja töötlemise tööriistade (Scrapy, BeautifulSoup, Playwright) kasutamist ning andmete töötlemise protsessi automatiseerimist. Andmete kogumise töövoog hõlmas URL-ide haldamist, veebilehtede sisu (DOM, PDF, DOCX) salvestamist S3 pilvemälus ning seejärel nende parsimist ja struktureerimist JSON-formaati, mis salvestatakse andmebaasi. Loodi ka skript andmete ajakohasuse kontrollimiseks ja uuendamiseks.

Töö tulemusena valmis automatiseeritud süsteem ja protsessi dokumentatsioon, mis võimaldab efektiivselt koguda ja töödelda suurt hulka avaliku sektori andmeid. Kogutud andmete kvaliteeti ja sobivust valideeriti pidevalt koostöös RIA ekspertidega. Kuigi keelemudeli reaalne treenimine antud andmetega jäi Bürokrati arendusjärgu tõttu testimata, loob loodud süsteem olulise aluse edasisteks arendusteks ja Eesti avalike teenuste parendamiseks Bürokrati abil.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 26 leheküljel, 7 peatükki, 4 joonist, 1 tabel.

Abstract

Collection and processing of public sector data for a large language model

This bachelor's thesis focuses on the development of an automated system for collecting and processing public sector data for the large language model Bürokratt. The aim of the thesis was to create an efficient solution that allows for the systematic collection of data from various public sector websites (e.g., emta.ee, terviseamet.ee), cleaning it, and structuring it into a format suitable for training the language model, while excluding news and other non-useful data.

The methodology involved analyzing a list of data sources prepared by the Estonian Information System Authority (RIA), using web scraping and parsing tools (Scrapy, BeautifulSoup, Playwright), and automating the data processing workflow. The data collection workflow included URL management, saving website content (DOM, PDF, DOCX) in S3 cloud storage, followed by parsing and structuring the data into JSON format, which was stored in a database. A script was also developed to check data freshness and update it as needed.

The result of the work is an automated system and process documentation, enabling the efficient collection and processing of a large volume of public sector data. The quality and suitability of the collected data were continuously validated in collaboration with RIA experts. Although the actual training of the language model with this data could not be tested due to Bürokratt's development stage, the created system provides a significant foundation for further developments and the improvement of Estonian public services through Bürokratt.

The thesis is in Estonian and contains 26 pages of text, 7 chapters, 4 figures, 1 table.

Lühendite ja mõistete sõnastik

API	Rakendusliides (*Application Programming Interface*)
DOM	Dokumendi objektimudel (*Document Object Model*); HTML- ja XML-dokumentide struktuuri esitusviis
HTML	Hüperteksti märgistuskeel (*HyperText Markup Language*)
JSON	JavaScripti objektimärgend (*JavaScript Object Notation*); kergekaaluline andmevahetusvorming
LLM	Suur keelemudel (*Large Language Model*)
Parsimine	Andmete struktuuri analüüsimine ja nende teisendamine edasiseks töötluseks sobivale kujule.
PDF	Kaasaskantav dokumendivorming (*Portable Document Format*)
RAG	intellektitehniline karkass, mis mudelite kvaliteedi tõstmiseks võtab andmeid välistest teadmuse allikatest (*Retrieval Augmented Generation*)
RIA	Riigi Infosüsteemi Amet
S3	Amazoni lihtne salvestusteenus (*Amazon Simple Storage Service*); objektide pilvemäluteenus
Spider	Python teegi Scrapy alusel tehtud andmete koguja
URL	Ühtne ressursilokaator (*Uniform Resource Locator*); veebiaadress
XML	Laiendatav märgistuskeel (*Extensible Markup Language*)

Sisukord

Jooniste loetelu

1 Sissejuhatus

Avaliku sektori käsutuses olev informatsioon on mahukas ja mitmekesine ressurss, mille efektiivne kasutamine võib märkimisväärselt parandada avalike teenuste kvaliteeti ja kättesaadavust. Kaasaegsed suured keelemudelid (LLM), nagu Eestis arendatav Bürokratt[1, 2], pakuvad uusi võimalusi selle informatsiooni töötlemiseks ja kodanikele suunatud teenuste automatiseerimiseks ning kasutajakogemuse täiustamiseks. Bürokrati eesmärk on pakkuda kodanikele kiiret ja ööpäevaringset tuge, vastates küsimustele ja aidates erinevate teenustega, vabastades sellega klienditeenindajate aega keerukamate ülesannete jaoks.

Suurte keelemudelite treenimiseks ja nende tõhusaks toimimiseks on aga kriitilise tähtsusega kvaliteetsete ja asjakohaste andmete olemasolu. Kuigi avaliku sektori andmeallikaid on Eestis rohkesti (näiteks ministeeriumite, ametite ja kohalike omavalitsuste veebilehed nagu emta.ee ja terviseamet.ee), on nende andmete süstemaatiline kogumine, puhastamine ja struktureerimine keeruline ja aeganõudev protsess. Manuaalne andmete kogumine ja analüüs ei ole suurte andmemahtude juures realistlik, olles lisaks ajamahukusele ka sageli ebatäpne. Seetõttu on vajalikud automatiseeritud lahendused, mis suudavad andmeid usaldusväärselt, järjepidevalt ja efektiivselt töödelda. Käesolev bakalaureusetöö keskendubki just sellise süsteemi väljatöötamisele ja kirjeldamisele, mis on spetsiaalselt loodud Eesti avaliku sektori andmete kogumiseks ja töötlemiseks suure keelemudeli Bürokratt tarbeks.

Töö on üles ehitatud järgmiselt: teises peatükis antakse ülevaade suurtest keelemudelitest, Bürokrati projektist ning avaliku sektori andmete rollist ja eripäradest. Kolmas peatükk keskendub töö üksikasjalikele eesmärkidele ja kirjeldab töö teostamise protsessi. Neljandas peatükis kirjeldatakse detailselt väljatöötatud andmekogumise ja -töötlemise metoodikat, sealhulgas kasutatud tööriistu, loodud töövoogu ning andmete struktureerimise põhimõtteid. Viiendas peatükis esitatakse loodud süsteemi tehniline ülesehitus ja saavutatud tulemused. Kuuendas peatükis analüüsitakse tulemuste hindamist, arutletakse töö käigus ilmnenud väljakutsete üle ning pakutakse välja võimalikke edasiarendusi. Töö lõpeb kokkuvõttega, kus võetakse kokku peamised tulemused ja hinnatakse eesmärkide saavutamist.

2 Taust: Suured keelemudelid ja avaliku sektori andmed

Kiiresti arenev tehisintellekti valdkond, eriti suurte keelemudelite (LLM) esiletõus, on avanud uued perspektiivid informatsiooni töötlemisel ja teenuste pakkumisel erinevates sektorites. Avalik sektor, mis haldab tohutul hulgal kodanike ja riigi toimimise seisukohast olulist informatsiooni, ei ole siinkohal erand. LLM-ide rakendamine võib oluliselt muuta seda, kuidas kodanikud suhtlevad riigiga ja kuidas avalikke teenuseid tarbitakse.

2.1 Suurte keelemudelite olemus

Suured keelemudelid on tehisintellekti mudelid, mis on treenitud massiivsetel tekstikorpustel, et mõista, genereerida ja manipuleerida inimkeelt. Nende võimekus ulatub lihtsast teksti genereerimisest ja küsimustele vastamisest kuni keerukate ülesannete lahendamiseni, nagu tekstide kokkuvõtmine, tõlkimine, koodi kirjutamine ja isegi loogiline arutelu. LLM-ide peamine tugevus seisneb nende võimes tuvastada mustreid ja seoseid suures hulgas andmetes, mis võimaldab neil pakkuda kontekstipõhist ja asjakohast informatsiooni.

2.2 Bürokratt: Eesti digiriigi järgmine samm

Eesti, olles tuntud oma e-valitsemise lahenduste poolest, on astumas järgmist sammu avalike teenuste digitaliseerimisel projektiga Bürokratt [1]. Bürokratt on sisuliselt avaliku sektori asutuste veebilehtedel olevate juturobotite ja virtuaalassistentide võrgustik. Selle peamine eesmärk on muuta avalike teenuste kasutamine inimeste jaoks radikaalselt lihtsamaks ja mugavamaks.

