УНИВЕРЗИТЕТ У БЕОГРАДУ
МАТЕМАТИЧКИ ФАКУЛТЕТ

Зорана Гајић

САВРЕМЕНИ АЛАТИ ЗА ПРИКУПЉАЊЕ ПОДАТАКА СА ВЕБ-СТРАНИЦА

мастер рад

Београд, 2023.

Ментор:

др Милена Вујошевић Јаничић, ванредни професор
Универзитет у Београду, Математички факултет

Чланови комисије:

др Ана Аничић, ванредни професор
University of Disneyland, Недођија

др Лаза Лазић, доцент
Универзитет у Београду, Математички факултет

Датум одбране: 15. јануар 2016.

Породици

Наслов мастер рада: Савремени алати за прикупљање података са веб-страница

Резиме:

Кључне речи: прикупљање података са веб-страница

Садржај

Глава 1

Увод

TODO:

Глава 2

Прикупљање података са веб-страница

Веб[1](енг. *World Wide Web, WWW*) представља највећи извор података у историји човечанства, али се већина ових података састоји од неструктурираних информација, што може отежати њихово прикупљање [6]. На многим веб-сајтовима забрањено је копирање и преузимање података, али на сајтовима на којима је преузимање података дозвољено, ручно копирање може потрајати данима или недељама.

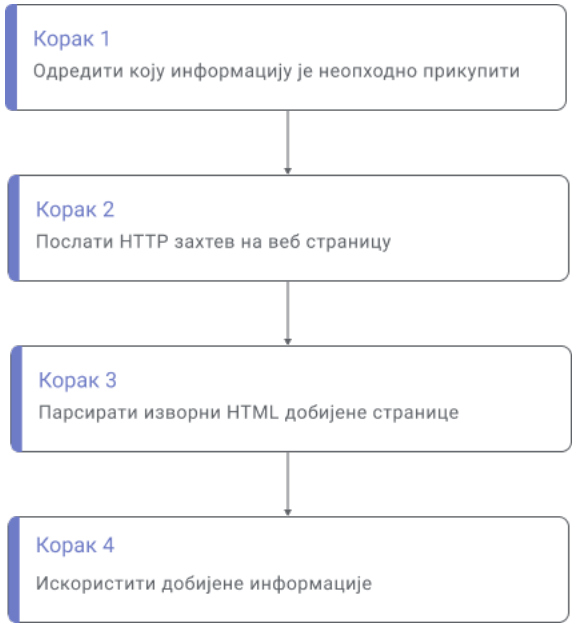
Веб скрејпинг (енг. *Web scraping*) представља аутоматизовани процес који омогућава издвајање података са различитих веб-страница и њихово чување у структурираном формату ради тренутне употребе или касније анализе. Постоје различити програмски језици који пружају подршку за имплементацију Веб скрејпинга, од којих су најпопуларнији: Пајтон (енг. *Python*), Јава (енг. *Java*) и Руби (енг. *Ruby*).

Поступак прикупљања информација састоји се од неколико фаза, које су приказане на слици 2.1. Прва фаза је проналажење одговарајуће веб-странице за прикупљање података (детаљније објашњено у одељку 2.1) и одређивање информација које су потребне за прикупљање. Након тога, потребно је послати *HTTP*[2](енг. *Hypertext Transfer Protocol*) захтев на жељену веб-страницу и преузети изворни код *HTML* странице. Пре него што се парсира *HTML* код, потребно је пронаћи најбољи начин за индексирање жељених елемената, а за-

[1]Светска мрежа, познатија као Веб, систем је међусобно повезаних, хипертекстуалних докумената који се налазе на интернету.

[2]*HTTP* је мрежни протокол који припада слоју апликације референтног модела ОСИ, представља главни и најчешћи метод преноса информација на Вебу.

тим парсирати изворни код *HTML* странице и извршити неопходну радњу са добијеним информацијама [12].



Слика 2.1: Фазе прикупљања и употребе података

2.1 Изазови

Веб скрејпинг се сматра корисним процесом за добијање увида у податке. Међутим потребно је пазити на правне аспекте, како би се избегли легални проблеми. Да би прикупљање података било успешно, од суштинског значаја је квалитет добијених података. Како би се добили квалитетни подаци, потребно је да је сам веб-сајт исправан, односно да не садржи неисправне линкове, јер се веб скрејпинг обично изводи преко целог веб-сајта, а не само преко одређених страница.

Када се ради о пројектима великих размера и обимних база података, један од честих изазова јесте складиштење података. Овај изазов је повезан са ефикасним прикупљањем, обрадом и анализом велике количине података који се могу прикупити путем веб скрејпинга са различитих извора. Овај проблем може бити решен употребом већ постојећих платформи за складиштење.

Када се ради о динамичким веб-сајтовима, решење може бити приказивање у претраживачу без заглавља (енг. *Headless Chromium* [9]), што омо-

гућава да се претраживач покреће у окружењу сервера. Пријављивање корисника на веб-страницу може представљати велики изазов, али се то може решити уз помоћ библиотека као што су *Selenium* и *Scrapy*.

У наставку ће бити описане најчешће заштите од напада на веб-странице који могу ометати процес веб скрејпинга:

1. *CAPTCHA* (енгл. *Completely Automated Public Turing test to tell Computers and Humans Apart*)

   CAPTCHA је технологија која се користи за проверу и потврду да је корисник веб-странице заиста човек, а не компјутерски програм [14]. Провера се постиже приказивањем изазова, на пример слике са текстом или бројевима, које је лако за људе да га виде и реше, али тешко за компјутерске програме да га аутоматски реше. Корисници обично морају да унесу решење изазова како би потврдили да су људи и дозволили приступ подацима на веб страницама.

2. Захтеви за аутентификацију

   Пре приступања подацима, кориснику може бити захтевано да унесе своје корисничко име и лозинку или да потврди своју електронску пошту.

3. Блокирање *IP* (енгл. *Internet Protocol address*) адреса.

   Веб странице могу блокирати *IP* адресе које се повезују са прекомерним бројем захтева или са ботовима који су идентификовани као нежељени. Ово може бити привремено или трајно.

4. Провера корисничког агента.

   Сваки *HTTP* захтев у заглављу шаље корисничког агента (енг. *user agent*). Коришћењем овог подешавања веб-сајт идентификује претраживач који му приступа: његову верзију и платформу. Уколико се користи исти кориснички агент у сваком захтеву, веб-сајт може лако да открије да је у питању аутоматизовани приступ страници.

5. Праћење учесталости прикупљања података.

   Да би се спречило преузимање садржаја са веб-странице, веб-сајт може увести ограничење фреквенције за ботове. Циљ је спречити преузимање

садржаја са веб-странице у превеликој количини или превеликој брзини. Ово ограничење може укључивати број захтева по јединици времена, максималну брзину преузимања.

6. Постојање *robots.txt* фајла.

   robots.txt фајл представља политику веб-сајта и најчешће се проналази на нивоу основног директоријума. Уколико фајл садржи линије попут ових приказаних у наставку, то значи да веб-сајт не жели да се прикупљају подаци са њега

   ```
   User-agent: *
   Disallow:/
   ```

7. Законске заштите

   У неким случајевима, веб странице се могу заштитити законима о ауторским правима, заштитом података или другим релевантним законима који спречавају копирање или употребу података са веб-странице без дозволе власника.

2.2 Примењене технологије

Веб скрејпинг технологије подразумевају различите методе и алате за издвајање података са веб-страница. У оквиру ових технологија користе се: регуларни изрази (енг. *Regular Expressions, RegEx*), тагови (енг. *tags*), *CSS* (енг. *Cascading Style Sheets*) селектори и *XPath* (енг. *XML Path Language* [2]).

Препоручени редослед идентификације елемената у оквиру *HTML* кода током веб скрејпинга је следећи:

1. Преко идентификатора — ако елементи имају јединствени идентификатор, најбрже и најпоузданије је користити овај начин идентификације.

2. По имену класе — ако се елементи налазе у истој класи, могу се идентификовати преко имена класе. Ово је корисно када је потребно издвојити групу елемената са заједничким стилом или функционалношћу.

3. По таговима — ако је неопходно издвојити све елементе са одређеним тагом, овај начин идентификације је најбољи.

4. *CSS* селектори — ако постоје елементи који немају јединствен идентификатор, али имају јединствен *CSS* стил, могу се идентификовати преко *CSS* селектора.

5. Регуларни изрази — ако је неопходно издвојити елементе на основу текста који се налази у њима.

6. *XPath* — ово је најопштији начин идентификације елемената у *HTML* коду.

Регуларни изрази

Регуларни изрази представљају метод за усклађивање специфичних образаца у зависности од датих комбинација, који се могу користити као филтери за добијање жељеног резултата. У прикупљању података регуларни изрази се често користе за поређење шаблона и издвајање података, за локализовање и издвајање специфичних података из *HTML* или *XML* докумената. Једна од најзначајнијих предности регуларних израза јесте у њиховој универзалности, тј. могу се применити на било коју врсту података.

У многим програмским језицима, регуларни изрази се подржавају кроз уграђене библиотеке или модуле. Модул *re* програмског језика Пајтон пружа подршку регуларним изразима за поређење шаблона и издвајање података.

У наставку је дат пример регуларног израза који би се могао искористити за претраживање и издвајање свих веб-адреса из изворног кода *HTML* странице. Конкретно, тражи се почетак хипервезе $<a$ која садржи атрибут $<href$, а затим се издваја веб-адреса из овог атрибута и ставља у групу.

```
1 regex_pattern = r"<a\s+(?:[^>]*?\s+)?href=\"([^\"]*)\""
```

CSS селектори

CSS селектори се могу користити у процесу сакупљања података са веб-страница како би се идентификовали и издвојили одређени елементи. Овакав приступ је посебно користан када се ради са веб-страницама које не поседују јасну структуру и организацију.

CSS селектори раде на принципу идентификације елемената према њиховом имену ознаке, имену класе или идентификатору. На пример, селектор

„div[class='imeKlase']" се користи за издвајање свих *div* елемената који имају класу *imeKlase*.

Тагови

Тагови играју кључну улогу у прикупљању података са веб-страница јер помажу у идентификацији и издвајању одређених информација из изворног кода *HTML* страница. Тагови у *HTML* коду се користе за дефинисање структуре веб-странице. Сваки таг представља одређени елемент или секцију странице, као што су заглавља, пасуси, слике и линкови.

У наставку су наведени тагови који се најчешће користе:

<html> — Означава почетак и крај HTML документа.

<body> — Представља садржај документа који је видљив кориснику.

<h1> до <h6> — Користе се за дефинисање наслова.

<p> — Користи се за дефинисање параграфа текста.

<a> — Ствара хиперлинк (енгл. *Hyperlink*) до друге веб-странице.

 и <li> — Користе се за стварање неуређене листе ставки.

 и <li> — Користе се за стварање уређене листе ставки.

 — Користи се за дефинисање одељка документа у сврху стилизовања.

 — Користи се за дефинисање малог дела текста у сврху стилизовања.

Језик *XPath*

Језик *XPath* представља флексибилан начин адресирања различитих делова *XML*[3] (енг. *Extensible Markup Language*) документа који су у формату *XML* или неком сличном формату. То га чини погодним за навигацију кроз објектни модел било ког таквог документа[4] (енг. *Document Object Model, DOM*), уз помоћ *XPath*[5] (енг. *XPathExpression*). Израз *XPath* дефинише образац за одабир скупа чворова и садржи преко 200 уграђених функција [2]. Овај језик је дефинисао *WWW* конзорцијум. У овом раду ће се језик *XPath* користити за одабир елемената са изворног кода *HTML* страница.

[3] *XML* представља проширив (мета) језик за означавање (енгл. *markup*)
[4] Објектни модел документа представља хијерархијски приказ структуре веб-сајта.
[5] *XPath* израз дефинише образац за одабир скупа чворова.

Синтакса језика *XPath*

Језик *XPath* користи изразе путања за избор чворова у *XML* документу. Чвор се одабира праћењем путање или корака. Неки корисни примери израза путања су наведени у наставку:

//h2 — Издваја све *h2* елементе.

//div//p — Издваја све *p* елементе који се налазе унутар *div* — блока.

//ul/li/a — Издваја све линкове који се налазе унутар неуређених листи.

//ol/li[2] — Издваја други елемент уређене листе.

//div/* — Издваја све неурђене елементе који се налазе унутар *div* блокова.

//*[@id=„id”] — Издваја елемент са одређеним идентификатором.

//*[@class=„class”] — Издваја све елементе са одређеном класом.

//a[@name or @href] — Издваја све линкове са атрибут *name* или *href*.

//a[last()] — Издваја последњи линк.

//table[count(tr)=1] — Издваја број редова у табели.

string(42) — Конвертује друге типове података у ниску.

number(„42”) — Конвертује друге типове података у број.

//* — Издваја све елементе.

//a/text() — Издваја текст линка.

./a — Тачка издваја тренутни чвор.

contains(„abcdefg”, „cde”) — Проналази део ниске који садржи задату подниску.

starts-with(„abcdefg”, „abc”) — Проналази део ниске који почиње задатом подниском.

ends-with(„abcdefg”, „efg”) — Проналази део ниске који се завршава задатом подниском.

Глава 3

Преглед алата за прикупљање података са веб-страница

TODO: Кратак увод шта ће се користити у поглављу, који програмски језик, које библиотеке, опис сајта, жељени циљеви, препреке са којима ћу се сусрести у раду

3.1 Библиотека *BeautifulSoap*

Библиотека *BeautifulSoap* је Пајтон библиотека која се користи за парсирање и претраживање *HTML* и *XML* докумената.[13]. Ова библиотека подржава различите врсте навигације кроз *HTML* и *XML* документе, као што су претраживање по имену тагова, претраживање по садржају тагова, претраживање по атрибутима тагова и слично. Једна од главних особина библиотеке *BeautifulSoap* је да је компатибилна са различитим парсерима, укључујући *html.parser* [3], *lxml* [4] и *html5lib* [8]. За разлику од других библиотека које ће се касније разматрати, ова библиотека не може сама да приступи веб-страници и потребни су јој помоћни модули.

Библиотека *BeautifulSoap* има многе карактеристике које олакшавају рад са њом. Библиотека се лако инсталира помоћу наредбе *pip* и има једноставан интерфејс (енг. *interface*). Такође, библиотека *BeautifulSoap* омогућава лако преузимање и извлачење података из *HTML* и *XML* докумената и рад са различитим парсерима.

Инсталација

Библиотека *BeautifulSoap* се може инсталирати користећи алат за инсталирање пакета за програмски језик Пајтон звани *pip* [1]. Неопходно је унети следећу наредбу у командну линију:

```
pip3 install bs4
```

Ова наредба ће преузети и инсталирати најновију верзију библиотеке *BeautifulSoap*. Након успешне инсталације, неопходно је увести библиотеку у Пајтон код користећи следећу наредбу:

```
from bs4 import BeautifulSoup
```

Провера динамичности веб-странице

Многе веб-странице, укључујући веб-страницу *kiwi*, која се анализира у овом раду, користе динамичке технологије које омогућавају промену садржаја без освежавања целе странице, што представља изазов при парсирању таквих страница. У овом контексту, библиотека *BeautifulSoap* се најчешће користи за анализу *HTML* или *XML* кода веб-страница, али због динамичности неких страница, могуће је да се не ухвате све промене на страници, што је управо случај са веб-страницом *kiwi*. Због тога се користе библиотеке попут *Selenium* [7] и *Scrapy* [5] за праћење промена у реалном времену, као и додаци за библиотеку *BeautifulSoap*, попут *Requests-HTML* [11], који омогућавају преузимање и анализу динамичког садржаја.

Прикупљање *HTML* кода веб-странице

Библиотека *BeautifulSoup* не представља самосталну библиотеку за прикупљање података са веб-страница. Да би се преузео *HTML* код веб-странице неопходно је инсталирати библиотеку *Requests-HTML*, која омогућава креирање *HTTP* захтева на одређену веб-страницу и за одговор добија *HTML* код те странице. Постоји неколико метода од значаја у пакету *Requests-HTML* [10]:

- `get(url, params, args)`

  Шаље *HTTP GET* захтев на наведену веб-адресу

- `post(url, data, json, args)`

  Шаље *HTTP POST* захтев на наведену веб-адресу

- `put(url, data, args)`

  Шаље *HTTP PUT* захтев на наведену веб-адресу

Код приказан на листингу 3.1 представља код у програмском језику Пајтон који преузима *HTML* код веб-странице. Важно је знати да преузимањем веб-странице помоћу Пајтон библиотеке *Requests-HTML*, постоји могућност да се деси да страница није доступна на серверу (или да је дошло до грешке у њеном преузимању), или да сервер није доступан.

```
1 import requests
2
3 website = 'https://www.kiwi.com/en/search/results/belgrade-serbia/
    paris-france'
4 response = requests.get(website)
5 html = response.text
```

Listing 3.1: Прикупљање *HTML* кода веб-странице

Парсирање *HTML* кода веб-странице

Пајтон нуди разне библиотеке за парсирање *HTML* кода, од којих су две најзаступљеније: *lxml* и *html.parser*. Парсер *lxml* је најбржи парсер веб-страница према званичној документацији библиотеке *BeautifulSoup* [13], који може да анализира велике и сложене документе. Парсер *html.parser* је уграђени Пајтон парсер који је намењен да ради са мањим и једноставнијим *HTML* документима [10].

Да би се извршило парсирање добијеног *HTML* кода веб-странице, прво је неопходно креирати објекат *BeautifulSoup* уз помоћ добијеног *HTML* кода и жељеног парсера. Осим наведеног корака, у Пајтон коду на листингу 3.2 је приказано да резултат креирања објекта *BeautifulSoup* нуди брзо издвајање наслова и текста веб-странице, поред разних других информација.

```
1 from bs4 import BeautifulSoup
2
3 soup = BeautifulSoup(html, 'lxml')
4
5 print(soup.title)
```

```
# <title>Belgrade-Paris trips</title>

print(soup.get_text())
# Belgrade Paris trips
# TravelCarsRoomsStoriesDealsTravel hacksloadingManage your trips...
```

Listing 3.2: Креирање објекта *BeautifulSoup*

Добијени објекат *BeautifulSoup* такође омогућава приступ различитим деловима *HTML* кода користећи методе као што су *.find()* и *.find_ all()*. Метода *.find()* користи се када је потребно пронаћи први елемент у *HTML* коду који одговара одређеном тагу или класи. Ова метода враћа први пронађени елемент који одговара постављеним критеријумима, док се метода *.find_ all()* користи када је потребно пронаћи све елементе у *HTML* коду који одговарају одређеном тагу или класи. Ова метода враћа листу свих пронађених елемената који одговарају постављеним критеријумима.

На пример, ако је неопходно да се издвоје сви летови са веб странице, треба да се искористи функција *.find_ all()* над добијеним објектом *BeautifulSoup*. Функција *.find_ all()* прихвата аргумент који представља начин идентификације жељеног елемента, у овом случају комплетног блока са информацијама о лету. Када се детаљно прегледа структура *HTML* кода веб-странице, може да се примети да је сваки лет издвојен у *div* са дугачким именом класе, што представља једино по чему се тај блок може идентификовати. На сваки појединачан лет треба применити функцију *.find()* како би се пронашао жељени елемент. Затим, када је жељени елемент пронађен, може да се приступи његовим атрибутима и вредностима, као и његовом тексту користећи различите атрибуте објекта *BeautifulSoup*. Дати пример описује код приказан на листингу 3.3.

```
    all_flights = soup.find_all('div', class_="
ResultCardstyled__ResultCardInner-sc-vsw8q3-9 hlQpUC")
    departure_date_regex = re.compile('.*DepartureDate.*')

    for flight in all_flights:
        dates = flight.find_all("p", {"class" : departure_date_regex})

        return_airport = flight.find_all('div',  {"class" :
departure_info_regex})[2].find('div').text

```

```
9         prices.append(flight.find('strong',  {"data-test" : '
    ResultCardPrice'}).text)
```

Listing 3.3: Парсирање *HTML* кода веб-странице

Прикупљање података са више веб-страница

Када се користи библиотека *BeautifulSoup* за прикупљање података са више веб-страница, могу се јавити проблеми у вези са аутоматским прикупљањем података са свих жељених страница. Наиме, када се прикупљају подаци са једне странице, обично се користи функција

```
requests.get(url)
```

за дохват *HTML* кода и затим функција

```
BeautifulSoup(html, 'lxml')
```

за анализу *HTML* кода и издвајање неопходних података. Међутим, ако се подаци прикупљају са више страница, неопходно је итерирати кроз све странице и аутоматски дохватити *HTML* код за сваку страницу. На пример, ако странице имају адресе које се разликују само по броју странице, може да се искористи петља која пролази кроз све адресе и дохвата *HTML* код сваке странице.

У случају веб-странице *kiwi*, постоји листа дестинација на једној веб-страници, где свака појединачна дестинација представља линк ка страници са детаљима о летовима за ту дестинацију. Како би сви подаци били прикупљени, неопходно је проћи кроз сваку страницу. Међутим, овај процес је знатно отежан у овом случају, јер је листа дестинација динамички добијена и уочити шаблон коришћених адреса није могуће.

(TODO: увезати некако пасус испод као закључак 3.1) Библиотека *BeautifulSoup* је корисна библиотека која омогућава лако и ефикасно парсирање *HTML* кода, уз коју се може лако приступити и манипулисати различитим деловима *HTML* кода.

3.2 Библиотека *Selenium*

(TODO: увод у 3.2 + сајтови покренути са JS)

Инсталација

Библиотека *Selenium*, слично библиотеци *BeautifulSoup*, се може инсталирати користећи алат *pip*. Неопходно је унети следећу наредбу у командну линију:

```
pip3 install selenium
```

Након успешне инсталације, неопходно је увести библиотеку у Пајтон код користећи следећу наредбу:

```
import selenium
```

Улога драјвера

Управљачки програм или драјвер (енг. *driver*) је рачунарски програм који омогућава комуникацију између програма вишег нивоа, као што је апликација, и рачунарске опреме. Када се користи библиотека *Selenium*, није могуће директно комуницирати са Веб-прегледачем (енг. *web browser*), већ је неопходно користити драјвер који ће посредовати у комуникацији између кода и Веб-прегледача и омогућити контролу над Веб-прегледачем користећи библиотеку *Selenium*. За сваки Веб-прегледач постоји одређени драјвер који се користи са библиотеком *Selenium*. На пример, за Веб-прегледач Гугл кроум (енг. *Google Chrome*) се користи драјвер *ChromeDriver*, док се за Веб-прегледач Мозила фајерфокс (енг. *Mozilla Firefox*) користи драјвер *GeckoDriver*.

Како би се омогућило коришћење драјвера у Пајтон коду, потребно је да се преузме одговарајућа верзија драјвера за неопходни Веб-прегледач. Након тога треба навести путању до драјвера и инстанцирати драјвер коришћењем методе *webdriver* из библиотеке *Selenium*. Наведени код на листингу 3.4 представља претходно описане кораке за случај када је коришћен Веб-прегледач Гугл кроум. Након тога, код може да управља Веб-прегледачем, отворити веб-адресу и радити са њом. На крају, линија *driver.quit()* затвара Веб-прегледач и ослобађа коришћене ресурсе.

```
1 from selenium import webdriver
2 from selenium.common import exceptions
3 from selenium.webdriver.chrome.service import Service
4 from selenium.webdriver.common.by import By
5
```

```
website = 'https://www.kiwi.com/en/search'
path = '/usr/local/bin/chromedriver_mac64/chromedriver'
service = Service(executable_path=path)
driver = webdriver.Chrome(service=service)
driver.get(website)
driver.maximize_window()
...
driver.quit()
```

Listing 3.4: Прикупљање *HTML* кода веб-странице

Лоцирање елемената

Библиотека *Selenium* дефинише два главна метода за лоцирање елемената на веб-страницама:

- *findElement*

  За резултат враћа један елемент који одговара задатом критеријуму.

- *findElements*

  За резултат враћа листу елемената који задовољавају дати критеријум.

Оба наведена метода прихватају аргумент у облику стратегије лоцирања. Библиотека *Selenium* нуди велики број уграђених стратегија за лоцирање елемената које могу бити коришћене у претходно наведеним методима. Неке од често коришћених стратегија укључују *ID, NAME, XPATH, TAG_NAME, CLASS_NAME и CSS_SELECTOR.*

TODO: примери TODO: пагинација, имплицитно и експлицитно чекање, попуњавање формулара, логин, кретање између страница, скрејповање странице са бесконачним скролом, како променити корисничког агента и зашто

3.3 Библиотека Scrapy

TODO: Инсталација, scrapy шаблони, креирање паука, стругање са више линкова, стругање са више страница, стругање АПИ, попуњавање формулара, логин, како променити корисничког агента, најосновније потребне функције LUA програмског језика (неопходно за SPLASH), SPLASH, шта представља SPLASH, зашто је неопходан, како превазићи Captcha)

Глава 4

Анализа резултата

TODO: Поређење перформанси, дијаграм односа времена стругања сајта и коришћене библиотеке, како оценити добијене резултате, табела са поређеним карактеристикама коришћених библиотека, препоруке, смернице за унапређење коришћених технологија

Глава 5

Закључак

TODO:

Библиографија

[1] pip documentation v23.1.1.

[2] Keio) 1999 W3C® (MIT, INRIA. XPath. on-line at: https://www.w3.org/TR/1999/REC-xpath-19991116/.

[3] Python Software Foundation 2001-2023. html.parser — Simple HTML and XHTML parser. on-line at: https://docs.python.org/3/library/html.parser.html.

[4] Stefan Behnel and Martijn Faassen. Parsing XML and HTML with lxml. on-line at: https://lxml.de/parsing.html.

[5] Maintained by Zyte (formerly Scrapinghub) and many other contributors. Scrapy. on-line at: https://scrapy.org/.

[6] Osmar Castrillo-Fernández. Web scraping: Applications and tools.

[7] 2023 Software Freedom Conservancy. Selenium. on-line at: https://www.selenium.dev/documentation/.

[8] Sam Sneddon Copyright 2006 2013, James Graham and contributors Revision 3e500bb6. html5lib. on-line at: https://html5lib.readthedocs.io/en/latest/.

[9] Gitiles. Headless Chromium. on-line at: https://chromium.googlesource.com/chromium/src/+/lkgr/headless/README.md.

[10] Ryan Mitchell. Web scraping with python. In *Web Scraping with Python*, 2015.

[11] A Kenneth Reitz P. Requests: HTTP for Humans. on-line at: https://requests.readthedocs.io/en/latest/.

[12] Emil Persson. Evaluating tools and techniques for web scraping.

[13] Leonard Richardson. Beautiful Soup Documentation, 2004-2023. on-line at: `https://www.crummy.com/software/BeautifulSoup/bs4/doc/`.

[14] 2000-2010 Carnegie Mellon University. CAPTCHA. on-line at: `http://www.captcha.net/`.

Биографија аутора

Зорана Гајић, рођена 04.11.1997 у Москви, где је ... TODO: