Rīgas 64. vidusskola

Scrapy un *BeautifulSoup* rasmošanas bibliotēku efektivitātes analīze

Zinātniski pētnieciskais darbs datorzinātņu un informātikas sekcijā

Darba autors:

Rīgas 64. vidusskolas 12. INZ klases skolniece Daiga Leitāne

Darba vadītājs:

Rīgas 64. vidusskolas programmēšanas skolotājs Edvards Bukovskis

Rīga, 2024

Anotācija

Zinātniski pētniecisko darbu "*Scrapy* un *BeautifulSoup* rasmošanas bibliotēku efektivitātes analīze" izstrādāja Rīgas 64.vidusskolas 12.INZ klases skolniece Daiga Leitāne, darba vadītājs – Rīgas 64. vidusskolas programmēšanas skolotājs Edvards Bukovskis.

Teorētiskajā daļa tika izpētīta rasmošanas definīcija, tās pielietojums, pamatprincipi un metodes. Tika apskatīts rasmošanas ietvars *Scrapy* un rasmošanas bibliotēka *BeautifulSoup*, kā arī šie rasmošanas rīki tika salīdzināti.

Praktiskajā daļā, balstoties uz autora izvirzītajiem kritērijiem, tika salīdzinātas rasmošanas bibliotēkas *Scrapy* un *BeautifulSoup*, pēc datu ieguves tika veikta mērījumu salīdzināšana.

Atslēgvārdi: rasmošana, tīmekļa vietne, bibliotēka, *Scrapy*, *BeautifulSoup*.

Abstract

Scientific research paper "Effectiveness analysis of web scraping libraries *Scrapy* and *BeautifulSoup*" was developed by Daiga Leitāne, a student of class 12.INZ ar Riga 64th Secondary School, supervised by Edvards Bukovskis, the programming teacher ar Riga 64th Secondary School.

The theoretical part dvelved into the definition of web scraping, its application, basic principles and methods. The web scraping framework *Scrapy* and the scraping library *BeautifulSoup* were reviewed and compared.

In the practical part, based on the criteria set by the autor, the scraping libraries *Scrapy* and *BeautifulSoup* were compared, and after data collection, a comparison of measurments was performed.

Keywords: web scraping, website, library, *Scrapy*, *BeautifulSoup*

Saturs

Ievads

Mūsdienās profesijas, kuras ir saistītas ar IT (informācijas tehnoloģijas) kļūst arvien aktuālākas tādēļ, ka šī sfēra pēdējo gadu laikā ir novērojami attīstījusies. Šo nozaru speciālistu darba uzdevumi bieži vien sevī ietver lielu datu daudzuma ievākšanu. Šī procesa veikšanai tiek izmantotas vairākas metodes, kā, piemēram, lietiskais internets, publiski pieejamie dati, sociālo mediju analīzes rīki un rasmošana. Piemērotas metodes izvēle ir ārkārtīgi svarīga, jo no tās ir atkarīgs cik veiksmīgi tiks veikts konkrētais uzdevums. Rasmošana ir viena no populārākajām izvēlēm, ja ir nepieciešams iegūt apjomīgu datu kopu, jo datu ievākšana izmantojot šo metodi ir efektīva un precīza. Šobrīd *Python* ir viena no populārākajām programmēšanas valodām, kas nozīmē to, ka tā tiek bieži izmantota datu iegūšanai un attiecīgi ir izveidots liels daudzums rasmošanas bibliotēku. Divas no visbiežāk izmantotajām bibliotēkām šai programmēšanas valodai ir *Scrapy* un *BeautifulSoup*. Ir svarīgi izvēlēties atbilstošo bibliotēku datu ievākšanas procesam, jo katrai no šīm bibliotēkām ir savas priekšrocības un trūkumi, kas var padarīt rasmošanu ne tik efektīvu kā iecerēts, ja tā tiek veikta uz dažāda rakstura mājaslapām.

Šajā darbā tiks salīdzinātas divas rasmošanas bibliotēkas, *Scrapy* un *BeautifulSoup*. Tiks analizēta to spēja iegūt datus no dažāda veida mājaslapām, izpētītas šo bibliotēku stiprās un vājās puses rasmošanas procesā saskaroties ar tīmekļa vietnēm ar atšķirīgu datu izkārtojumu un šķēršļiem, kurus būs jāpārvar, lai veiksmīgi piekļūtu autora izvēlētajiem datiem.

Darba mērķis: salīdzināt rasmošanas bibliotēkas Scrapy un BeautifulSoup, analizēt to darbību datu iegūšanā no dažāda tipa mājaslapām un izsecināt, kura bibliotēka ir visefektīvākā.

Hipotēze: izvēloties visefektīvāko bibliotēku ir iespējams optimizēt datu iegūšanu no dažāda veida mājaslapām.

Uzdevumi:

1. Izpētīt rasmošanas bibliotēkas, kuras ir piemērotas *Python* programmēšanas valodai;
2. Izpētīt, kā tiek veikta datu rasmošana no dažādu veidu mājaslapām;
3. Izstrādāt kritērijus, pēc kuriem tiks salīdzinātas bibliotēkas;
4. Salīdzināt *Scrapy* un *BeautifulSoup* bibliotēku darbību;
5. Analizēt, kura bibliotēka ir visefektīvākā rasmošanas veikšanai;
6. Veikt datu analīzi, apkopot rezultātus, pamatojoties uz tiem izteikt secinājumus.

Darbā izmantotās metodes:

1. Literatūras apskats;
2. Programmēšanas valodas *Python* pielietošana rasmošanas procesam praktiskajā daļā;
3. Rasmošanas bibliotēku darbības analīze;

Darba struktūra: darbs sastāv no ievada, 2 nodaļām, 8 apakšnodaļām, secinājumiem, izmantoto informācijas avotu saraksta un pielikuma. Darbā ir 17 attēli un 1 tabula.