Bürokrati visioon näeb ette, et tulevikus saaks kodanik suhelda riigiga ühe keskse kanali kaudu, esitades oma küsimused ja saades vajalikud teenused ilma, et peaks ise teadma, milline asutus millise küsimusega tegeleb või kust täpselt mingit informatsiooni otsida. Inimesed ei peaks teenuste kasutamiseks omama eriteadmisi ega nägema selleks liigset vaeva. Bürokratt peaks pakkuma proaktiivseid teavitusi (nt tähtaegade saabumisel) ja

võimaldama kasutajal lihtsalt ning arusaadavalt langetada valikuid [2].

Selleks, et Bürokratt suudaks pakkuda kvaliteetset ja asjakohast tuge, on kriitilise tähtsusega tema äju"– keelemudel – treenimine mitmekesise ja korrektse informatsiooniga Eesti avaliku sektori kohta. Bürokratt peab mõistma Eesti seadusandlust, ametkondade pakutavaid teenuseid, protseduure ja muid asjakohaseid detaile, et kodanikke korrektselt juhendada ja nende päringutele vastata.

3 Töö eesmärgid ja teostuse protsess

Käesolevas peatükis kirjeldatakse põhjalikumalt bakalaureusetööle püstitatud eesmärke ning antakse ülevaade töö teostamise protsessist, sealhulgas koostööst tellijaga ja meeskonnasisesest töökorraldusest.

3.1 Töö üksikasjalikud eesmärgid

Bakalaureusetöö peamiseks sihtmärgiks oli luua automatiseeritud süsteem avaliku sektori andmete kogumiseks ja töötlemiseks, et toetada suure keelemudeli Bürokratt treenimist. Selleks püstitati järgmised üksikasjalikud eesmärgid:

- **Avaliku sektori andmete süstemaatiline kogumine:** Arendada lahendus, mis suudab metoodiliselt koguda andmeid erinevatelt Eesti avaliku sektori veebilehtedelt. See hõlmab võimekust töödelda mitmesuguseid andmeformaate, peamiselt HTML, aga ka PDF ja DOCX dokumente, milles sisaldub tekstiline informatsioon.
- **Andmete puhastamine ja filtreerimine:** Tagada, et kogutud andmetest eraldatakse ja välistatakse keelemudeli treenimise seisukohalt ebavajalik või ebasobiv info. Keskseks nõudeks oli uudisvoogude, võõrkeelse sisu (v.a eesti keel) ja mittetekstuaalsete elementide (nagu pildid ja XML-failid) eemaldamine või ignoreerimine.
- **Andmete struktureerimine keelemudelile sobival kujul:** Teisendada puhastatud andmed struktureeritud JSON-formaati. See formaat peab olema üles ehitatud viisil, mis hõlbustab andmete edasist kasutamist keelemudeli treeningprotsessides, säilitades võimalikult palju algsest kontekstist (nt pealkirjade ja tekstilõikude seosed).
- **Protsessi automatiseerimine ja ajakohasus:** Luua skriptid ja mehhanismid, mis automatiseerivad kogu andmekogumis- ja töötlemisprotsessi. See sisaldab andmeallikate perioodilist kontrolli ja kogutud andmete uuendamist, et tagada andmestiku võimalikult suur ajakohasus.
- **Efektiivsete tööriistade rakendamine:** Kasutada andmekogumiseks ja parsimiseks valdkonnas tunnustatud ja efektiivseid Pythoni teeke nagu Scrapy, BeautifulSoup

ning vajadusel Playwright (dünaamilise sisu jaoks), lähtudes nende võimekusest, kiirusest ja ressursikasutusest.

- **Tehnilise dokumentatsiooni ja kasutusjuhendi koostamine:** Dokumenteerida loodud süsteemi arhitektuur, komponentide funktsionaalsus ja tööprotsessid. Koostada kasutusjuhend, mis võimaldab süsteemi edasist haldamist, testimist ja potentsiaalset arendamist nii tellija kui ka teiste arendajate poolt.
- **Koostöö tellijaga (RIA):** Teha tihedat koostööd Riigi Infosüsteemi Ametiga, et tagada lahenduse vastavus nende ootustele ja vajadustele Bürokrati projekti kontekstis, ning valideerida regulaarselt tehtud tööd ja kogutud andmete kvaliteeti.

Nende eesmärkide saavutamine pidi looma vundamendi kvaliteetse ja asjakohase andmestiku tekkeks, mida saab kasutada Bürokrati keelemudeli "õpetamiseks"Eesti avaliku sektori teemadel.

3.2 Töö teostamise protsess ja metoodika

Bakalaureusetöö valmis iteratiivse protsessina, mis hõlmas nii iseseisvat tööd, meeskonnasisest koostööd kui ka regulaarset suhtlust tellija, Riigi Infosüsteemi Ametiga. Projekti juhtimiseks ja ülesannete haldamiseks kasutati GitLabi platvormi.

Tööprotsess oli üles ehitatud järgmiselt:

- **Nädala eesmärkide püstitamine ja valideerimine:** Igal nädalal toimusid regulaarsed videokoosolekud tellija esindajatega. Nendel kohtumistel seati konkreetsed eesmärgid järgmiseks nädalaks, arutati läbi tekkinud küsimused ja probleemkohad ning valideeriti eelneval perioodil tehtud tööd. Tellija poolt antud tagasiside oli oluline sisend süsteemi funktsionaalsuse ja andmete kvaliteedi tagamisel. Samuti arutati koosolekutel projekti üldisemat plaani ja võimalikke tulevikusuundi.
- **Meeskonnasisene koostöö ja ülesannete jaotus:** Lisaks tellijaga toimunud kohtumistele leidsid aset ka meeskonnasisesed regulaarsed arutelud. Tööülesanded jagati vastavalt meeskonnaliikmete tugevustele ja huvidele ning dokumenteeriti GitLabi task-idena. Iga meeskonnaliige tegeles talle määratud ülesannetega peamiselt iseseisvalt, endale sobival ajal, kuid tagades, et kokkulepitud tähtajast kinni peetaks.
- **Paindlik töökorraldus:** Kuigi igal liikmel olid oma vastutusvaldkonnad, tegutseti

vajadusel ka ühiselt keerukamate probleemide lahendamisel või olulisemate funktsionaalsuste väljatöötamisel. Selline paindlikkus võimaldas efektiivselt reageerida ettetulevatele takistustele ja tagada projekti sujuv edenemine.

- **Iteratiivne arendus:** Süsteemi arendati samm-sammult, alustades põhiliste komponentide loomisest ja liikudes edasi keerukamate funktsionaalsuste implementeerimiseni. Iga iteratsiooni järel koguti tagasisidet ja tehti vajalikke kohandusi.

Selline kombineeritud lähenemine – regulaarne suhtlus tellijaga, selge ülesannete jaotus meeskonnas ning iteratiivne arendustsükkel – võimaldas edukalt saavutada projektile seatud eesmärgid.

4 Andmekogumise ja -töötlemise metoodika

Käesolevas peatükis kirjeldatakse detailselt metoodikat, mida rakendati avaliku sektori andmete kogumiseks ja töötlemiseks suure keelemudeli Bürokratt tarbeks. Protsess hõlmas andmeallikate valikut, sobivate tööriistade identifitseerimist, andmete kogumise ja töötlemise töövoo loomist ning andmete struktureerimist.

4.1 Andmeallikate valik

Andmeallikate valiku aluseks oli Riigi Infosüsteemi Ameti (RIA) poolt ette valmistatud nimekiri avaliku sektori veebilehtedest. Nimekirja analüüsimisel hinnati iga potentsiaalse allika kasulikkust keelemudeli jaoks ning tehnilist teostatavust andmete kogumiseks. Keskenduti allikatele, mis sisaldavad kodanike jaoks olulist teavet ja millelt käsitsi informatsiooni otsimine võiks olla keeruline. Näidetena võib tuua Maksu- ja Tolliameti (emta.ee) ning Terviseameti (terviseamet.ee) veebilehed. Valikust välistati allikad, mis sisaldasid peamiselt uudisvooge. Samuti jäeti hilisemas töötlusetapis kõrvale andmeformaadid nagu XML ja pildid, mis ei sisalda keelemudeli jaoks otseselt kasulikku tekstilist informatsiooni.

