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 Весна Маринковић, доцент
Универзитет у Београду, Математички факултет

др Александар Картељ, доцент
Универзитет у Београду, Математички факултет

Датум одбране: 15. јануар 2016.

Велика захвалност менторки на саветима и мотивацији и породици на подршци.

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

Резиме: TODO:

Кључне речи: прикупљање података са веб-страница, веб-скрејпинг, парсирање *HTML* кода, библиотека *BeautifulSoap*, библиотека *Selenium*, библиотека *Scrapy*

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.

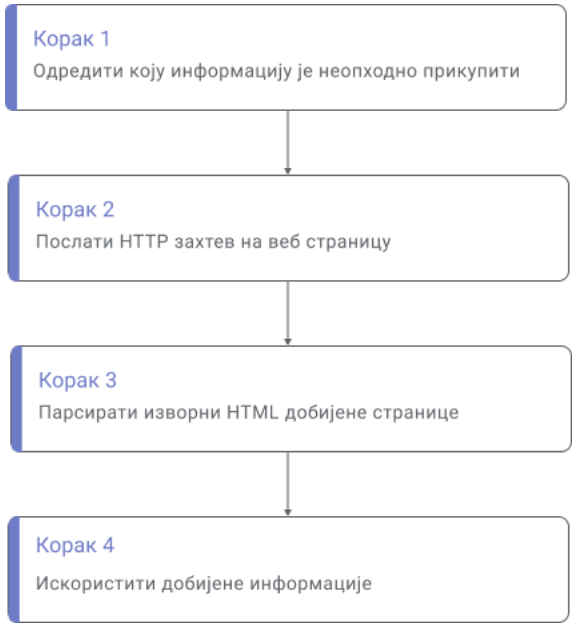
Веб скрејпинг (енг. *Web scraping*) представља аутоматизовани процес који омогућава издвајање података са различитих веб-страница и њихово чување у структурираном формату ради тренутне употребе или касније анализе. Постоје различити програмски језици који пружају подршку за имплементацију Веб скрејпинга, 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 [11].



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.

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 програм [13]. Провера се постиже приказивањем изазова, на пример слике са текстом или бројевима које је потребно препознати. Изазов је обично лак људима за решавање, али је тежак за програме који то треба брзо и аутоматски да реше. Корисници обично морају да унесу решење изазова како би потврдили да су људи и како би им био дозвољен приступ подацима на веб-страницама.

2. Захтеви за аутентификацију

   Пријава корисника на веб-страницу може да представља велики изазов приликом веб-скрејпинга динамичних веб-страница. Уобичајени процес пријаве обухвата уношење корисничког имена и лозинке у одговарајућа поља на веб-страници, а затим клик на дугме за пријављивање. Приликом аутоматизације овог процеса могу се јавити потешкоће, али се оне могу решити уз помоћ библиотека као што су *Selenium* [7] и *Scrapy* [5].

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 као и 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* [2] (енг. *XML Path Language*).

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.

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.

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

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

`<div>` — Користи се за дефинисање одељка документа у сврху стилизовања.

`<span>` — Користи се за дефинисање малог дела текста у сврху стилизовања.

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*.