1. Literatūras apskats

1.1.Rasmošanas pamati

Rasmošana ir process, kura laikā tiek ievākti dati no tīmekļa vietnēm. Vienlaicīgi informāciju ir iespējams iegūt no liela daudzuma mājaslapām, kas padara to par ļoti efektīvu un ātru metodi datu ievākšanai un uzglabāšanai. Par pirmo uz rasmošanu balstītu projektu tiek uzskatīta vispasaules tīmekļa izveide (angliski: World Wide Web) 1989. gadā. (Britannica, 2024) Šī interneta pārlūkprogramma meklēja visus pieejamos datus un apkopoja tos vienā vietā ērtai pārskatīšanai, kas arī kalpoja kā vienots tīkls datu apmaiņai starp visiem datoriem pasaulē. (Berners-Lee, 2023) Šīs pārlūkprogrammas izveide un pieejamo datu pieaugums iedvesmoja citus izstrādātājus veidot savas, jau attīstītākas un pilnveidotākas programmas. Par pašu veiksmīgāko pārlūkprogrammu tiek uzskatīts *Google Chrome*, kas meklē visu internetā pieejamo informāciju, apskata atsevišķas tīmekļa vietnes un apkopo tās lietotājiem pārskatāmā veidā, izmantojot indeksus pārlūkprogrammas apakšā. (Khder, 2021)

Ar rasmošanu mēs saskaramies katru dienu, bet nevis no izstrādātāja skatupunkta, bet tieši pretēji, no lietotāja. Tā tiek izmantota interneta pārlūkos un MI (mākslīgais intelekts), ko mūsdienu interneta lietotājs pielieto, lai ikdienā ātri atrastu nepieciešamo informāciju.

To pielieto vairākās sfērās, kuras balstās uz informācijas iegūšanu, bet it īpaši tā ir aktuāla komercizpētes jomā. (Khder, 2021) Šīs nozares pārstāvji savu darba uzvedumu ietvaros iegūst datus no speciāli atlasītām mājaslapām, lai novērotu tendences produktu cenā un atsauksmēs, kā arī, lai atrastu un piesaistītu potenciālos klientus.

Rasmošana tiek izmantota arī mašīnmācīšanās nozarē, kas ir mākslīgā intelekta apakšnozare, kur datorsistēmas tiek apmācītas efektīvāk apieties ar datiem. (Mahesh, 2020) Mūsdienās pieejamo datu apjoms ir ārkārtīgi liels un parastam cilvēkam nav iespējams apskatīt un izanalizēt visus pieejamos resursus, lai šo problēmu risinātu tiek izmantota mašīnmācīšanās. Datorsistēmas tiek apmācītas izmantojot lielu datu daudzumu, kas pārsvarā tiek iegūts rasmošanas procesā, jo ar rasmošanas palīdzību var izvēlēties specifiskus kritērijus pēc kuriem tiek meklēta nepieciešamā informācija. (Scrapfly, 2023)

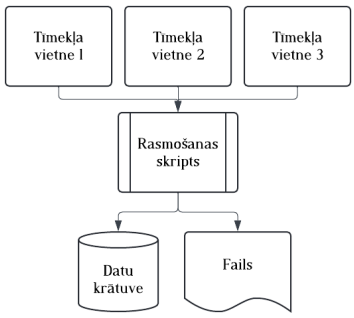
Rasmošana ir kļuvusi par nozīmīgu daļu lielākajai daļai mūsdienu sabiedrības, gan tiem, kuri izmanto to darbam, gan arī parastiem lietotājiem. To var sastapt visiem pieejamos resursos kā, piemēram, interneta pārlūkos un mākslīgajā intelektā, bez kuriem informācijas iegūšana parastiem interneta lietotājiem ir neiespējama. Rasmošana piedāvā risinājumu liela informācijas daudzuma analizēšanai, jo to dara nevis cilvēks, kurš pārskata visu manuāli, bet gan programma, kas to dara ātrāk, efektīvāk un tieši atbilstoši kritērijiem, kuri tika pielāgoti specifiska uzdevuma izpildei.

1.2.Datu iegūšana

Kā jau iepriekš tika minēts, rasmošana ir neatņemama daļa liela datu daudzuma iegūšanas procesā. Šajā nodaļā tiks dziļāk apskatīti rasmošanas pamatprincipi un metodes. Rasmošanas mērķis ir padarīt datu iegūšanu par efektīvu, ņemot vērā pieejamās informācijas daudzumu, kura ir atrodama internetā. Lai veiksmīgi iegūtu datus rasmojot ir jāiepazīstas ar šī procesa pamatprincipiem: jāizprot kā tiek veikta datu iegūšana, jāizvēlas atbilstošā metode un jāapstrādā iegūtie dati.

1.2.1. Rasmošanas pamatprincipi

Rasmošanas process sevī ietver 3 galvenos posmus: piekļuve tīmekļa vietnei (fetching stage), datu iegūšana (extraction stage), kas ir rasmošanas skripta rakstīšana, un datu transformācija (transformation stage). (Khder, 2021)



(1. attēls: Rasmošanas pamatprincipa shēma (ResearchGate, 2020))

Piekļuve tīmekļa vietnei: lai piekļūtu tīmekļa vietnei ir nepieciešams HTTP (HyperText Transfer Protocol), kas tiek izmantots kā komunikācija starp lietotāju un izvēlēto vietni, tīmekļa serveriem tiek pieprasīta piekļuve tīmekļa vietnēm, kuru saturs tiek apstrādāts un tiek iegūti vietnes HTML dati. (Webscraping.ai, 2024) (Khder, 2021)

Datu iegūšana: pēc vietnes HTML datu saņemšanas tiek pielietotas rasmošanas bibliotēkas kā, piemēram, *BeautifulSoup*, kas spēj pārskatīt HTML koda struktūru, atrodot un izceļot nepieciešamos datus. (Khder, 2021) Šī posma galvenais uzdevums ir rasmošanai izvēlēto datu izolācija no citiem, atdalot pārējās vietnes HTML koda daļas.

Datu transformācija: iegūtie dati tiek pārveidoti strukturētā formātā, kas ir standarta sintakse datu organizēšanai sistemātiskā veidā, un tiek pārvietoti datu krātuvē vai lietotāja atlasītajā failā. (Khder, 2021)

1.2.2. Rasmošanas metodes

Mūsdienās pieejamās tīmekļa vietnes pēc uzbūves ir sarežģītākas nekā tad, kad to izmantošana tikai sāka gūt popularitāti. Statiskās tīmekļa vietnes, kuras vienmēr attēlo vienu un to pašu saturu, vairs nav tik bieži izmantotas, to vietā stājas dinamiskās tīmekļa vietnes, tas ir, vietnes, kuras mijiedarbojas ar tās lietotāju atkarībā no tā darbībām un vajadzībām, rādot to saturu, kurš piesaista konkrēto vietnes lietotāju. (Lenovo, 2022) Šobrīd gandrīz jebkura tīmekļa vietne attēlo reklāmas, dažas pat pieprasa lietotājam reģistrēties pirms tiek dota iespēja aplūkot tajā attēloto informāciju. Sakarā ar šīm izmaiņām tīmekļa vietņu raksturā ir svarīgi tās izanalizēt, lai saprastu, kura no rasmošanas metodēm būs vispiemērotākā.