4.2 Tööriistade valik ja põhjendus

Projekti tehnoloogilise baasi valikul lähtuti mitmest kaalutlusest, sealhulgas tellija soovidest, valitud keele ja andmebaasi sobivusest antud ülesande spetsiifikaga ning olemasolevate teekide küpsusest ja kogukonna toest [3].

4.2.1 Programmeerimiskeel Python

Projekti peamiseks programmeerimiskeeleks valiti Python[4]. Selle valiku peamiseks ajendiks oli Riigi Infosüsteemi Ameti kui tellija soov, mis tagab parema ühilduvuse ja integreeritavuse nende olemasolevate süsteemide ja kompetentsidega. Lisaks on Python laialdaselt tunnustatud oma lihtsa süntaksi, suure hulga kolmandate osapoolte teekide ning kiire arendustsükli poolest. Pythoni teegid nagu Scrapy, BeautifulSoup ja Playwright on

spetsiaalselt veebist andmete kogumiseks ja töötlemiseks loodud ning pakuvad robustset ja paindlikku funktsionaalsust, mis oli käesoleva projekti eesmärkide saavutamiseks hädavajalik.

4.2.2 Andmebaasisüsteem PostgreSQL

Andmete hoiustamiseks valiti PostgreSQL[5] relatsioonandmebaasisüsteem, samuti lähtudes RIA soovitusest. PostgreSQL on tuntud oma stabiilsuse, andmete terviklikkuse tagamise mehhanismide, laiendatavuse ja võimekuse poolest käsitleda keerukaid päringuid ning suuri andmemahte. Selle toetus JSON-andmetüübile oli eriti kasulik struktureeritud, kuid paindlike andmete salvestamiseks.

4.2.3 Scrapy

Valiti peamiseks andmekraapimise[6] (web scraping) tööriistaks selle võimekuse tõttu käsitleda suuri andmekoguseid kiiresti ja ressursisäästlikult. Scrapy võimaldab süstemaatiliselt läbida veebilehtede struktuuri ja koguda vajalikku sisu.

4.2.4 BeautifulSoup

Kasutati koos Scrapyga HTML-i parsimiseks. Kuna Scrapy enda parsimisvõimekus võib olla piiratud, siis BeautifulSoup[7] pakub paindlikumat ja võimekamat lahendust HTML-struktuurist relevantse info eraldamiseks.

4.2.5 Playwright

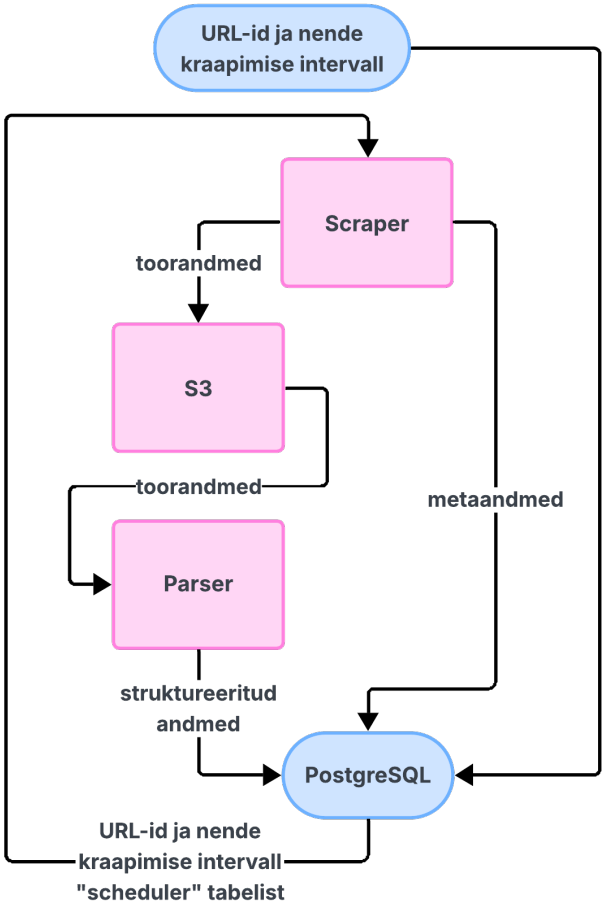
Rakendati vaid juhul, kui oli vaja parsida dünaamilisi veebilehti, mis kasutavad keerukamat JavaScripti sisu genereerimiseks. Kuigi Playwright[8], [9] on aeglasem kui Scrapy ja BeautifulSoup, on see võimekas tööriist JavaScripti poolt renderdatud sisu kättesaamiseks. Selle kasutamine oli kooskõlas ka tellija (RIA) soovitustega.

Alternatiividena kaaluti Seleniumit ja Pythoni requests teeki. Selenium, kuigi võimekas dünaamiliste lehtedega, jäeti kõrvale selle aegluse ja suure ressursinõudlikkuse tõttu, mis ei sobi suurte andmemahtude töötlemiseks. Requests teek on küll lihtne, kuid ei ole piisavalt võimekas keerukamate veebilehtede ja kraapimisstsenaariumite haldamiseks. Seega osutus Scrapy, BeautifulSoup ja Playwright kombinatsioon antud ülesande jaoks optimaalseimaks

[10]. Lisade nimekirjas on välja toodud tabel, mis illustreerib tööristade valikut (Lisa 2 – Tööriistade valiku põhjendus).

4.3 Andmekogumise ja -töötlemise töövoog

Väljatöötatud automatiseeritud töövoog koosneb mitmest etapist (Joonis 1):



Joonis 1. Andmekogumise ja -töötlemise töövoog.

4.3.1 URL-ide haldamine

Kogutavad alg-URL-id sisestatakse scheduler tabelisse koos uuendamisintervallidega, mis määravad, kui sageli tuleks vastavalt domeenilt andmeid uuesti koguda.

4.3.2 Andmete kraapimine (Scraping)

Kui käivitatakse scraper, otsitakse scheduler tabelist kõik URL-id, mille uuendamisintervall on möödas ja lisatakse need kraapimise järjekorda. Scraper kogub veebilehtedelt sisu,

milleks võib olla renderdatud HTML (DOM), PDF- või DOCX-failid. Dünaamilise sisu korral kasutatakse Playwright'i lehe renderdamiseks enne sisu salvestamist.

4.3.3 Andmete salvestamine (toorandmed)

Kogutud toorandmed (HTML, PDF, DOCX) salvestatakse S3[11] pilvemällu. HTML-failid salvestatakse täiskujul koos kõikide elementide ja võimaliku JavaScriptiga ("raw data").

4.3.4 Andmete parsimine

Järgmine protsess, parser, võtab S3-st HTML-failid ja kasutades BeautifulSoupi teeki, töötleb need. Parsimise käigus eemaldatakse ebavajalikud HTML-tagid, skriptid ja muu müra ning eraldatakse ainult tekstiline sisu, mis on keelemudeli jaoks relevantne. PDF- ja DOCX-failide töötlemiseks kasutatakse vastavaid teeke teksti eraldamiseks.

4.3.5 Andmete struktureerimine ja salvestamine

Parsitud ja puhastatud tekstiline informatsioon struktureeritakse JSON-objektideks. Iga JSON-objekt sisaldab tavaliselt vähemalt algset URL-i ja eraldatud tekstilist sisu. Need struktureeritud andmed salvestatakse andmebaasi edasiseks kasutamiseks. Toorandmete eraldi salvestamine ja struktureeritud andmete loomine väldib vajadust iga kord aeglast kraapimisprotsessi uuesti käivitada

4.4 Andmete puhastamine ja struktuur

Nagu eelnevalt mainitud, salvestatakse esmalt toorandmed (renderdatud HTML, PDF, DOCX). HTML-i puhul tähendab see kogu lehe DOM-struktuuri koos skriptide ja stiilidega. Parsimise etapis toimub oluline puhastus, kus eemaldatakse navigeerimiselemendid, reklaamid, jalused ja muu sisu, mis ei ole keelemudeli jaoks informatiivne. Eesmärk on saada kätte lehe põhiline tekstiline sisu.

Lõplikud struktureeritud andmed salvestatakse JSON-formaadis. Tüüpiline JSON-objekti struktuur võiks välja näha järgmine (Joonis 2):

```
{
  "url": "URL",
  "timestamp": "Datetime",
  "content": [
    {
      "header": "Veebilehe pealkiri",
      "paragraphs": [
        "Veebilehe sisu algab siit."
      ]
    },
    {
      "header": "Teine pealkiri",
      "paragraphs": [
        "Jargmine loik algab siit.",
        "Veel uks loik samas jaotises."
      ]
    }
  ]
}
```

Joonis 2. Struktureeritud JSON väljund veebilehe sisu salvestamiseks.

5 Rakenduse ülesehitus ja tulemused

Käesolevas peatükis kirjeldatakse detailselt avaliku sektori andmete kogumiseks ja töötlemiseks loodud süsteemi arhitektuuri, selle peamisi komponente ning nende implementatsiooni. Samuti esitatakse ülevaade saavutatud tulemustest.

5.1 Süsteemi arhitektuur

Loodud süsteem on mitmeastmeline ja modulaarne, tagades andmete süstemaatilise liikumise algallikatest kuni struktureeritud ja töödeldud kujul lõpphoiustamiseni. Arhitektuuri keskmes on PostgreSQL andmebaas, mis sisaldab nii kraapimise ajakava (*scraping_schedule*) kui ka kogutud linkide metaandmeid ja parsimistulemusi (*link_metadata*). Kogu protsess on konteineriseeritud Dockeriga, mis lihtsustab komponentide käivitamist ja haldamist.

Süsteemi töövoog on järgmine:

- **Ajakava haldur (Scheduler):** Määrab, milliseid veebiaadresse (URL) ja millise intervalliga tuleb kraapida. Andmed selleks loetakse scraping_schedule tabelist.
- **Andmekraapija (Scraper):** Scrapy-põhine komponent, mis käivitub Docker konteinerina. See loeb ajakavast töödeldavad URL-id, külastab vastavaid veebilehti, tuvastab ja järgib asjakohaseid linke ning salvestab lehtede toorsisu (HTML, PDF, DOCX) failidena.
- **Andmete vahepealne salvestus (S3):** Kraapija poolt kogutud toorfailid salvestatakse S3-ühilduvasse pilvemällu. Failinimed kodeeritakse Base64 [12], [13] formaati, et tagada unikaalsus ja vältida erimärkidest tulenevaid probleeme.
- **Andmete töötleja (Parser):** Eraldiseisev Pythoni skript, mis loeb S3-st salvestatud HTML-failid, puhastab need ebavajalikust sisust, eraldab relevantse tekstilise informatsiooni ja pealkirjad ning struktureerib andmed JSON-formaati.
- **Andmete lõpphoiustamine:** Parsitud JSON-andmed salvestatakse link_metadata tabelisse PostgreSQL andmebaasis, seostades need algse URL-iga ja lisades parsimise ajatempli.

5.2 Komponentide implementatsioon

Järgnevalt kirjeldatakse iga süsteemi põhikomponendi tehnilist teostust ja funktsionaalsust. Allpool on esitatud projekti struktuuri illustreeriv pilt (Joonis 3):

```
ria-projekt
├── config/                    # Configuration files
│   └── app_config.py          # Project-wide settings
├── models/                    # Data models
│   ├── database.py            # Database setup and models
│   └── scraped_data.py        # Data serialization models
├── scrapy_project/            # Scrapy project
│   ├── spiders/               # Scrapy spider
│   │   ├── spider.py          # Main spider class
│   │   └── settings.py        # Scrapy settings
├── test/                      # Tests
├── utils/                     # Utility scripts
│   ├── parsing_utils.py       # Parsing helpers
│   └── test_parse.py          # Test parsing
└── Dockerfile                 # Docker image definition
└── docker-compose.yml         # Multi-container Docker setup
└── Pipfile                    # Pipfile with requirements
└── Pipfile.lock               # Pipfile.lock
└── parse_raw_data.py          # Main parsing method
```

Joonis 3. Projekti struktuur.

Projekti stabiilsuse ja reprodutseeritavuse tagamiseks erinevates arendus- ja käituskeskkondades kasutati Pythoni sõltuvuste haldamiseks Pipenv[14] tööriista. Pipenv loob iga projekti jaoks virtuaalkeskkonna ning haldab projekti sõltuvusi Pipfile ja Pipfile.lock failide abil.

Pipfile sisaldab inimloetavat nimekirja projekti otsestest sõltuvustest ja nende versiooninõuetest, samuti arenduskeskkonna spetsiifilistest pakettidest. See võimaldab arendajatel selgelt deklareerida, milliseid teeke projekt vajab.

Pipfile.lock on automaatselt genereeritud fail, mis lukustab kõik projekti sõltuvused kindlatele versioonidele. See tagab deterministliku paigalduse – iga kord, kui projekt koos oma sõltuvustega paigaldatakse, kasutatakse täpselt samu teekide versioone, mis olid kasutusel Pipfile.lock faili loomise hetkel. See välistab probleemid, mis võivad tekkida sõltuvuste uuematest, potentsiaalselt mitteühilduvatest versioonidest, ning tagab, et

rakendus käitub kõikides keskkondades identselt. Projekti struktuuris (Joonis 3) on need failid ka näha, rõhutades nende olulisust projekti konfiguratsioonis.

5.2.1 URL-ide haldur ja ajakava plaanur (Scheduler)

Kraapimise ajakava haldamiseks kasutatakse PostgreSQL andmebaasis tabelit nimega *scraping_schedule*. See tabel on defineeritud SQLAlchemy mudelina ScrapingSchedule ning sisaldab järgmiseid välju:

- *id*: Unikaalne identifikaator (täisarv, primaarvõti, automaatselt suurenev).
- *title*: Kraapitava allika nimetus (tekst, kohustuslik).
- *url*: Kraapimise algusaadress (tekst, kohustuslik, unikaalne).
- *scraped_at*: Viimase kraapimise aeg (ajatempel, vaikimisi hetkeaeg).
- *scraping_interval*: Kraapimise intervall (ajavahemik, kohustuslik), mis määrab, kui sageli tuleks antud URL-i uuesti töödelda.
- *is_active*: Märge, kas antud ajakava kirje on aktiivne (boolean, vaikimisi tõene).

Kraapimise algus-URL-id sisestatakse sellesse tabelisse esialgu käsitsi. Kui Scraperi Docker konteiner käivitatakse, pärib see scraping_schedule tabelist aktiivsed (*is_active* = *True*) URL-id, mille puhul *scraped_at* koos *scraping_interval* väärtusega näitab, et järgmine kraapimine on vajalik (st *scraped_at* + *scraping_interval* <= praegune aeg). Pärast URL-i ja sellega seotud alamlehtede eduka kraapimise lõppu uuendatakse vastava kirje *scraped_at* väli kraapimisprotsessi lõpetamise ajatempliga.

5.2.2 Andmekraapija (Scraper)

Andmekraapija on realiseeritud Pythoni Scrapy teegi abil ning selle keskseks osaks on Spider klass.

Andmekraapija käitumist ja parameetreid juhitakse Scrapy projekti standardse *settings.py* konfiguratsioonifaili kaudu. Selles failis määratakse mitmed olulised seadistused, mis mõjutavad spiderite tööd, päringute tegemist ja andmete väljastamist.

Mõned näited projekti *settings.py* failis defineeritud parameetritest:

- BOT_NAME: Määrab Scrapy projekti nime, mis on kasutusel logides ja identifitsee-

rimisel.

- SPIDER_MODULES ja NEWSPIDER_MODULE: Defineerivad, millistes kaustades asuvad spiderite klassid.
- USER_AGENT ja DEFAULT_REQUEST_HEADERS: Võimaldavad seadistada HTTP päringute kasutajaagendi ja vaikepäised, et jäljendada veebibrauseri käitumist ja vältida kraapimise blokeerimist. Antud projektis kasutatakse keskkonnamuutujast loetavat *USER_AGENT* väärtust.
- ROBOTSTXT_OBEY = True: Tagab, et scraper järgib veebisaitide *robots.txt* failis seatud reegleid, mis on oluline eetilisuse seisukohast.
- FEEDS: Konfigureerib andmete väljastamise formaati ja sihtkohta. Näiteks võib siin määrata, et kogutud andmed salvestatakse JSON-faili kindla kodeeringu ja taandusega.
- PLAYWRIGHT_BROWSER_TYPE ja PLAYWRIGHT_LAUNCH_OPTIONS: Seadistused Playwright'i kasutamiseks, näiteks brauseri tüüp (nt Chromium) ja käivitusparameetrid (nt headless režiim).
- CONCURRENT_REQUESTS, CONCURRENT_REQUESTS_PER_DOMAIN, CONCURRENT_REQUESTS_PER_IP: Need parameetrid võimaldavad peenhäälestada samaaegsete päringute arvu nii globaalselt, domeenipõhiselt kui ka IP-aadressi põhiselt, et optimeerida kraapimise kiirust ja vältida serverite ülekoormamist. Väärtused loetakse sageli keskkonnamuutujatest, et tagada paindlikkus erinevates keskkondades.
- LOG_ENABLED, LOG_LEVEL, LOG_STDOUT: Seadistused logimise haldamiseks, sealhulgas logimise tase (nt INFO) ja väljund (nt standardväljund).
- DB_SESSION_FACTORY: Viide funktsioonile (*get_db_session*), mis loob andmebaasi sessiooni, võimaldades spideritel andmeid otse andmebaasi salvestada.

Need ja teised *settings.py* failis olevad seadistused annavad arendajale detailse kontrolli kraapimisprotsessi üle.

Kraapija peamised funktsionaalsused ja omadused on järgmised:

- **Initsialiseerimine:** Spideri käivitamisel määratakse algus-URL-id (saadud scraping_schedule tabelist), lubatud domeenid (tuletatakse esimesest algus-URL-ist) ja hoitakse meeles juba külastatud linke, et vältida korduvaid päringuid ja tsükleid.

- **Kehtivuse kontroll (*is_valid_url*):** Enne iga lingi järgimist kontrollitakse, kas see vastab määratud kriteeriumidele: see peab kuuluma lubatud domeeni alla, kasutama HTTP või HTTPS protokolli ning URL-i teekond ei tohi sisaldada võõrkeelseid (/ru/, /en/) ega uudistele viitavaid (uudised, news) segmente.
- **Keele tuvastamine (*detect_language*):** HTML-lehtede puhul üritatakse tuvastada keelt *<html>* tag'i *lang* atribuudi kaudu. Metaandmeid salvestatakse andmebaasi (link_metadata tabelisse) ainult eestikeelsete (et) või tundmatu keelega lehtede kohta. Muukeelsed lehed (nt en, ru) jäetakse edasisest töötlusest välja (*ignore_language* meetod).
- **JavaScripti vajaduse tuvastamine (*requires_javascript*):** Kraapija püüab tuvastada, kas lehe sisu täielikuks kuvamiseks on vajalik JavaScripti käivitamine, otsides lehe lähtekoodist levinud JavaScripti raamistikele (React, Angular, Vue) või dünaamilisusele viitavaid indikaatoreid (*<script*, *spinner*, *loading*).
- **Dünaamilise sisu kraapimine (*scrape_with_playwright_async*):** Kui tuvastatakse JavaScripti vajadus, kasutatakse Playwright teeki lehe asünkroonseks renderdamiseks Chromium brauseris [15], [16]. Lehte keritakse mitu korda allapoole, et tagada kogu dünaamilise sisu laadimine, enne kui HTML-sisu salvestatakse.
- **Staatilise sisu kraapimine (*scrape_with_beautifulsoup*):** Lihtsamate, staatiliste HTML-lehtede puhul kasutatakse sisu parsimiseks ja linkide leidmiseks BeautifulSoup teeki otse Scrapy vastusest.
- **Failitüüpide käsitlemine:**
  - PDF (*check_for_pdf*, *save_pdf*): Kui Content-Type päis viitab PDF-failile, salvestatakse see otse baitidena.
  - HTML: HTML-lehtede sisu salvestatakse tekstifailina (.html laiendiga).
  - DOCX/DOC: DOCX ja DOC failidele viitavad lingid tuvastatakse, kuid nende spetsiifiline allalaadimine ja salvestamine toimub sarnaselt PDF-idele, kui Scrapy need otse kätte saab, või kui Playwright need kättesaadavaks teeb.
  - XML (*ignore_xml*): XML-failidele viitavad URL-id ignoreeritakse.
- **Sisu salvestamine (*save_content*, *create_folder_and_file*):** Kraabitud HTML-sisu salvestatakse failisüsteemi data kausta, kus iga domeeni jaoks luuakse eraldi alamkaust. Failinimed (URL-i teekond ilma domeenita) kodeeritakse Base64 formaati (*encode_url* meetod, eemaldades täitemärgid), et tagada turvaline ja ühilduv failinimi, ning

lisatakse .html laiend.

- **Metaandmete salvestamine (*save_metadata_to_db*):** Iga kraabitud (ja keelefiltri läbinud) URL-i kohta salvestatakse või uuendatakse kirje link_metadata tabelis PostgreSQL andmebaasis. Salvestatav info sisaldab URL-i, tuvastatud keelt, serveri poolt antud viimase muutmise aega (Last-Modified päis), HTTP staatusekoodi ja kraapimise aega.

5.2.3 Andmete vahepealne salvestus (S3)

Kraapija poolt kogutud toorfailid (HTML, PDF, DOCX) salvestatakse S3-ühilduvasse pilvemällu. Ehkki hetkel toimub failide salvestamine lokaalsesse failisüsteemi (*data* kausta), on arhitektuuris ette nähtud S3 kasutamine, mis tähendaks *create_folder_and_file* meetodi kohandamist S3 API-ga suhtlemiseks. See oli tellija poolt välja toodud nõue.

Failide nimede kodeerimine toimub URL-i selle osa peal, mis järgneb domeenile. See osa kodeeritakse Base64 formaati ja eemaldatakse lõpust (=) täitemärgid, et saada puhas string. See tagab unikaalsed ja platvormist sõltumatud failinimed.

5.2.4 Andmete töötleja (Parser)

Parser on eraldiseisev Pythoni skript, mis on mõeldud S3 HTML-failide lugemiseks, nende parsimiseks ja struktureeritud JSON-andmete genereerimiseks ning andmebaasi salvestamiseks.

Parseri peamised funktsioonid ja loogika:

- Initsialiseerimine ja keskkonnamuutujad: Skript kasutab *RAW_DATA_DIR* keskkonnamuutujat (lahendus lokaalse kaustaga. See oli samuti tellija palve selleks, et hiljem saaksid nad seda ise testida ja seadistada endale sobival kujul), et leida toorandmete kaust (vaikimisi */app/data*). Logimine on seadistatud, et jälgida protsessi kulgu ja võimalikke vigu.
- Failide töötlemise tsükkel (*process_all_raw_data*, *process_files*): Skript itereerib läbi *RAW_DATA_DIR* alamkaustade (mis vastavad domeenidele) ja iga alamkausta sees töötleb kõiki *.html* laiendiga faile.
- URL-i taastamine failinimest (*extract_url_from_filename*): Kuna failinimed on

Base64 kodeeritud, siis enne parsimist dekodeeritakse failinimi tagasi algseks URL-i osaks. Lisaks konstrueeritakse täielik URL, kasutades kausta nime (domeeni) ja tagades *https://www.* prefiksi olemasolu, kui skeem puudub. Funktsioon arvestab ka Base64 täitemärkide (=) võimaliku puudumisega kodeeritud nimes.

- HTML-i parsimine (*parse_html_to_json*):
  - HTML-faili sisu loetakse sisse ja parsitakse BeautifulSoup teegiga (*html.parser*).
  - Eraldatakse lehe pealkiri (<*title*> tag).
  - Struktureeritud sisu eraldamiseks itereeritakse läbi kõikide pealkirjaelementide (h1 kuni h6) ja neile järgnevate tekstilõikude (p). Iga pealkiri koos sellele järgnevate lõikudega moodustab ühe sektsiooni JSON-väljundis.
  - Lõplik JSON-struktuur sisaldab algset URL-i, lehe pealkirja ja sisu, mis on esitatud sektsioonide loendina (iga sektsioon sisaldab pealkirja ja lõikude loendit).
- Andmete salvestamine andmebaasi:
  - Parsitud JSON-andmed salvestatakse PostgreSQL andmebaasi *link_metadata* tabelisse.
  - Kasutatakse SQLAlchemy *LinkMetadata* mudelit. Enne salvestamist kontrollitakse, kas antud URL-ile vastav kirje on juba olemas.
  - Kui kirje on olemas, uuendatakse *parsed_at* ajatempel, *parsed_data* väli (JSON-andmetega) ja vajadusel *status_code*.
  - Kui kirjet pole, luuakse uus kirje koos parsimise aja, JSON-andmete ja vaikimisi staatusekoodiga 200.
  - Andmebaasi sessioonid hangitakse *get_db_session* funktsiooni kaudu ja muudatused kinnitatakse (*session.commit()*) või tagasi võetakse (*session.rollback()*) vigade ilmnemisel.

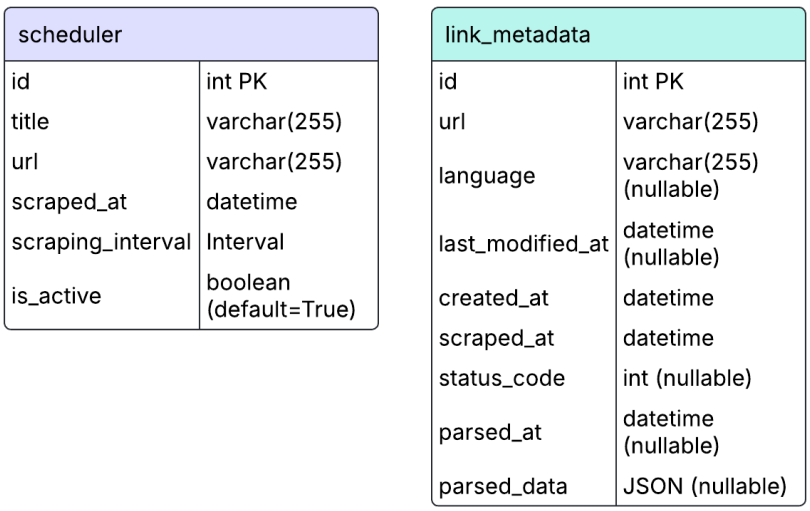
5.2.5 Andmete lõpphoiustamine (andmebaas)

Andmete lõpphoiustamiseks kasutatakse PostgreSQL andmebaasi, mis oli tellija (RIA) poolt soovitatud valik. Andmebaasi struktuur on defineeritud Pythoni SQLAlchemy teegi abil. Kasutusel on kaks peamist tabelit (Joonis 4):

- ***link_metadata*:** Sisaldab infot iga unikaalse kraabitud URL-i kohta.
  - *id*: Unikaalne identifikaator.

- *url*: Kraabitud lehe täielik URL (tekst, unikaalne).
- *language*: Tuvastatud lehe keel (tekst, võib olla tühi).
- *last_modified_at*: Serveri poolt antud viimase muutmise aeg (ajatempel, võib olla tühi).
- *created_at*: Kirje loomise aeg andmebaasis (ajatempel, vaikimisi hetkeaeg).
- *scraped_at*: URL-i viimase kraapimise aeg (ajatempel, vaikimisi hetkeaeg).
- *status_code*: HTTP staatusekood kraapimisel (täisarv, võib olla tühi).
- *parsed_at*: HTML-sisu parsimise ja JSON-iks teisendamise aeg (ajatempel, võib olla tühi). *parsed_data*: Parsitud ja struktureeritud sisu JSON-formaadis (JSON-tüüp andmebaasis, võib olla tühi).

- ***scraping_schedule*:** Kirjeldatud punktis 5.2.1, haldab kraapimise ajakava.



Joonis 4. Andmebaasi diagramm.

Andmebaasiühendus luuakse *get_db_session* funktsiooni abil, mis konstrueerib ühenduse stringi keskkonnamuutujatest (*POSTGRES_USER*, *POSTGRES_PASSWORD*, *POSTGRES_HOST*, *POSTGRES_PORT*, *POSTGRES_DB*). SQLAlchemy *Base.metadata.create_all(engine)* tagab, et vastavad tabelid luuakse andmebaasis, kui need veel ei eksisteeri.

5.2.6 Automatiseerimis- ja orkestreerimisskript

Kogu süsteemi komponentide käivitamine, haldamine ja omavaheline sidumine on lahendatud Docker Compose tööriista abil. *docker-compose.yml* fail kirjeldab mitmekonteinerilist

rakendust, defineerides iga teenuse jaoks vajalikud ehitusjuhised, käivitamiskäsud, pordid, andmemahud ja keskkonnamuutujad.

- **ria-project** (Andmekraapija): See teenus ehitatakse projekti juurkataloogis asuva *Dockerfile*'i põhjal. Määratud on käivitamiskäsk, konteineri nimi, taaskäivituspoliitika (*unless-stopped*) ning andmemahtude sidumine lokaalse failisüsteemiga. Olulised on ka keskkonnamuutujad, mis edastatakse konteinerile, näiteks andmebaasi ühenduse parameetrid (*POSTGRES_USER*, *POSTGRES_PASSWORD*, *POSTGRES_HOST*, *POSTGRES_DB*), ajatsoon (*TZ*) ning Scrapy jõudlusseaded (*CONCURRENT_REQUESTS* jne). See teenus sõltub *postgres* teenusest.
- **postgres** (Andmebaas): Kasutab standardset PostgreSQL 14. Määratud on konteineri nimi, taaskäivituspoliitika, andmemaht (*postgres-data*), kuhu salvestatakse andmebaasi andmed, ning keskkonnamuutujad PostgreSQL serveri seadistamiseks (kasutaja, parool, andmebaasi nimi). Samuti on pordid seadistatud nii, et andmebaasile pääseb ligi ka väljastpoolt Docker võrku.
- **parser** (Andmete Töötleja): Sarnaselt kraapijale ehitatakse see teenus samast *Dockerfile*'ist. Ka siin on määratud käivitamiskäsk, konteineri nimi, taaskäivituspoliitika ja andmemahtude sidumine. Keskkonnamuutujate kaudu antakse parserile teada toorandmete asukoht (*RAW_DATA_DIR*) ja andmebaasi ühenduse parameetrid. See teenus sõltub nii *ria-project* kui ka *postgres* teenustest.

docker-compose.yml faili kasutamine lihtsustab oluliselt kogu süsteemi ülesseadmist ja käivitamist üheainsa käsuga (*docker-compose up*), tagades, et kõik komponendid on korrektselt konfigureeritud ja omavahel ühendatud.

Kogu süsteemi automatiseerimine ja komponentidevaheline orkestreerimine on lahendatud järgmiselt:

- **Kraapija (Scraper):** Käivitatakse Docker[17] konteinerina. Konteineri käivitamisel alustab Scrapy spider tööd, lugedes esmalt scraping_schedule tabelist vajalikud ülesanded (URL-id, mille kraapimisintervall on möödunud). See toimib iseseisva protsessina, mis teostab kraapimistsükleid vastavalt ajakavale.
- **Parser:** Käivitatakse eraldiseisva Pythoni skriptina. See skript on mõeldud perioodiliseks käivitamiseks, et töödelda S3-e (või lokaalsesse *RAW_DATA_DIR* kausta)

kogunenud uusi HTML-faile. Parseri *process_all_raw_data* funktsioon itereerib läbi kõikide vastavas kaustas olevate domeenide alamkaustade ja töötleb neis leiduvaid HTML-faile.

Vigade logimine toimub mõlemas komponendis Pythoni standardse *logging* mooduli abil. Logid kirjutatakse standardväljundisse ning sisaldavad informatsiooni protsessi kulgemise, töödeldavate failide, tekkinud vigade ja andmebaasioperatsioonide edukuse kohta. Andmete uuendamise tsükkel on tagatud scraping_schedule tabeli ja Scraperi loogika kaudu, mis kontrollib *scraping_interval* välja, ning Parseri perioodilise käivitamisega, mis töötleb uusi kogutud faile.

5.3 Saavutatud tulemused

Käesoleva bakalaureusetöö tulemusena valmis täisfunktsionaalne automatiseeritud süsteem avaliku sektori veebilehtedelt andmete kogumiseks, töötlemiseks ja struktureeritud kujul salvestamiseks. Loodud lahendus hõlmab kõiki peatükis 4.2 kirjeldatud komponente alates kraapimise ajakava haldamisest kuni parsimistulemuste salvestamiseni andmebaasi. Lisaks tehnilisele teostusele koostati ka süsteemi kasutamisjuhend, mis aitab lahendust edaspidi hallata ja arendada.

Süsteemi testimise faasis keskenduti selle võimekuse valideerimisele erinevat tüüpi veebilehtedega. Kraapimist ja parsimist katsetati mitmesugustel Eesti avaliku sektori veebisaitidel, sealhulgas nii kaasaegsetel JavaScript-põhistel lehtedel (nt Reacti kasutavad) kui ka traditsioonilisematel staatilistel HTML-lehtedel. Näidetena testitud ja edukalt töödeldud allikatest võib tuua Maksu- ja Tolliameti (emta.ee), Kaitseliidu (kaitseliit.ee), Riigikogu (riigikogu.ee), Terviseportaali (terviseportaal.ee) ja Transpordiameti (transpordiamet.ee) veebilehed.

Kuna töö peamine fookus oli süsteemi tehnilise lahenduse väljatöötamine ja funktsionaalsuse tagamine, siis testimise käigus kogutud andmeid ei salvestatud pikaajaliselt edasiseks laiaulatuslikuks kasutamiseks Bürokrati treenimisel. Andmekogumine toimus peamiselt süsteemi komponentide testimise ja valideerimise eesmärgil. Seetõttu ei esitata siinkohal kvantitatiivset statistikat kogutud andmemahtude või unikaalsete JSON-kirjete arvu kohta, kuna need ei peegeldaks süsteemi potentsiaalset tootlusvõimekust, vaid pigem testimise

ulatust. Siiski on oluline märkida, et süsteem on võimeline genereerima struktureeritud JSON-andmeid vastavalt parseris defineeritud loogikale.

6 Tulemuste hindamine ja arutelu

Kogutud ja töödeldud andmete kvaliteedi ning loodud süsteemi efektiivsuse hindamine oli oluline osa käesolevast tööst. Kuna projekti üheks peamiseks eesmärgiks oli luua andmestik Bürokrati keelemudeli jaoks, toimus hindamine tihedas koostöös tellija, Riigi Infosüsteemi Ametiga (RIA).

6.1 Hindamismeetodid

Andmete kvaliteeti ja sobivust hinnati peamiselt läbi pideva valideerimisprotsessi RIA ekspertide poolt. See hõlmas regulaarseid ülevaatusi, kus kontrolliti:

- **Kogutud andmete relevantsust** (kas need pärinevad õigetest allikatest ja sisaldavad avaliku sektori teavet).
- **Andmete täielikkust** (kas oluline informatsioon on lehtedelt kätte saadud).
- **Andmete puhtust** (kas ebavajalik info, nagu HTML-mürä, navigeerimiselemendid, on eemaldatud).
- **Struktureeritud andmete (JSON) korrektsust** ja vastavust eeldatavale formaadile.
- **Automatiseeritud protsessi töökindlust** ja vigade logimist.

Lisaks koodi ja andmete ülevaatusele hinnati ka loodud dokumentatsiooni selgust ja piisavust edasiseks kasutamiseks.

6.2 Hindamise tulemused ja piirangud

Pidev koostöö ja valideerimine RIAga tagasid, et loodud andmekogumis- ja töötlemissüsteem vastab tellija ootustele ja vajadustele. Süsteem suutis edukalt koguda andmeid erinevatest allikatest ja formaatidest (HTML, PDF, DOCX) ning neid struktureerida JSON-formaati.

Olulise piiranguna tuleb aga välja tuua, et kogutud andmete reaalset mõju ja sobivust

Bürokrati keelemudeli treenimisel ei olnud võimalik töö raames praktiliselt testida. Selle põhjuseks oli asjaolu, et Bürokrati enda arendus ei olnud töö teostamise ajal veel jõudnud faasi, kus uute andmetega treenimist oleks saanud läbi viia. Seega põhineb hinnang andmete sobivusele peamiselt ekspertide hinnangul ja teoreetilisel vastavusel keelemudelite treenimise headele tavadele.

6.3 Ilmnenud väljakutsed

Projekti teostamisel kerkis esile mitmeid väljakutseid

- **Iseseisev õppimine:** Kuna tegemist oli uudse valdkonnaga ja spetsiifiliste tehnoloogiatega, nõudis projekt autorilt märkimisväärset iseseisvat õppimist ja suure hulga tehnilise dokumentatsiooni läbitöötamist (nt Scrapy, BeautifulSoup, Playwright dokumentatsioonid).
- **Tellija nõuded:** Tulemuse saamine täpselt sellisel kujul, nagu tellija (RIA) seda soovis, nõudis pidevat suhtlust, täpsustamist ja iteratiivset arendust.
- **Teekide valik ja kasutamine**: Sobivate Pythoni teekide leidmine erinevateks ülesanneteks (nt PDF/DOCX parsimine, S3 haldamine, andmebaasiühendused) ning nende korrektne integreerimine ja tööle saamine osutus kohati keerukaks.
- **Veebilehtede mitmekesisus:** Avaliku sektori veebilehed on ülesehituselt ja tehnoloogiliselt väga erinevad, mis muutis universaalse kraapimis- ja parsimisloogika loomise keeruliseks. Mõned lehed nõudsid Playwright'i kasutamist dünaamilise sisu tõttu, mis aeglustas protsessi.
- **Sisu dünaamilisus ja ajakohasuse tagamine:** Avaliku sektori informatsioon (seadused, määrused, teenuste kirjeldused, kontaktandmed) on pidevas muutumises. See seab kõrged nõudmised loodud süsteemi võimele andmeid regulaarselt uuendada ja tagada nende ajakohasus. Kraapimisintervallide seadistamine ja muudatuste tuvastamine on pidev ülesanne.
- **Irrelevantse info eraldamine (müra eemaldamine):** Avalikud veebilehed sisaldavad sageli palju informatsiooni, mis ei ole keelemudeli treenimiseks otseselt kasulik või isegi segav (nt uudisvood, reklaamid, navigeerimismenüüd, jalused, küpsiste teavitused). Nende elementide korrektne tuvastamine ja eemaldamine parsimise käigus oli oluline, et tagada treeningandmete kvaliteet. Käesolevas töös keskenduti

vales keeles ja uudiseid sisaldavate veebilehtede välistamisele juba kraapimise tasemel URL-i mustrite alusel. Samuti välistati XML-failid ja pildid kui mittetekstuaalne sisu.

6.4 Võimalikud edasiarendused

Loodud süsteem pakub tugevat alust, kuid seda on võimalik mitmel viisil edasi arendada:

- **Praktiline testimine:** Kõige olulisem edasine samm on kogutud andmete kasutamine Bürokrati keelemudeli treenimisel ja tulemuste põhjal süsteemi täiendamine.
- **Vektorandmebaasi ja RAG-mudeli implementeerimine:** Et Bürokratt saaks kogutud informatsiooni põhjal efektiivselt ja täpselt vastuseid genereerida, võiks kaaluda spetsiaalse vektorandmebaasi loomist käesoleva projekti raames kogutud andmete jaoks. Sellesse andmebaasi salvestataks tekstilõikude manused (embeddings). Seejärel saaks implementeerida RAG mudeli, mis võimaldaks Bürokrati keelemudelil päringu saamisel esmalt otsida relevantsed tekstilõigud vektorandmebaasist ja seejärel kasutada neid kontekstina vastuse genereerimisel. Kuigi RIA-l on olemas üldine lahendus teisest projektist, võiks spetsiaalselt selle andmekogu jaoks loodud vektorandmebaas ja RAG-integratsioon pakkuda täiendavat paindlikkust ja optimeerimisvõimalusi.
- **Skaleeritavuse optimeerimine:** Analüüsida ja optimeerida süsteemi jõudlust veelgi suuremate andmemahtude ja allikate arvu korral.
- **Asünkroonsuse rakendamine alg-URL-ide tasemel:** Praeguses lahenduses toimub erinevate alg-URL-ide (domeenide) andmete kogumine Scraperi poolt sünkroonselt, st üks domeen töödeldakse enne järgmise juurde asumist. Kuigi URL-isiseste alamlehtede kraapimine võib toimuda teatud määral asünkroonselt tänu Scrapy enda arhitektuurile, võiks süsteemi üldist jõudlust ja läbilaskvust oluliselt parandada, implementeerides asünkroonse lähenemise ka erinevate alg-URL-ide samaaegseks töötlemiseks. See võimaldaks paremini ära kasutada ressursse ja lühendada kogu andmekogumistsükli aega, eriti suure hulga erinevate allikate puhul.

7 Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli luua automatiseeritud süsteem avaliku sektori andmete kogumiseks ja töötlemiseks suure keelemudeli Bürokratt tarbeks. Töö käigus analüüsiti avaliku sektori andmeallikaid, valiti sobivad tehnoloogilised vahendid (Scrapy, BeautifulSoup, Playwright) ning töötati välja mitmeetapiline töövoog andmete kraapimiseks, puhastamiseks, struktureerimiseks (JSON) ja ajakohasena hoidmiseks.

Töö peamise tulemusena valmis funktsioneeriv prototüüp automatiseeritud andmekogumis- ja töötlemissüsteemist, mis suudab käsitleda erinevaid andmeformaate (HTML, PDF, DOCX) ja salvestada keelemudelile sobivaid struktureeritud andmeid. Loodi ka süsteemi kasutamist ja edasiarendamist võimaldav dokumentatsioon. Püstitatud eesmärgid – koguda andmeid, rakendada efektiivset metoodikat, struktureerida andmed JSON-formaadis, luua automatiseerimisskript ja dokumentatsioon – said täidetud.

Kuigi andmete reaalset mõju Bürokrati treenimisele ei olnud võimalik töö raames testida, valideeriti andmete kvaliteeti ja süsteemi toimivust edukalt koostöös Riigi Infosüsteemi Ametiga. Töö käigus ilmnenud väljakutsed, nagu iseseisev õppimine ja tehniliste lahenduste leidmine, pakkusid väärtuslikke kogemusi.

Loodud süsteem annab olulise panuse Bürokrati arendamisse, pakkudes mehhanismi vajalike treeningandmete efektiivseks hankimiseks. See omakorda aitab kaasa Eesti avalike teenuste kvaliteedi ja kättesaadavuse parandamisele tulevikus. Edasised sammud peaksid keskenduma süsteemi testimisele reaalse keelemudeli treeningu kontekstis ning võimalikele täiendustele andmekvaliteedi ja skaleeritavuse osas.

Kasutatud kirjandus

[1] Kratid. *Bürokratt*. Kasutatud: 02-02-2025. URL: https://www.kratid.ee/burokratt.

[2] Kratid. *Bürokrati tutvustus*. Kasutatud: 02-02-2025. URL: https://www.kratid.ee/burokrati-tutvustus.

[3] A. Avirzayev. *Mastering the Art of Web Scraping: Best Practices and Techniques*. Kasutatud: 08-02-2025. URL: https://www.medium.com/@avirzayev/mastering-the-art-of-web-scraping-best-practices-and-techniques-51e9f0e3cb33.

[4] Python Software Foundation. *The Python 3 Documentation*. Kasutatud: 12-02-2025. URL: https://www.docs.python.org/3/.

[5] PostgreSQL. *Documentation*. Kasutatud: 15-03-2025. URL: https://www.postgresql.org/docs/.

[6] Scrapy. *Scrapy 2.11 documentation*. Kasutatud: 12-02-2025. URL: https://www.docs.scrapy.org/en/latest/.

[7] Leonard Richardson. *Beautiful Soup Documentation*. Kasutatud: 12-02-2025. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[8] Playwright. *Playwright for Python Documentation*. Kasutatud: 16-03-2025. URL: https://www.playwright.dev/python/.

[9] ScrapingBee. *Playwright for Python: A Guide to Web Scraping*. Kasutatud: 16-03-2025. URL: https://www.scrapingbee.com/blog/playwright-for-python-web-scraping/.

[10] Oxylabs. *Scrapy vs. BeautifulSoup: Which One to Choose for Web Scraping?* Kasutatud: 11-02-2025. URL: https://www.oxylabs.io/blog/scrapy-vs-beautifulsoup.

[11] Amazon Web Services. *Amazon S3 Documentation*. Kasutatud: 02-04-2025. URL: https://www.docs.aws.amazon.com/s3/.

[12] Python Software Foundation. *base64 — Base16, Base32, Base64, Base85 data encodings*. Kasutatud: 10-04-2025. URL: https://www.docs.python.org/3/library/base64.html.

[13] Mozilla Developer Network. *Glossary: Base64*. Kasutatud: 10-04-2025. URL: https://www.developer.mozilla.org/en-US/docs/Glossary/Base64.

[14] Python Packaging Authority. *pip lock*. Kasutatud: 17-02-2025. URL: https://pip.pypa.io/en/stable/cli/pip_lock/.

[15] ScrapeOps. *The Python Scrapy Playbook - JavaScript Rendering Guide*. Kasutatud: 12-02-2025. URL: https://www.scrapeops.io/python-scrapy-playbook/scrapy-javascript-rendering-guide/.

[16] ISPRAS. *scrapy-puppeteer: Scrapy middleware to render pages using Puppeteer. GitHub hoidla.* Kasutatud: 16-03-2025. URL: https://www.github.com/ispras/scrapy-puppeteer.

[17] Docker. *Documentation*. Kasutatud: 25-03-2025. URL: https://www.docs.docker.com/.

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

Meie, Artjom Vysotski ja Artur Volnikov

1. Anname Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "Avaliku sektori andmete kogumine ja töötlemine suure keelemudeli tarbeks", mille juhendaja on Evelin Halling
    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. Oleme teadlikud, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autoritele.
3. Kinnitame, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

19.05.2025

[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 – Tööriistade valiku põhjendus

Tööriist	**Plussid**	**Miinused**	**Valiku põhjendus antud töös**
Playwright	+ Väga hea JavaScripti renderdamise võimekus + Toetab mitmeid brausereid (Chromium, Firefox, WebKit) + Hea asünkroonne tugi	- Aeglasem kui Scrapy/BeautifulSoup staatiliste lehtede puhul - Suur ressursinõudlus - Nõuab eraldi brauserite paigaldamist ja haldamist	Valitud keerukate, JavaScripti-rikaste lehtede jaoks, kus Scrapy-st üksi ei piisa. Hea integreeritavus Pythoniga. Kooskõlas tellija soovitustega.
Selenium	+ Pikaajaline ja laialt levinud tööriist veebiautomatiseerimiseks ja testimiseks	- Üldiselt peetakse Playwright'ist aeglasemaks ja ressursinõudlikumaks	Jäeti kõrvale peamiselt suurema ressursinõudluse ja aegluse tõttu, mis ei sobi suurte andmemahtude süstemaatiliseks kogumiseks antud projekti kontekstis.
Scrapy	+ Väga kiire ja efektiivne staatiliste ja pool-dünaamiliste lehtede kraapimiseks	- Nõrgem sisseehitatud JavaScripti renderdamise võimekus ilma lisateekideta - Vajab kombineerimist teiste tööriistadega (nt BeautifulSoup parsimiseks, Playwright JS renderdamiseks)	Valitud peamiseks kraapimisraamistikuks tänu kiirusele, efektiivsusele ja heale mastaapsusele suurte veebisaitide puhul.
BeautifulSoup	+ Väga paindlik ja lihtne HTML/XML parsimiseks	- Ei ole iseseisev kraapimistööriist (vajab midagi HTTP päringute tegemiseks, nt requests või Scrapy)	Valitud HTML-i parsimiseks tänu oma lihtsusele ja võimekusele ka vigase HTML-iga toime tulla. Hea koostöö Scrapyga.

Tööriist	**Plussid**	**Miinused**	**Valiku põhjendus antud töös**
Python Requests	+ Hea dokumentatsioon ja suur kasutajaskond + Väga lihtne ja kergekaaluline HTTP päringute tegemiseks	- Ei suuda iseseisvalt JavaScripti renderdada - Ei suuda JavaScripti renderdada ega parsida HTML-i	Liiga lihtne antud projekti keerukuse jaoks, kus oli vaja hallata linkide järgimist, sessioone ja dünaamilist sisu.