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=\"([^\"]*)\""
```

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* (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.

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:

[3] *XML* представља проширив (мета) језик за означавање (енгл. *markup*)

[4] Објектни модел документа представља хијерархијски приказ структуре веб-сајта.

`//h2` — Издваја све елементе *h2*.

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

`//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 блокова *div*.

`//*[@id=,,id'']` — Издваја елемент са одређеним идентификатором *„id"*.

`//*[@class=,,class'']` — Издваја све елементе са одређеном класом *„class"*.

`//a[@name or @href]` — Издваја све линкове који имају атрибут *name*, атрибут *href* или оба.

`//a[last()]` — Издваја последњи линк.

`//table[count(tr)=1]` — Издваја табеле које имају само један ред у њима.

`string(n)` — Конвертује друге типове података у ниску. На пример, уколико је *n* број 42, онда ће резултат бити ниска „42".

`number(n)` — Конвертује друге типове података у број. На пример, уколико је *n* ниска „42", онда ће резултат бити број 42.

`//*` — Издваја све елементе.

`//a/text()` — Издваја текст линка.

`./a` — Тачка издваја тренутни чвор.

`contains(a, b)` — Проверава да ли се одређена ниска појављује унутар друге ниске. Први аргумент је ниска у којем се врши претрага, а други аргумент је ниска која се тражи. На пример, уколико је *a* ниска „abcdefg", а *b* ниска „bcd", онда ће резултат бити вредност *true*.

`starts-with(a, b)` — Проверава да ли одређена ниска почиње задатом подниском, односно да ли ниска *a* почиње са ниском *b*. На пример, уколико је *a* ниска „abcdefg", а *b* ниска „abc", онда ће резултат бити вредност *true*.

`ends-with(a, b)` — Проверава да ли одређена ниска завршава задатом подниском, односно да ли се ниска *a* завршава са ниском *b*. На пример, уколико је *a* ниска „abcdefg", а *b* ниска „efg", онда ће резултат бити вредност *true*.

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.[12]. 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 *audible.com*, 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 *audible*. 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* [10], 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* [9]:

- `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 url = 'https://www.audible.com/search'
4 try:
5     response = requests.get(url)
6 except requests.exceptions.RequestException as err:
7     print("Error fetching page")
8     exit()
9
10 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* [12], 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 [9].

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)
6 print(soup.get_text())
```

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.

```
1     all_flights = soup.find_all('div', class_="
    ResultCardstyled__ResultCardInner-sc-vsw8q3-9 hlQpUC")
2     departure_date_regex = re.compile('.*DepartureDate.*')
3
4     for flight in all_flights:
5         dates = flight.find_all("p", {"class" : departure_date_regex})
6
7         return_airport = flight.find_all('div',  {"class" :
    departure_info_regex})[2].find('div').text
8
```

```
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. 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.

3.2 Библиотека *Selenium*

Библиотека *Selenium* је популарна библиотека програмског језика Пајтон која се користи за ефикасну аутоматизацију интеракције са веб-страницама. Она омогућава симулирање корисничке интеракције са веб-страницама, као што су уношење текста, кликтање, претраживање елемената и прикупљање података.

Библиотека *Selenium* пружа богат скуп функција за претрагу елемената на веб-страници, као што су проналажење елемената по идентификатору, имену, класи, ознаци или изразу *XPath*. Ово омогућава једноставну манипулацију одређеним деловима веб-страница. Још једна корисна особина ове библиотеке је могућност руковања чекањима и интеракцијом са динамичким елементима странице. На пример, могуће је да се сачека да се одређени елемент учита пре него што се изврше следеће наредбе.

Укратко, библиотека *Selenium* је моћан алат за аутоматизацију веб-прегледача који омогућава тестирање, интеракцију и прикупљање података са веб-страница на ефикасан начин.

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 отвори веб-адресу и управља истом. 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
6 website = 'https://www.kiwi.com/en/search'
7 path = '/usr/local/bin/chromedriver_mac64/chromedriver'
8 service = Service(executable_path=path)
9 driver = webdriver.Chrome(service=service)
10 driver.get(website)
11 driver.maximize_window()
12 ...
13 driver.quit()
```

Listing 3.4: Прикупљање *HTML* кода веб-странице

Headless режим

Headless режим се односи на извршавање програма у позадини, без потребе за приказивањем корисничког графичког интерфејса и интеракције са корисником путем миша и тастатуре. Ова врста извршавања програма је корисна у различитим контекстима и служи за разне сврхе, а једна од њих је веб-скрејпинг.

Веб-скрејпинг је процес прикупљања података са веб-страница. Програм који ради у *headless* режиму може аутоматски посетити веб-странице, извршавати одређене акције и прикупљати податке без потребе за приказивањем страница кориснику. На пример, може се извршити претраживање и прикупљање информација са различитих веб-страница, без приказа слика, дугмади или падајућих менија на екрану. Иако се ови елементи не приказују, и даље је могуће навигирати између веб-страница, кликнути на било који елемент и извршавати сличне акције.

За коришћење *headless* режима у Пајтон коду са библиотеком *Selenium*, потребно је конфигурисати драјвер за одговарајући веб-прегледач и поставити опцију за *headless* извршавање, што је приказано у коду на листингу 3.5

```
1 from selenium.webdriver.chrome.options import Options
2 options = Options()
3 options.add_argument('--headless')
4 driver = webdriver.Chrome(service=service, options=options)
```

Listing 3.5: Омогућавање *headless* режима

Имплицитно и експлицитно чекање

Постоје два основна метода чекања у оквиру библиотеке *Selenium*: имплицитно чекање и експлицитно чекање. Обе методе се користе како би се осигурало да се одређена радња изврши тек након што се испуни одређени услов, као што је приказивање одређеног елемента на веб-страници или завршетак одређене акције.

Експлицитно чекање је доступно у оквиру библиотеке *Selenium* за императивне програмске језике и омогућава коду да заустави извршавање програма или замрзне нит све док се не испуни услов који му се преда. Услов се проверава са одређеном учесталошћу све док се не истакне време чекања. То значи да ће, све док услов не врати вредност *falsy*, покушавати и чекати [7].

Модул *WebDriverWait* омогућава чекање одређеног временског периода док се одређени услови не испуне на веб-страници. За инстанцирање класе *WebDriverWait* неопходно је проследити два аргумента у конструктор: инстанцу објекта *WebDriver* (који представља веб-драјвер за аутоматско управљање Веб-прегледачем) и време чекања у секундама. Затим се може употребити метод *until* објекта *WebDriverWait* са прослеђеним аргументом који представља жељени услов који треба да се испуни. На пример, код који је приказан на листингу 3.6 чека да се дугме на локацији датог израза *XPath* учини кликабилним пре него што се настави са извршавањем кода.

```
1 reject_cookies_btn = WebDriverWait(driver,10).until(EC.
    element_to_be_clickable((By.XPATH, "//div[contains(@class, '
    ButtonPrimitiveContentChildren') and contains(text(), 'Reject all')
    ]")))
2 reject_cookies_btn.click()
```

Listing 3.6: Кликтање на елемент

Модул *expected_ conditions* садржи различите услове који проверавају одређене карактеристике елемената на веб-страници. На пример, за проверу да ли је одређен елемент кликабилан може да се искористи метод *expected_ conditions.element_ to_ be* а за проверу да ли је елемент видљив на страници може да се искористи метод *expected_ conditions.visibility_ of_ element_ located*. Ови услови се користе у комбинацији са објектом модула *WebDriverWait* како би се сачекали одређени услови пре него што се настави са извршавањем кода. Коришћење оба модула показало се јако корисно у случају постојања интерактивних елемената на страницама које се динамички учитавају или ако је неопходно проверити одређене карактеристике пре него што се настави са прикупљањем података са веб-странице.

Имплицитно чекање се поставља само једном и примењује глобално на све радње које извршава драјвер. Када се користи имплицитно чекање, драјвер ће чекати одређено време пре него што баци изузетак *ElementNotVisibleException* или *NoSuchElementException* уколико не може пронаћи елемент. Имплицитно чекање подразумева да *WebDriver* периодично претражује *DOM* у одређеном временском периоду када покушава да пронађе било који елемент. Ово може бити корисно када одређени елементи на веб-страници нису одмах доступни и захтевају неко време да се учитају [7].

У оквиру библиотеке *Selenium* такође постоји и такозвани *FluentWait*. Инстанца *FluentWait* дефинише максимално време чекања на услов, као и учесталост провере услова [7].

Лоцирање елемената

Библиотека *Selenium* дефинише два главна метода за ефикасно 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 ова метода прихватају аргумент у облику стратегије лоцирања елемента. Стратегија лоцирања одређује на који начин ће се елемент пронаћи на веб-страници. Неке од често коришћених стратегија су:

ID — Лоцирање елемента по јединственом идентификатору.

NAME — Лоцирање елемента по његовом имену атрибута.

XPATH — Лоцирање елемента помоћу израза *XPath* који пружа путању до елемента.

TAG_NAME — Лоцирање елемента по називу ознаке.

CLASS_NAME — Лоцирање елемента по називу *CSS* класе.

CSS_SELECTOR — Лоцирање елемента помоћу *CSS* селектора.

Помоћу ових стратегија за лоцирање елемената, могуће је тачно идентификовати жељене елементе на веб-страници и извршити различите операције над њима. Једна од операција може бити кликтање на елемент и то је приказано у коду на листингу 3.6. Такође, могуће је изабрати опцију из падајуће листе користећи методе *select_by_visible_text()* или *select_by_value()* уз употребу класе *Select*. Приказан код на листингу 3.7 проналази падајућу листу за избор валуте на веб-страници, чека да буде кликабилна и одабира опцију са вредношћу *eur* из те листе.

```
1 currency_dropdown_wait = WebDriverWait(driver, 10).until(EC.
    element_to_be_clickable((By.XPATH, "//select[@data-test='
    CurrencySelect']")))
2 currency_dropdown = Select(currency_dropdown_wait)
3 currency_dropdown.select_by_value('eur')
```

Listing 3.7: Одабир опције из падајуће листе

.

Читање садржаја елемента са веб-странице се може извршити користећи методу *text*. Код приказан на листингу 3.8 чита вредност која представља трајање путовања у оба правца.

```
1 duration_xpath = '//div[@data-test="TripDurationBadge"]'
2 departure_duration.append(info[0].find_element(By.XPATH, value=
    duration_xpath).text)
3 return_duration.append(info[1].find_element(By.XPATH, value=
    duration_xpath).text)
```

Listing 3.8: Читање садржаја елемента

.

Прикупљање података са више веб-страница

Начин имплементације за прикупљање свих жељених података зависи од структуре конкретног *HTML* кода веб-странице. Постоји могућност да веб-страница користи пагинацију или бесконачан скрол.

За веб-сајт који користи пагинацију, подаци се могу прикупити на следећи начин:

1. Учитати почетну страницу.
2. Идентификовати елементе који садрже жељену информацију и прикупити податке са веб-странице.
3. Проверити да ли постоји навигациони елемент за прелазак на следећу страницу.
4. Ако постоји, извршити клик на навигациони елемент за прелазак на следећу страницу.
5. Сачекати да се учита следећа страница.
6. Поновити кораке 2—5 све док се не прикупе подаци са свих страница у пагинацији.

Кључни корак у овој имплементацији је итерирање кроз све странице пагинације, прикупљање података са сваке странице и прелазак на следећу страницу све док се не прикупе сви жељени подаци.

За веб-сајт који користи бесконачан скрол, подаци се могу прикупити на следећи начин:

1. Учитати почетну страницу.
2. Идентификовати елементе који садрже жељену информацију и прикупити податке са веб-странице.
3. Извршити скрол на дно веб-странице користећи функционалности библиотеке *Selenium*.
4. Сачекати да се учитају нови подаци.
5. Поновити кораке 2—5 све док се не прикупе сви подаци.

Кључни корак у овој имплементацији је непрекидно скроловање на дно странице и прикупљање података који се динамички учитавају.

Када је у питању веб-страница *kiwi* при учитавању летова за сваку дестинацију, постоји скрол и дугме за учитавање. У коду на листингу 3.9 је приказана функција под називом *load_more* која је одговорна за учитавање више летова притиском на дугме *„Load more"* на веб-страници. Унутар петље *while*, проверава се број покушаја пролиставања *scroll_attempts* и ако је мањи од *MAX_SCROLL_ATTEMPTS*, наставља се са покушајем учитавања. Користи се *WebDriverWait* да се пронађе и кликне дугме *„Load more"*. Затим се користи класа *ActionChains* да би се померио курсор миша на дугме пре клика. Након клика, функција чека паузу од 5 секунди да би се нови летови учитали на страници. Број покушаја *scroll_attempts* се повећава за 1. Ако се деси *TimeoutException*, што означава да дугме *„Load more"* више није кликабилно или није пронађено, функција прекида петљу користећи наредбу *break*. Ово омогућава излазак из петље и наставак извршавања других делова кода. Ова функција је корисна за учитавање додатних летова на страници, тако да корисник може пролистати и прикупити све доступне информације о летовима.

```
1  def load_more(driver):
2      wait = WebDriverWait(driver, 20)
3      scroll_attempts = 0
4
5      while scroll_attempts < MAX_SCROLL_ATTEMPTS:
6          try:
7              load_more_btn = wait.until(EC.element_to_be_clickable((By.
       XPATH, "//div[contains(@class, 'ButtonPrimitiveContentChildren')
       and contains(text(), 'Load more')]")))
8              actions = ActionChains(driver)
9              actions.move_to_element(load_more_btn).perform()
10             load_more_btn.click()
11             time.sleep(5)
12             scroll_attempts += 1
13         except exceptions.TimeoutException as e:
14             break
```

Listing 3.9: Учитавање елемената на веб-страници и скрол

.

Изазови аутентикације и аутоматизације

При веб скрејпингу, изазови аутентикације и аутоматизације односе се на проблеме који се јављају приликом приступа и пријаве на веб странице. Веб странице захтевају аутентификацију корисника, обично путем корисничког имена и лозинке, пре него што дозволе приступ одређеним подацима. Уколико није успешно извршена пријава на веб страницу, приступ циљаним подацима је обично онемогућен. Да би спречиле аутоматизовани приступ и веб-скрејпинг, веб странице могу користити различите технике, као што је *CAPTCHA*. Ове мере могу онемогућити успешну пријаву приликом веб скрејпинга.

У коду на листингу 3.10 је приказано како да се превазиђе проблем пријављивања на веб-страницу користећи програмски језик Пајтон и библиотеку *Selenium*. Код аутоматски попуњава поља за унос корисничког имена и лозинке на веб-страници користећи функцију *send_keys*. Након што су унети подаци, неопходно је искористити функцију *send_keys(Keys.ENTER)* како би се симулирао притисак тастера *Enter* и послала форма за пријаву.

```
1 from selenium.webdriver.common.keys import Keys
2
3 # Find username and password inputs
4 username_field = driver.find_element(By.ID, value="username")
5 password_field = driver.find_element(By.ID, value="password")
6
7 # Enter user name and password
8 username_field.send_keys("your_username")
9 password_field.send_keys("your_password")
10
11 # Submitting the login form
12 password_field.send_keys(Keys.ENTER)
```

Listing 3.10: Пријављивање на веб-страници

.

3.3 Библиотека *Scrapy*

TODO: uvod u scrapy Библиотека *Scrapy* је библиотека специфично дизајнирана за веб-скрејпинг, која пружа скуп алата и функционалности како би се олакшао процес прикупљања података са веб-страница.

Инсталација

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

```
pip3 install scrapy
```

Након успешне инсталације, потребно је креирати *Scrapy* пројекат. За креирање пројекта, треба се преференцијално позиционирати у жељени директоријум у оквиру терминала и извршити следећу команду која ће аутоматски генерисати почетне директоријуме и датотеке које су потребне.

```
scrapy startproject project_name
```

Креирање паука

Паук (енг. *Spider*) је кључна компонента у библиотеци *Scrapy* која се користи за дефинисање логике веб-скрејпинга и извлачења података са веб-страница. Паук представља Пајтон класу која наслеђује класу *scrapy.Spider* која садржи методе и атрибуте потребне за прикупљање података са веб-страница.

За креирање паука неопходно је следити кораке:

1. Креирати датотеку унутар иницијално креираног директоријума *spiders*.
2. У креирану датотеку импортовати потребне модуле, као што је библиотека *Scrapy*, следећом наредбом:

   ```
   import scrapy
   ```

3. Дефинисати класу паука. Класа мора да наследи класу *scrapy.Spider*. TODO....

TODO: 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] Ryan Mitchell. Web scraping with python. In *Web Scraping with Python*, 2015.

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

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

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

[13] 2000-2010 Carnegie Mellon University. CAPTCHA. on-line at: `http://www.captcha.net/`.

Биографија аутора

Зорана Гајић, рођена је 04.11.1997. у Москви, где је завршила први разред основне школе, због чега је по повратку у Београд наставила и завршила основно и средње образовање у Руској школи при Амбасади Руске Федерације са одликованом златном медаљом од стране Руске Федерације за посебна достигнућа у настави. Смер Информатика на Математичком факултету Универзитета у Београду уписала је 2015. године, а завршила у јулу 2019. године са просечном оценом 8.8. Након завршених основних студија, уписала је мастер студије информатике на истом факултету.

Од септембра 2020. године је запослена у компанији *Smart Apartment Data* где ради у фронт-енд тиму на изради апликације која нуди поуздан извор тржишне интелигенције за стамбену индустрију. Нуде свеобухватне платформе података за власнике, брокере, компаније, тимове и добављаче којима су потребне тачне детаљне информације за пословне одлуке и информисана улагања. Тренутно ради на позицији вође фронт-енд тима у истој фирми.