Kopēšana un ievietošana (copy and paste): šī metode visbiežāk tiek izmantota, ja ir nepieciešams iegūt nelielu datu kopu, jo tā prasa lietotāju veikt vienu un to pašu darbību atkārtoti, tas ir, iezīmēt nepieciešamo informāciju un ievietot to tai paredzētajā vietā, kamēr visi nepieciešamie dati ir ievākti. Šī metode nav piemērota lielas datu kopas ievākšanai, jo rasmošanas process būs pārlieku garš un sarežģīts. (Khder, 2021)

HTML parsēšana (HTML parsing): tiek analizēts HTML kods, kas piedod tīmekļa vietnei struktūru un attēlo nepieciešamo informāciju, lai tiktu noteikta tā struktūra un iegūti

nepieciešamie dati. Šīs analīzes laikā HTML kods tiek "sajaukts" līdz tā pamatelementiem, kā, piemēram, tagi, atribūti un teksta saturs. Tad izmantojot kādu no programmēšanas valodām, visbiežāk *Python*, un atbilstošu bibliotēku tiek iegūti nepieciešamie dati. (DataHen, 2024)

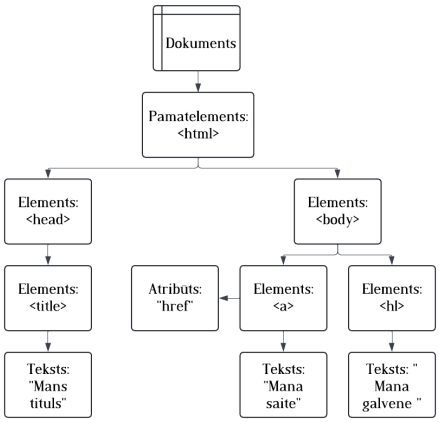
Rasmošanas programmas: šīs programmas spēj automātiski noteikt un analizēt tīmekļa vietnes struktūru, kas novērš vajadzību lietotājam manuāli pārskatīt vietnes HTML saturu. (Khder, 2021) Pēc rasmošanas, programma pārskatāmi saglabā un organizē datus. (Simplilearn, 2024) Šīs programmas ir visefektīvākās rasmošanai, jo tās visu dara automātiski, bez cilvēka iejaukšanās vajadzības.

1.3.*Scrapy* un *BeautifulSoup* bibliotēku salīdzināšana

Rasmošanai visbiežāk tiek izmantotas attiecīgās bibliotēkas izvēlētajai programmēšanas valodai, jo ar to palīdzību datu iegūšanas process ir efektīvāks nekā to manuālā datu atlasīšana. *Scrapy* un *BeautifulSoup* ir divas no visbiežāk izmantotajām rasmošanas bibliotēkām *Python* programmēšanas valodai, taču neskatoties, ka to galvenais uzdevums ir viens, tas ir, iegūt datus no tīmekļa vietnēm, katrai no tām ir savas darbības īpatnības.

***Scrapy*:** ir bezmaksas atvērtā pirmkoda rasmošanas ietvars, kas ir uzrakstīts *Python* programmēšanas valodai. *Scrapy* ir integrēta sistēma, kas sevī ietver pieprasījumu saņēmēju, lejupielādētāju tīmekļa vietņu iegūšanai un klases, kuras sarakstīja šīs bibliotēkas lietotāji, lai parsētu iegūtās atbildes un iegūtu nepieciešamos vienumus. Klases, kuras arī sauc par zirnekļiem (*spiders*), tiek izmantotas lai definētu veidu, ar kuru dati tiks iegūti no tīmekļa vietnes. (Myers & McGuffee, 2015)

***BeautifulSoup*:** ir *Python* rasmošanas bibliotēka, kas ir spējīga iegūt datus no HTML un XML failiem. (Pluralsight, 2020) *BeautifulSoup* izveido parsēšanas koku (parsing tree) analizētajām lapām, ko var izmantot datu iegūšanā no HTML failiem. (Crummy, 2016) Šī bibliotēka izmanto tagus, teksta saturu un atribūtus kā galvenos meklēšanas kritērijus. (LangChain, 2023)



(2.attēls: HTML parsēšanas koka struktūra (W3Schools))

Galvenā šo bibliotēku atšķirība - *Scrapy* ir rasmošanas ietvars, bet *BeautifulSoup* ir parsēšanas bibliotēka. *Scrapy* piedāvā iespēju definēt tīmekļa vietnes URL (vienotais resursu vietrādis) izmantojot papildus parametrus un izpildīt rasmošanu pilnībā, tas ir, iegūt un lejupielādēt tīmekļa vietņu saturu, kamēr *BeautifulSoup* vienkārši iegūs prasītos datus. (Oxylabs, 2023)

2. Praktiskā daļa

Praktiskajā daļā autors salīdzinās *Scrapy* un *BeautifulSoup* rasmošanas bibliotēkas.

Teorētiskajā daļā autors apskatīja rasmošanas bibliotēkas *Scrapy* un *BeautifulSoup*. Praktiskajā daļā autors veica šo bibliotēku salīdzināšanu pēc sekojošiem kritērijiem: laiks, procesora iekšējā temperatūra, koda garums, CPU un RAM lietojums. Autors salīdzināja bibliotēkas izmantojot 5 dažāda veida tīmekļa vietnes. Lai izprastu autora bibliotēku salīdzinājumu ir jāizprot izvēlēto kritēriju nozīme, tas ir, kas tieši tiek mērīts

Laiks ir viens no galvenajiem vērtēšanas kritērijiem, jo no tā ir atkarīgs uzdevuma veikšanas ilgums, kas attiecīgi ietekmē rasmošanas procesa efektivitāti. Laiks tiek mērīts no programmas palaišanas brīža līdz tam momentam, kad visi nepieciešamie dati ir iegūti un gatavi pārskatīšanai. Laiks tiks mērīts sekundēs (s).

Procesora iekšējā temperatūra ir parametrs, kas ļauj saprast vai CPU darbība iekļaujas normas robežās. Vidējā temperatūra aktīvi nelietojot ierīci ir 30°- 40°, tomēr tā var paaugstināties līdz 60° - 70° aktīvas izmantošanas laikā. Uzturēt zemu iekšējo temperatūru it būtiski, jo tā nodrošina optimālu datora darbību. Dati par procesora iekšējo temperatūru tiks ievākti izmantojot programmu "*Core Temp*". Temperatūra tiks mērīta grādos pēc Celsija (C°).

Koda garums ir parametrs, kas ietekmē laiku, kas tiek pavadīts rakstot programmu, kā arī koda pārskatāmību. Īsāks kods visbiežāk nozīmē to, ka tas ir vieglāk saprotams un kļūdu pamanīšana ir veiksmīgāka. Koda garums tiks vērtēts saskaitot visas programmā ietilpstošās koda rindiņas neieskaitot atstarpes un citus nepieciešamos atdalījumus.

CPU lietojums ir parametrs, kas parāda to, kāda daļa no pieejamās datora jaudas ir izmantota konkrētā laika brīdī. CPU tiek attēlots procentos, kas attiecīgi norāda uz izmantotās jaudas daudzumu. 0% norāda uz to, ka programma vai aplikācija neizmanto jaudu, kamēr 100% norāda, ka uz kāda konkrēta procesa veikšanu tiek izmantota visa pieejamā datora jauda. Uzturēt zemu CPU ir svarīgi, jo augsts CPU spēj palēlināt datora darbību. Šo parametru ietekmē koda garums. Ir svarīgi optimizēt uzrakstīto kodu, lai uzturētu zemu CPU rādītāju, kas savukārt ietekmēs rasmošanas laiku. CPU lietojums tiks mērīts procentos (%).

RAM lietojums tiek ietekmēts balstoties uz CPU. RAM ir datora īstermiņa atmiņa, kas uztur sevī nepieciešamos datus konkrēta procesa veikšanai. Rasmojot tiek lejupielādēti tīmekļa vietņu dati, kuri satur informāciju, kas uzglabājas RAM. Tāpat kā CPU, uzturot zemu RAM rasmošana būs ātrāka. RAM tiks mērīts megabaitos (MB).

2.1.Datu kopas izvēle

Lai efektīvi salīdzinātu *Scrapy* un *BeautifulSoup* ir nepieciešama daudzveidīga datu kopa. Autors izvēlējās 5 dažāda veida tīmekļa vietnes, ar kuru palīdzību tiks salīdzināta šo bibliotēku darbība un efektivitāte. Datu kopā ietilpst tīmekļa vietnes, kas ir veidotas izmantojot *Java* programmēšanas valodu, statiskās HTML vietnes, elektroniskās tirdzniecības vietnes, vietnes ar autentifikāciju, kā arī vietnes, kas satur dažāda formāta datus.

Tīmekļa vietnes, kas veidotas izmantojot *Java* programmēšanas valodu parasti ir dinamiskas mājaslapas, kuru saturs mainās atkarībā no lietotāja izvēlēm un darbībām. Sakarā ar to, ka *Scrapy* un *BeautifulSoup* nevar automātiski iegūt *JavaSritpt* tipa saturu, tiks pārbaudīta to spēja rasmot dinamiska formāta datus, kā arī citas nepieciešamās darbības un soļi *JavaScript* datu

iegūšanai. Autors izvēlējās ziņu tīmekļa vietni "*The Guardian*" (https://www.theguardian.com/europe).

Statiskās HTML vietnes ir vietnes, kuru saturs paliek nemainīgs neskatoties uz lietotāju darbībām. Šādas tīmekļa vietnes ir vispiemērotākās tiešai autora izvēlēto bibliotēku salīdzināšanai, jo tajās nav papildelementu, kas apgrūtinātu rasmošanu. Rezultātā iegūstot precīzu bibliotēku darbības attēlu. Autors izvēlējās elektroniskās bibliotēkas "*Project Gutenberg*" sadaļu ar populārākajām grāmatām (https://www.gutenberg.org/ebooks/search/?sort_order=downloads).

Elektroniskās tirdzniecības vietnes satur lielu datu apjomu, kas sevī ietver gan tekstu, gan attēlus. Šo vietņu pamatā parasti ir HTML kods, tomēr tas ir sazarots un grūti pārskatāms, kas pārbaudīs *Scrapy* un *BeautifulSoup* bibliotēku spēju apieties ar sarežģītu HTML koda struktūru. Autors izvēlējās elektroniskās tirdzniecības vietnes "*eBay"* televizoru nodaļu (https://www.ebay.com/b/TVs/11071/bn_738302?_pgn=).

Vietnes ar autentifikāciju ir statistiskās vai dinamiskās tīmekļa vietnes ar papildus drošības līmeni, tomēr tas padara piekļuvi vietnes datiem grūtāku. Lai piekļūtu vietnes saturam vispirms ir jāievada lietotājvārds un parole, un tikai tad būs dota atļauja un pieeja skatīt vietnes aizsargātos datus. Autors izvēlējās profesionālo sociālo tīklu "*Linkedin*" sadaļu ar notikumiem (https://www.linkedin.com/events/)

Vietnes ar dažāda formāta datiem ir tīmekļa vietnes, kas sevī ietver tabulas, attēlus, tekstu un citus datu attēlojuma veidus vienas lapas ietvaros. Rasmojot datus no šādām tīmekļa vietnēm tiks pārbaudīta *Scrapy* un *BeautifulSoup* spēja veikt atbilstošās darbības un apstrādāt dažāda veida datu formātus. Autors izvēlējās enciklopēdijas "*Wikipedia*" šķirkli latviešu valodā par COVID-19 pandēmiju (https://lv.wikipedia.org/wiki/COVID-19_pand%C4%93mija).

2.2.*BeautifulSoup* bibliotēkas testēšana

Autors praktiskās daļas izveidē izmantoja *Visual Studio Code* programmēšanas vidi, kodu rakstot *Python* programmēšanas valodā. Visi *BeautifulSoup* kodi, kuri tika izveidoti darba ietvaros ir pieejami 1. pielikumā. *BeautifulSoup* bibliotēkas testēšana sevī ietver 5 dažādas autora uzrakstītās programmas, kas atbilst iepriekš izvirzītajiem uzdevumiem, tas ir, izvēlēto datu ieguve un to izvērtēšana izmantojot autora izvirzītos kritērijus.

1. Uz Java programmēšanas valodu balstīta tīmekļa vietne "*The Guardian*"

Autors no tīmekļa vietnes izvēlējās iegūt datus par ziņu virsrakstiem saistībā ar notikumiem Eiropā.

```
soup = BeautifulSoup(response.text, 'html.parser')
links = soup.find_all('a', attrs={'aria-label': True})
aria_labels = [link['aria-label'] for link in links]
```

(3. attēls: "*The Guardian*" rasmošana izmantojot *BeautifulSoup*)

Apskatot 3. attēlu var novērot kā tiek veikta virsrakstu iegūšana ar BeautifulSoup. "*BeautifulSoup(response.text, 'html.parser'*" izveido parsētu HTML dokumenta attēlu, kas nodrošina ērtu datu ieguvi. "*soup.find_all('a', attrs={'aria-label': True})*" pārskata iepriekš parsēto HTML dokumentu, meklējot visus "*<a>*" tagus, kas satur atribūtu "*aria-label*". "*[link['aria-label'] for link in links]*" izskata katru atrasto "*<a>*" tagu un iegūst atribūta "*aria-label*" vērtību.

2. Statiskā HTML tīmekļa vietne "Project Gutenberg"

Autors no tīmekļa vietnes izvēlējās iegūt grāmatu nosaukumus no visu grāmatu saraksta pirmajām 5 lapām.

```
soup = BeautifulSoup(response.text, 'html.parser')
titles = [title.text.strip() for title in soup.find_all('span', class_='title')]
```

(4. attēls: "*Project Gutenberg*" rasmošana izmantojot *BeautifulSoup*)

3. attēlā var novērot, kā *BeautifulSoup* tiek izmantots grāmatu nosaukumu iegūšanā. "*soup = BeautifulSoup(response.text, 'html.parser')*" parsē tīmekļa vietnes HTML saturu, lai tas atbilstu *BeautifulSoup* objekta nosacījumiem, kas padara HTML struktūru pārskatāmu un piemērotu datu iegūšānai. "*soup.find_all('span', class_='title')*" pārskata HTML dokumentu un meklē "**" tagus, kuri satur atribūtu "*title*". "*[title.text.strip() for title in soup.find_all('span', class_='title')]*" iegūst tekstu no katra "**" taga, attīra tekstu no liekiem tukšumiem un apkopo iegūto tekstu sarakstā.

3. Elektroniskās tirdzniecības vietne "eBay"

Autors no tīmekļa vietnes izvēlējās iegūt datus par televizoru nosaukumiem un cenām no pirmajām 5 lapām.

```
soup = BeautifulSoup(driver.page_source, 'html.parser')
tv_nosaukumi = soup.find_all('h3', class_='s-item__title')
tv_cenas = soup.find_all('span', class_='s-item__price')

for nosaukumi, cenas in zip(tv_nosaukumi, tv_cenas):
    tv_list.append({
        'nosaukimi': nosaukumi.get_text(strip=True),
        'cenas': cenas.get_text(strip=True)
    })
```

(5. attēls: "*eBay*" rasmošana izmantojot *BeautifulSoup*)

5. attēlā var novētot, kā *BeautifulSoup* tiek izmantots televizoru nosaukumu un cenu iegūšanā. "*soup = BeautifulSoup(driver.page_source, 'html.parser')*" pārveido vietnes HTML saturu (izmantojot *Selenium* bibliotēku – "*driver.page_source*") uz *BeautifulSoup* objektu, kas nodrošina efektīvāku datu parsēšanu un turpmāko iegūšanu. "*soup.find_all('h3', class_='s-item__title')*" atrod visus "*<h3>*" elementus, kas satur "*s-item_title*" klasi, kura sevī ietver televizoru nosaukumus, savukārt "*soup.find_all('span', class_='s-item__price')*" atrod visus "**" elementus, kas satur "*s-item_price*" klasi, kura sevī ietver televizoru cenas. Teksts tiek iegūts izmantojot "*title.get_text(strip=True)*" un "*price.get_text(strip=True)*", kas attiecīgi iegūst tekstveida informāciju par televizoru nosaukumiem un cenām, vienlaikus attīrot tos no tukšajiem lauciņiem, kuri tiek izmantoti teksta attēlošanai datoros. Dati tiek apkopoti sarakstā izmantojot "*{nosaukumi: ..., cenas: ...}*", kas apkopo noteiktas kategorijas tekstu vienviet un tie tiek saglabāti "*tv_list*", kas satur visu iepriekš iegūto informāciju.

4. Vietne ar autentifikāciju "Linkedin"

Autors no tīmekļa vietnes izvēlējās iegūt datus par notikumiem, kas tiek ieteikti apmeklēšanai balstoties uz autora personīgo profilu.

```
def iegut_events():
    soup = BeautifulSoup(driver.page_source, 'html.parser')
    events = soup.find_all("p", class_="events-components-shared-discovery-card__event-title link-without-visited-state t-black t-bold")
    print("\nNotikumi:")
    for event in events:
        print(f"- {event.get_text()}")
```

(6. attēls: "*Linkedin*" rasmošana izmantojot *BeautifulSoup*)

6. attēlā tiek attēlota notikumu nosaukumu iegūšana izmantojot *BeautifulSoup*. "*soup = BeautifulSoup(driver.page_source, 'html.parser')*" iegūst vietnes HTML saturu izmantojot *Selenium* bibliotēku – "*driver.page_source*" un parsē to izmantojot *BeautifulSoup*. "*events = soup.find_all("p", class_="events-components-shared-discovery-card__event-title link-without-visited-state t-black t-bold")*" atrod visus "*<p>*" elementus, kuri sevī ietver klases, kas satur informāciju par notikumiem. "*for event in events: print(f"- {event.get_text()}")*" tiek pārskatīts saraksts ar iepriekš iegūtiem elementiem un tiek iegūts teksts izmantojot "*.get_text()*", kas beigās tiek printēts *Visual Studio Code* programmas terminālī.

5. Vietne ar dažāda formāta datiem "*Wikipedia*"

Autors izvēlējās iegūt 2 dažāda formāta datus – tabula un parastais teksts. Tika izvēlēta tabula, kas attēlo COVID-19 statistiku, tas ir saslimšanas gadījumu un mirušo skaits pasaulē, kā arī rindkopa par Latviju saistībā ar šo pandēmiju.

```
soup = BeautifulSoup(response.text, 'html.parser')

tabula = soup.find('table', class_='wikitable')

if tabula:
    rows = tabula.find_all('tr')
    for row in rows:
        cols = row.find_all(['td', 'th'])
        cols = [col.get_text(strip=True) for col in cols]
        print("\t".join(cols))
```

(7. attēls: "*Wikipedia*" rasmošana izmantojot *BeautifulSoup* (1))

```
for heading in soup.find_all('div', class_='mw-heading mw-heading5'):
    if heading.find('h5', id='Latvija'):
        latvija = heading
    elif heading.find('h5', id='Igaunija'):
        igaunija = heading
```

(8. attēls: "*Wikipedia*" rasmošana izmantojot *BeautifulSoup* (2))

```
if latvija and igaunija:
    content = []
    current_section = latvija.find_next('h5')
    while current_section and current_section != igaunija:
        if current_section.name == 'h5':
            if 'Igaunija' in current_section.get_text():
                break
        elif current_section.name == 'p':
            content.append(current_section.get_text(strip=True))
        current_section = current_section.find_next()
```

(9. attēls: "*Wikipedia*" rasmošana izmantojot *BeautifulSoup* (3))

Apskatot 7., 8. un 9. attēlu var novērot kā *BeautifulSoup* tiek izmantots dažāda formāta datu iegūšanai. "*tabula = soup.find('table', class_='wikitable')*" atrod "*<table>*" elementu ar klasi "*wikitable*", kas ir specifiska tabulu klase "*Wikipedia*" tīmekļa vietnē balstīta uz tās CSS klasi. "*rindas = table.find_all('tr')*" tiek izmantots tabulas rindu atrašanai, izmantojot *<tr>* elementa meklēšanu. "*for rinda in rindas: ...*" (skatīt 7. attēlu) pārskata tabulas rindu saturu un atrod tekstu izmantojot meklēšanu pēc "*<td>*" un "*<th>*" elementiem. "*for nosaukums in soup.find_all('div', class_='mw-heading mw-heading5'): ...*" (skatīt 8. attēlu) pārskata vietnes HTML saturu un meklē "* *" elementus ar specifisku klasi, kas nodrošina "*<h5>*" elementu noteikšanu balstoties uz noteiktajiem identifikatoriem – "*Latvija*" un "*Igaunija*". "*if latvija and igaunija: ...* " (skatīt 9. attēlu) iegūst saturu no visiem "*<p>*" elementiem starp "*Latvija*" un "*Igaunija*" nodaļām, kas nodrošina, ka iegūtais teksts saturēs tikai atbilstošo informāciju par nodaļu "*Latvija*".

2.3.*Scrapy* bibliotēkas testēšana

Scrapy bibliotēkas testēšanas process ir līdzīgs *BeautifulSoup*, atšķirība starp rasmošanas bibliotēkām ir manāma koda struktūrā un izmantoto funkciju izvēlē. Visi *Scrapy* kodi, kuri tika izveidoti darba ietvaros ir pieejami 2. pielikumā. *Scrapy* bibliotēkas testēšana sevī ietver 5 dažādas autora uzrakstītās programmas, kas atbilst iepriekš izvirzītajiem uzdevumiem, tas ir, izvēlēto datu ieguve un to izvērtēšana izmantojot autora izvirzītos kritērijus.

1. Uz Java programmēšanas valodu balstīta tīmekļa vietne "*The Guardian*"

Autors no tīmekļa vietnes izvēlējās iegūt datus par ziņu virsrakstiem saistībā ar notikumiem Eiropā.

```
def parse(self, response):
    links = response.css('a[aria-label]')
    aria_labels = [link.attrib['aria-label'] for link in links]
```

(10. attēls: "*The Guardian*" rasmošana izmantojot *Scrapy* (1))

```
if __name__ == "__main__":
    process = CrawlerProcess()
    process.crawl(GuardianSpider)
    process.start()
```

(11. attēls: "*The Guardian*" rasmošana izmantojot *Scrapy* (2))

Aplūkojot 10. un 11. attēlu var novērot kā *Scrapy* tiek izmantots ziņu virsrakstu iegūšanai. Definētā "*parse*" metode (skatīt 10. attēlu) nodrošina, ka iepriekš iegūtais HTML saturs tiek pārskatīts specifisku elementu meklēšanai. "*response.css('a[aria-label]')*", atrod visus "*<a>*" elementus, kas satur "*aria-label*" atribūtu izmantojot CSS atlasītāju "*response.css()*". *"[link.attrib['aria-label'] for link in links]*" iegūst "*aria-label*" atribūta vērtības no iepriekš atlasītajiem elementiem, kas ir saglabāti mainīgajā "*links*". "*process.crawl(GuardianSpider)*" un "*process.start()*" aktivizē *Scrapy* un palaiž "*GuardianSpider*". Šis "zirneklis" iegūst vietnes URL, parsē atbildes izmantojot "*parse*" metodi un rasmo datus.

2. Statiskā HTML tīmekļa vietne "*Project Gutenberg*"

Autors no tīmekļa vietnes izvēlējās iegūt grāmatu nosaukumus no visu grāmatu saraksta pirmajām 5 lapām.

```
nosaukumi = response.css('span.title::text').getall()
print(f"Lapa nr.{self.page_count}:")
for nosaukums in nosaukumi:
    print(nosaukums)
```

(12. attēls: "*Project Gutenberg*" rasmošana izmantojot *Scrapy*(1))

```
yield scrapy.Request(url=next_page_url, callback=self.parse)
```

(13. attēls: "*Project Gutenberg*" rasmošana izmantojot *Scrapy*(2))

12. un 13. attēlā var novērot kā *Scrapy* tiek pielietots grāmatu nosaukumu iegūšanai. "*nosaukumi = response.css('span.title::text').getall()*" (skatīt 12. attēlu) tiek izmantota "*response.css*" metode visu CSS elementu atlasīšanai, kas satur klasi "*title*" un atrodas "**" tagos. "*: : text*" elements iegūst tekstveida datus no iepriekš atlasītajiem elementiem, un "*.getall()*" apkopo datus sarakstā. "*for nosaukums in nosaukumi: ...* " pārskata iegūtos datus un izmantojot "*print()*" funkciju tie tiek izvadtīti terminālī. "*yield scrapy.Request(url=next_page_url, callback=self.parse)*" (skatīt 13. attēlu) izveido jaunu pieprasījumu nākošajai lapai izmantojot "*next_page_url*", kas izsauc "*parse*" metodi no jauna un veic visu rasmošanas procesu (skatīt 12. attēlu), līdz dati it iegūti no pirmajām 5 lapām.

3. Elektroniskās tirdzniecības vietne "eBay"

Autors no tīmekļa vietnes izvēlējās iegūt datus par televizoru nosaukumiem un cenām no pirmajām 5 lapām.

```
tvs = response.css('.s-item')

for tv in tvs:
    nosaukums = tv.css('h3.s-item__title::text').get()
    cena = tv.css('span.s-item__price::text').get()

    if nosaukums and cena:
        yield {
            'tv_name': nosaukums.strip(),
            'tv_price': nosaukums.strip(),
            'url': url
        }
```

(14. attēls: "*eBay*" rasmošana izmantojot *Scrapy*)

Apskatot 14. attēlu var novērot kā *Scrapy* tiek izmantots televizoru nosaukumu un cenu iegūšanā. "*response.css('s.-item')*" iegūst visus datus, kas atbilst "*.s-item*" CSS selektoram, kurš savukārt apraksta individuālos televizoru ziņojumus. "*tv.css('h3.s-item_title::text').get()*" iegūst tekstu no "*<h3>*" taga ar klasi "*s-item_title*", kas atbilst televizora nosaukumam. "*tv.css('span.s-item_price::text').get()*" iegūst tekstu no "**" taga ar klasi "*s-item_price*", kas atbilst televizora cenai. "*yield*" funkcija apkopo un izvada iegūtos datus bibliotēkas veidā, kuras saturu var saglabāt vai izvadīt terminālī.

4. Vietne ar autentifikāciju "Linkedin"

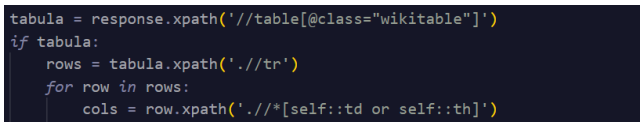
Autors no tīmekļa vietnes izvēlējās iegūt datus par notikumiem, kas tiek ieteikti apmeklēšanai balstoties uz autora personīgo profilu.

(15. attēls: "*Linkedin*" rasmošana izmantojot *Scrapy*)

Aplūkojot 15. attēlu var novērot kā *Scrapy* tiek izmantots notikumu nosaukumu iegūšanai. "*WebDriverWait(self.driver, 10)*" gaida līdz 10 sekundēm, pārbaudot vai meklētais elements atbilst specifiskajam noteikumam. "*EC.presence_of_element_located(...)*" pārbauda vai specifiskais elements, šajā gadījumā notikuma nosaukums, tika ielādēts tīmekļa vietnes lapā. "*p.events-components-shared-discovery-card__event-title.link-without-visited-state.t-black.t-bold*" ir virsrakstam atbilstošais CSS atlasītājs, dati tiek meklēti pēc "*<p>*" elementa, kurš satur notikumu nosaukumiem atbilstošās klases. "*self.driver.find_elements(...)*" metode meklē visus elementus, kas atbilst dotajam CSS selektoram. "*if events: ...*" pārbauda vai "*events*" datu saturošais saraksts nav tukšs un izmantojot "*print(f"- {event.text}*" terminālī tiek izprintēti katram "*event*" elementam atbilstošais teksts.

5. Vietne ar dažāda formāta datiem "*Wikipedia*"

Autors izvēlējās iegūt 2 dažāda formāta datus – tabula un parastais teksts. Tika izvēlēta tabula, kas attēlo COVID-19 statistiku, tas ir saslimšanas gadījumu un mirušo skaits pasaulē, kā arī rindkopa par Latviju saistībā ar šo pandēmiju.



(16. attēls: "*Wikipedia*" rasmošana izmantojot *Scrapy* (1))

(17. attēls: "*Wikipedia*" rasmošana izmantojot *Scrapy* (2))

Apskatot 15. un 16. attēlu var novērot kā ar *Scrapy* palīdzību tiek iegūti tabulas un parastā teksta dati. Mainīgais "*tabula*" (skatīt 16. attēlu) meklē tabulu izmantojot XPath - "*//table[@class="wikitable"]*", un ja tāda tabula eksistē, tad tiek iegūti dati no rindām – "*<tr>*" un dati no katras kolonnas – "*<td>*" vai "*<tr>*". Mainīgais "*virsraksti*" (skatīt 17. attēlu) nodrošina specifisku, ar Latviju un Igauniju saistītu, nodaļu meklēšanā – tiek pārbaudīts vai virsrakstu HTML struktūra sevī ietver "*<h2>*" elementu ar "*mw-headline*" klasi. Ja tiek atrasts virsraksts saturošs identifikatoru "*Latvija*", tiek sākta visu "*<p>*" elementa saturošu datu ieguve tiek konstatēts ka tika atrasta vārdu "*Igaunija*" saturoša nodaļa.

2.4.*Scrapy* un *BeautifulSoup* rezultātu salīdzināšana

Autors, analizējot abu bibliotēku spēju iegūt datus no dažāda veida tīmekļa vietnēm, secināja par *BeautifulSoup* pārākumu *Scrapy*. Visi iegūtie mērījumu dati, to salīdzinājums, izveidotās tabulas un diagrammas ir pieejamas 3. pielikumā.

Tabula 2.4.1

BeautifulSoup un *Scrapy* darbības procentuālā atšķirība					
Posms	CPU	Atmiņa	Temperatūra	Koda garums	Laiks
Pirms	30.53%	-8.98%	0%	-19.44%	8.01%
Palaists	10.63%	-7.74%	0.89%		
Izvade	-34.47%	-7.33%	2.40%		
Beigas	-182.16%	-8.62%	0.47%		

Aplūkojot tabulu 2.4.1, var novērot *BeautifulSoup* un *Scrapy* darbības procentuālo atšķirību. Iegūtais procentuālais rezultāts atspoguļo ievākto vidējo mērījumu atšķirību. Ja procentuālā vērtība ir pozitīva, tad *BeautifulSoup* salīdzinātā vidējā mērījuma vērtība ir lielāka nekā *Scrapy* vidējā vērtība, ja tā ir negatīva – vidējā mērījuma vērtība mazāka. Apskatot tabulu var secināt, ka *BeautifulSoup* ieguva labākus rezultātus salīdzinājumā ar *Scrapy*. Piemēram, apskatot CPU iegūto atšķirību, var novērot, ka atšķirība starp iegūtajiem rezultātiem nav pārlieku liela, kā arī izvades un beigu posmā tā ievērojami ir zemāka nekā *Scrapy*. Apskatot atmiņas izmantošanas salīdzinājuma atšķirību, var secināt, ka *BeautifulSoup* ir pārāks visos posmos, jo tiek izmantots mazāks atmiņas daudzums. Tomēr, var novērot, ka *Scrapy* pārspēj *BeautifulSoup* temperatūras mērījumos, tomēr iegūtā atšķirība nav ievērojami augstāka, kas liecina par to, ka vidējā temperatūra abu bibliotēku testēšanas procesā bija līdzīga. Koda garums ir vēl viena pozīcija, kur *BeautifulSoup* ir pārāks par *Scrapy*, vidēji *BeautifulSoup* koda garums ir gandrīz 20% īsāks nekā Scrapy.

Šie secinājumi atspoguļo tikai abu bibliotēku veiktspēju konkrētā darbam izvēlētā datu kopā. Ir svarīgi ņemt vērā, ka autora izvēlētās datu kopas sevī ietvēra dinamiskās tīmekļa vietnes, no kurām iegūt nepieciešamos datus ir efektīvāk izmantojot *BeautifulSoup*, pateicoties tā integrācijai ar tīmekļa vietņu automatizācijas bibliotēku *Selenium*.

Rasmošanas ietvars *Scrapy* darbojas efektīvāk iegūstot datus no liela mēroga tīmekļa vietnēm vai vairākām vietnēm vienlaikus, kā arī tas ir labāk piemērots statisko tīmekļa vietņu rasmošanai. *Scrapy* integrācija ar *Selenium* nav pilnīga, ko var secināt apskatot tabulas kolonnu par koda garumu. Lai īstenotu rasmošanu no dinamiskām tīmekļa vietnēm ir nepieciešams garāks kods, jo tiek izmantots lielāks bibliotēku skaits, kā arī aprakstītie kritēriji par specifisku datu iegūšanu un pārvietošanās procesa automatizēšanu tīmekļa vietnē ir garāki salīdzinājumā ar *BeautifulSoup*.

Secinājumi

Autoram izdevās veiksmīgi sasniegt darba mērķi un izvirzītos uzdevumus, kā arī pielietot izvirzītās pētīšanas metodes. Autors izpētīja, kas ir rasmošana, kur tā ir nepieciešama un kādu procesu veikšanai tā tiek izmantota. Autors secināja, ka rasmošana ir nozīmīga efektīvai datu iegūšanai no tīmekļa vietnēm un, ka izvēloties atbilstošo rasmošanas bibliotēku vai ietvaru ir nozīmīgi, jo tas būtiski ietekmē iegūto datu kvalitāti un rasmošanas procesa ātrumu. Autoram izdevās izprast rasmošanas bibliotēkas *BeautifulSoup* un rasmošanas ietvara *Scrapy* darbības

principus. Testēšanas procesā autoram izdevās veiksmīgi analizēt un salīdzināt *BeautifulSoup* un *Scrapy* darbību saskaroties ar dažāda veida tīmekļa vietnēm.

Praktiskajā daļā autors salīdzinot *BeautifulSoup* un *Scrapy*, secināja, ka *BeautifulSoup* rasmošanas bibliotēka ir efektīvāka datu ieguvei, pateicoties tās spējai veiksmīgi iegūt datus no statiskajām tīmekļa vietnēm un dinamiskajām vietnēm, pateicoties veiksmīgai integrācijai ar automatizēšanas bibliotēku *Selenium*. Tomēr, rasmošana izmantojot *Scrapy* var būt efektīva, ja tiek izmantota atbilstoša datu kopa, tomēr autors darbam izvēlējas plašu datu klāstu, kur datu iegūšanas process tika apgrūtināts dinamisko tīmekļa vietņu dēļ, kā rezultātā vidējie mērījumi parādīja, ka *BeautifulSoup* darbība ir efektīvāka.

Darba autoram izdevās veiksmīgi apstiprināt izvirzīto hipotēzi, izpildīt darba uzdevumus un realizēt darba mērķi.

Izmantotās literatūras sarakts

1. Khder, M. A. (2021). Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application. *Int. J. Advance Soft Compu. Appl.*, 13(3). Pieejams: https://www.i-csrs.org/Volumes/ijasca/2021.3.11.pdf
2. Britannica. World Wide Web. Pieejams: https://www.britannica.com/topic/World-Wide-Web
3. Mahesh, B. (2020). Machine Learning Algorithms - A Review. *ResearchGate*. Pieejams: https://www.researchgate.net/profile/BattaMahesh/publication/344717762_Machine_Learning_Algorithms_-A_Review/links/5f8b2365299bf1b53e2d243a/Machine-Learning-Algorithms-A-Review.pdf
4. Scrapfly. How Web Scraping Can Revolutionize Machine Learning? Pieejams: https://scrapfly.io/use-case/how-web-scraping-can-revolutionize-machine-learning
5. Berners-Lee, T. Frequently asked questions. Pieejams: https://www.w3.org/People/Berners-Lee/FAQ.html
6. Lenovo. What is a dynamic website? Pieejams: https://www.lenovo.com/us/en/glossary/dynamic-website/?orgRef=https%253A%252F%252Fwww.google.com%252F&srsltid=AfmBOoqLn3FiRO3oDn0W9xS_rYyPUkPa046B3tAYSjrg6gGQADJwEh7D
7. DataHen. Python HTML Parser Guide. Pieejams: https://www.datahen.com/blog/python-html-parser/
8. Simplilearn. What Is Web Scraping? Pieejams: https://www.simplilearn.com/what-is-web-scraping-article
9. ResearchGate. Overview of Web Scraping System [Attēls]. Pieejams: https://www.researchgate.net/figure/Overview-of-web-scraping-system_fig2_347999311
10. Webscraping.ai. What is HTTP and how is it used in web scraping? Pieejams: https://webscraping.ai/faq/http/what-is-http-and-how-is-it-used-in-web-scraping
11. IEEE Computer Society. Bulletin of the Technical Committee on Data Engineering, 23(4), December 2000. Pieejams: https://cs.brown.edu/courses/cs227/archives/2017/papers/data-cleaning-IEEE.pdf#page=5
12. Myers, D., & McGuffee, J. W. Choosing Scrapy. *Northern Kentucky University*. Pieejams: https://www.researchgate.net/profile/James-Mcguffee/publication/314179276_Choosing_Scrapy/links/58b8a088a6fdcc2d14d9a326/Choosing-Scrapy.pdf

13. Pluralsight. Implementing Web Scraping with Scrapy. Pieejams: https://www.pluralsight.com/resources/blog/guides/implementing-web-scraping-with-scrapy
14. Crummy. Beautiful Soup Documentation. Pieejams: https://www.crummy.com/software/BeautifulSoup/bs4/doc/
15. LangChain. Beautiful Soup. Pieejams: https://python.langchain.com/docs/integrations/document_transformers/beautiful_soup/
16. Oxylabs. Scrapy vs. Beautiful Soup: A Comparison of Web Scraping Tools. Pieejams: https://oxylabs.io/blog/scrapy-vs-beautifulsoup
17. W3Schools. HTML Tree [Attēls]. Pieejams: https://www.w3schools.com/js/pic_htmltree.gif

Pielikums

1. Github, *BeautifulSoup* testēšana. Pieejams: https://github.com/daigale1/ZPD_BeautifulSoup.git
2. Github, *Scrapy* testēšana. Pieejams: https://github.com/daigale1/ZPD_Scrapy.git
3. Github, Bibliotēku testēšanas mērījumi. Pieejams: https://github.com/daigale1/ZPD_Excel.git
4. Attēls: *BeautifulSoup* un *Scrapy* Java tīmekļa vietnes rasmošanas kodu shēmas: