Prikupljanje podataka s Weba u programskom jeziku Python

Fabijanić, Dorijan

Sveučilište u Rijeci – Odjel za informatiku

Preddiplomski dvopredmetni studij filozofije i informatike

Dorijan Fabijanić

Prikupljanje podataka s weba u programskom jeziku Python

Završni rad

Mentor: izv. prof. dr. sc. Ana Meštrović

Rijeka, Svibanj 2018. godine

Sadržaj

1. Uvod

Godine 1989. Tim Berners – Lee napisao je utjecajan znastveni rad koji je objavljen pod nazivom: „Information Management: A Proposal“ te koji je postao temelj za jedan od najvažnijih izuma 20. stoljeća – WWW (World Wide Web). Tim Berners – Lee dizajnirao je WWW na način da on ne bude vlasnički sustav kojim će upravljati jedan centralni autoritet nego da bude neutralan, univerzalan, decentralizirani sustav kojem će svi jednako moći pristupati i na kojem će svi moći objavljivati podatke bez bilo kakve dozvole [1]. Tim Berners – Lee jednom je dao svoje mišljenje o tome zašto je Internet u tako kratkom vremenu postao toliko popularan [1]:

> „Da je tehnologija bila vlasnička i pod mojom potpunom kontrolom, vjerojatno nikada ne bi zaživjela. Ne možete predložiti da nešto bude univerzalan prostor, a da pritom istovremeno držite nadzor nad njime“.

Dakle, na Internetu su odmah od početka, svi ljudi mogli objavljivati podatke kada god i u kojoj god količini su to htjeli. To je rezultiralo enormnom količinom nestrukturiranih podataka koji su se sada svi nalazili na jednom mjestu. S vremenom je postalo jasno kako ta velika količina podataka može biti korisna ukoliko bi se mogla automatski prikupljati i strukturirati te se kao posljedica razvila tehnologija web struganja. Osnovna ideja web struganja sastoji se od četiri faze [2]:

1) Prikupljanje HTML podataka s web domene
2) Parsiranje prikupljenih podataka s ciljem lociranja tražene informacije
3) Spremanje tražene informacije
4) Opcionalno, prelaženje na drugu web domenu i ponavljanje procesa.

Najučestaliji način korištenja Interneta je zasigurno putem web preglednika. web preglednici kao što su Chrome, Firefox, Opera prikazuju moderno dizajnirane stranice izvodeći JavaScript, prikazujući multimediju i pružajući najbolje moguće iskustvo za korisnika. S druge strane korištenjem web strugača otvara se mogućnost preuzimanja vrlo velikih količina podataka s velikog broja web stranica te spremanje istih u baze podataka. Nakon toga, ti se podatci mogu analizirati te iz njih možemo dobiti nova znanja o raznim djelatnostima.

Python je jezik koji ima dobre mogućnosti za prikupljanje podataka s weba pa ću se stoga koristiti primarno njime. Postoji nekoliko biblioteka i okvira koji omogućavaju da se

programski jezik Python koristi za prikupljanje podataka s weba [3]. Postoje razne mogućnosti spremanja podataka nakon što se podaci prikupe i parsiraju. Podaci se mogu pohraniti u CSV datoteke ili u bazu podataka. Postoji velik broj različitih tipova baza podataka koji se mogu koristiti za pohranu prikupljenih podataka, a ja ću u ovom završnom radu koristiti MongoDB [4]. Sve nabrojane biblioteke imaju određenu, vrlo specifičnu ulogu u softveru za prikupljanje podataka s weba pa je zbog toga softver za prikupljanje podataka moguć samo njihovim kombiniranjem. U ovom završnom radu predstavit ću i dva alata koji su multifunkcionalni, odnosno u sebi imaju mehanizme za prikupljanje, parsiranje i pohranu podataka, a to su okvir za web struganje: „Scrapy“ i skup alata za automatizaciju web preglednika: „Selenium“.

U ovom završnom radu napisao sam nekoliko programa koji koriste prethodno navedene tehnologije. Svi programi su tematski povezani i implementirani na različitim web stranicama koje koriste različite tehnologije. Cilj istraživanja je bio prikazati prednosti i nedostatke svakog od navedenih alata na način da ih primjenim na nekoliko različitih stranica koje se bavi istom djelatnošću. U prvim poglavljima rada usporedit ću biblioteke Beautiful Soup i lxml. Nakon toga ću prikazati na koji sve način možemo pohraniti prikupljene podatke, a na poslijetku ću na nešto kompleksnijim projektima prikazati funkcionalnosti okvira Scrapy i Selenium.

2. Pregled alata za prikupljanje podataka s weba

U ovom poglavlju je dan pregled alata koji se koriste za prikupljanje podataka s weba. Python ima velik broj alata koji se koriste za prikupljanje podataka s weba počevši od biblioteka koje se koriste za specijalizirane zadatke poput biblioteka koje su zadužene za slanje HTTP zahtjeva i parsiranja HTML kôd pa sve do gotovih okvira koji sadrže sve te funkcionalnosti.

2.1. Python

Programski jezik Python jedan je od najpopularnijih opcija za prikupljanje podataka s weba. Python je objektno – orijentirani programski jezik otvorenog kôda koji je po svojoj sintaksti sličan programskom jeziku PERL. Ovaj programski jezik ima čistu i jasnu sintaksu te ga krasi karakteristika čitljivosti [5]. Jedna od velikih prednosti programskog jezika Python je velik broj biblioteka koje su besplatno dostupne korisnicima pa tako ima i velik broj biblioteka za prikupljanje podataka s weba. Neke od najpoznatijih biblioteka za prikupljanje podataka s weba su prethodno spomenute BeautifulSoup i lxml. U ovome završnom radu koristit ću obje biblioteke te ukazati na njihove prednosti i mane. U nastavku ovog seminarskog rada koristit ću 3.x verziju programskog jezika Python [6].

2.2. Beautiful Soup

Beautiful Soup je biblioteka koja služi za parsiranje HTML i XML datoteka. Surađuje s parserom te stvara idiomatski način navigacije, pretraživanja i izmjenjivanja stabla parsiranja. BeautifulSoup biblioteka dobila je ime po istoimenoj pjesmi Lewisa Carrolla u njegovom djelu „Alisa u Zemlji Čudesa“. Poput pjesme iz ovog poznatog književnog djela, koja nastoji pronaći smisao u besmislu, BeautifulSoup biblioteka nastoji pronaći smisao u velikim, nestrukturiranim, često besmislenim bespućima weba na način da popravlja loše napisan HTML kôd i predstavlja nam pregledne XML strukture [2].

2.3. lxml

lxml je jedna od najbogatijih i najjednostavnijih biblioteka za procesiranje HTML i XML datoteka u programskom jeziku Python. Kako popularnost programskog jezika Python i XML drastično raste, postoji pregršt biblioteka koje se bave prikupljanjem podataka iz XML datoteka baš kao i lxml. Međutim, lxml ima nekoliko prednosti u odnosu na ostale biblioteke.

Prva prednost biblioteke lxml je njena brzina izvođenja: čitanje i pisanje XML datoteka obavlja se u vrlo kratkom vremenu. Druga prednost je ta što lxml ima jednostavniju sintaksu od ostalih Python biblioteka te se samim time lakše uči i brže piše programski kôd [7].

2.4. Requests

Requests je HTTP biblioteka koja koristi Apache2 licencu te je napisana u programskom jeziku Python. Dizajnirana je kako bi je koristili ljudi i pomoću nje komuniciraju s jezikom. Drugim riječima, ova biblioteka automatizira proces slanja HTTP/1.1 zahtjeva koristieći Python [8].

2.5. Selenium

Selenium je skup alata koji služe za automatizaciju web preglednika. Selenium za razliku od ostalih alata kojima ću se koristiti nije primarno razvijen u svrhu prikupljanja podataka s weba, ali se sa sve većom pojavom dinamički generiranih web stranica sve više počeo integrirati u softver za prikupljanje podataka s weba zbog mogućnosti izvođenja Javascript kôda [9].

2.6. Scrapy

Scrapy je okvir otvorenog kôda za izvlačenje podataka s web stranica. Scrapy je aplikacijski okvir koji radi u suradnji s bibliotekama za parsiranje HTML i XML datoteka poput BeautifulSoup i lxml. Ovaj okvir u sebi ima ugrađeni mehanizam za izvlačenje podataka s weba, ali ne mora ga se nužno koristiti već se u njega mogu uključiti prethodno spomenute biblioteke te se mogu koristiti njihovi mehanizmi. Najkorisnije svojstvo okvira Scrapy je pisanje „web pauka“ koji odlaze na stranice i izvlače podatke. Ti web pauci mogu se potom spremati i pokretati na Cloudu ili na vlastitom web serveru. U ovome seminarskom radu prikazati ću korisnost ovog okvira za web struganje na cjelovitom projektu [10].

2.7. MongoDB

Za pohranu podataka koristit ću MongoDB. MongoDB je dokumentacijska baza podataka koja pohranjuje podatke u JSON datotekama što znači da podaci mogu biti različiti u svakom dokumentu i da se struktura baze podataka može konstanto mijenjati. Baza napisana u

MongoDB tehnologiji ne zahtjeva od dizajnera da unaprijed definira fiksnu shemu što omogućuje dodavanje i brisanje polja po potrebi. To je najveća razlika između dokumentacijskih baza podataka i onih relacijskih, a ujedno je i primarni razlog zbog kojeg mnogi dizajneri baza podataka prelaze na tehnologije poput MongoDB. Osim toga, koncept reda u dokumentacijskim bazama podataka ne postoji te se zamjenjuje fleksibilnijim modelom kojeg se naziva „dokument“. Dokument dopušta da se u njega unose drugi dokumenti i polja podataka što omogućuje prikazivanje složenih hijerarhijskih odnosa s jednim unosom. Razvijanje baze podataka u MongoDB ubrzava razvoj softvera te je s ovom tehnologijom jednostavnije eksperimentirati. MongoDB je besplatan i otvorenog je kôda te je izdan pod GNU Affero licencom [11].

3. Uvod u web struganje

Web struganje je tehnologija koja, ukoliko se pravilno implementira, daje bolji uvid u veliku količinu podataka na način da se prikupljeni podaci s web stranica strukturiraju i pohrane u spremnik, na primjer u bazu podataka. Implementiranje web strugača korisno je u području podatkovnih znanosti, posebice na području velikih podataka gdje se iz mnoštva nestrukturiranih informacija na webu može doći do novih saznanja. Osim toga, ova tehnologija se također često implementira u poslovanju tvrtki koje moraju pratiti, na primjer, cijene ili proizvode na tržištu zbog toga što se njenim korištenjem automatizira proces pregleda cijena i svojstava predmeta te za to ne treba biti zadužena fizička osoba. To znači da osoba ne mora više sama odlaziti na web stranice koje sadrže podatke o tim cijenama već program sve obavlja sam, od preuzimanja i parsiranja HTML datoteke do pohranjivanja podataka.

Osim u poslovne svrhe, prikupljanje podataka s weba može biti korisno i u osobne svrhe. Mogućnost pregleda cijena artikala sa različitih web stranica kako bi se ustanovilo koja je najpolovoljnija, najočitiji je primjer korisnosti tehnologija za web struganje. U današnje vrijeme, video igre se sve manje pohranjuju na CD i DVD optičke diskove te su uglavnom postale digitalni medij što znači da se većinski dio transakcija odvija putem weba. Iz tog razloga, cijene često variraju, a proizvodi se prodaju na mnogo različitih stranica. U nastavku ovog seminara napravit ću program koji će prikupljati podatke o cijenama s nekoliko najpoznatijih stranica za kupnju digitalnih video igara te ih spremati u bazu podataka koja će

omogućiti korisniku da lakše i efikasnije pregledava cijene i time donosi bolje odluke o kupnji.

Kako bismo mogli parsirati HTML datoteke uz pomoć biblioteka Beautiful Soup ili lxml, te iz njih prikupljati podatke o cijenama, potrebno ih je prvo preuzeti. Najjednostavniji i najefikasniji način za preuzimanje HTML ili XML datoteka je koristeći biblioteku Requests.

```
import requests
```

Pomoću ove biblioteke moguće je poslati sve vrste HTTP zahtjeva od osnovnih *get* i *set* zahtjeva do ostalih poput *put*, *delete*, *head* i *options* zahtjeva. Svi oni pozivaju se na sličan način.

```
r = requests.get('https://api.github.com/events')
r = requests.head('http://httpbin.org/get')
r = requests.options('http://httpbin.org/get')
r = requests.post('http://httpbin.org/post', data = {'key':'value'}) [8]
```

Ukoliko iskoristimo jednu od ovih metoda, HTML stranica pohranit će se u varijablu *r*. Međutim ukoliko napišemo samo tu jednu liniju kôda, a pojavi se greška na koju mi ne možemo utjecati, skripta će se srušiti i neće prikupiti podatke sa stranice, a neće nam ni dati povratnu informaciju. Zbog toga je potrebno uvesti iznimku koja će se izvesti ukoliko se neka greška pojavi. Svi podaci o greškama nalaze se u *requests.exceptions.RequestException* modulu.

```
website_url =
'https://www.newegg.com/Product/ProductList.aspx?Submit=ENE&IsNodeId=1&N=10002
1392%20600489407%20600489408%20600538759%20600560190&name=Steam-PC-Download'
page_html = ''
try:
    r = requests.get(website_url)
    page_html = r.text
    r.close()
except requests.exceptions.RequestException as e:
    print ('Error: ' + str(e))
    r = None
```

U ovom slučaju, ukoliko dođe do greške zbog koje se stranica neće otvoriti, javit će nam o

kojoj se vrsti greške radi, a *r* dobiva vrijednost „*None*“ [8]. Nakon izvođenja kôda u varijabli *r* je spremljena HTML datoteka.

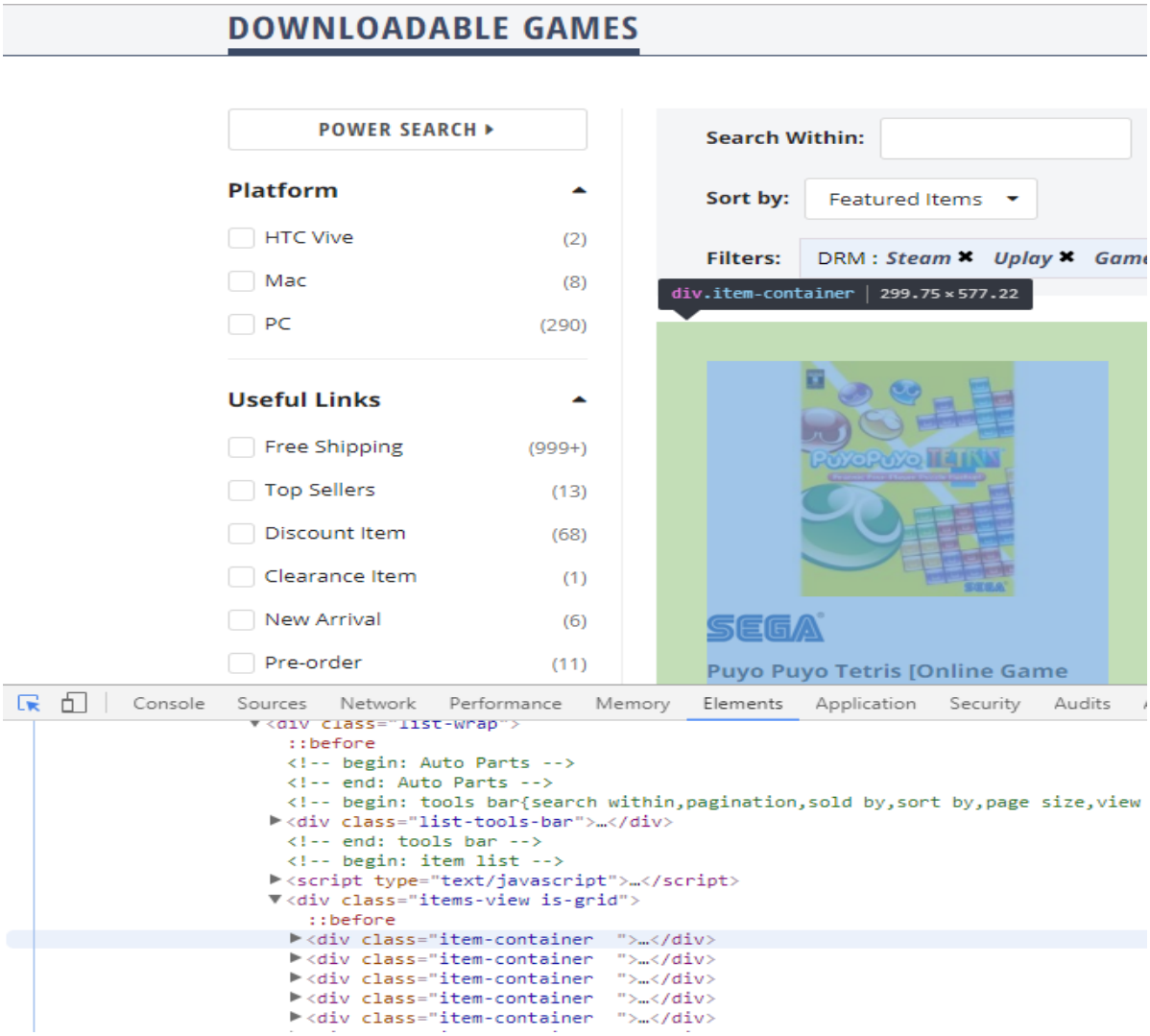
Nakon što se HTML datoteka uspješno spremi u varijablu može se započeti sa samim parsiranjem. Parsiranje HTML datoteke u programskom jeziku python moguće je korištenjem lxml ili Beautiful Soup biblioteka. U ovom primjeru koristit ću Beautiful Soup biblioteku te ću započeti s njenim uključenjem.

```
from bs4 import BeautifulSoup
```

HTML kôd koji sam pohranio u varijablu je kôd preuzet s web stranice www.newegg.com. Proces parsiranja u računarstvu predstavlja razlaganje većeg bloka podataka, prema određenim pravilima, na manje djelove kako bi ga računalo moglo jednostavnije i brže interpretirati, oblikovati i prenositi [12]. Kako bismo mogli upravljati podacima dostupnim unutar preuzetog HTML kôda, potrebno ih je parsirati. Kako bih to uspio, moram pozvati biblioteku BeautifulSoup koju sam prethodno uključio u program te parsirane podatke spremiti u varijablu. BeautifulSoup prihvaća dva argumenta. Prvi argument je HTML datoteka koju pokušavamo parsirati, a drugi argument je string *html.parser* kojim označavamo da se radi o HTML datoteci, a ne o XML ili nekoj drugoj vrsti datoteke.

```
parsed_html = BeautifulSoup(page_html, 'html.parser')
```

Sljedeći korak je pregledavanje izvornog kôda iz kojeg je potrebno iščitati u kojim se oznakama nalaze traženi podaci. Na web stranici www.newegg.com svaki artikl nalazi se u oznaci div kojoj je pridružena klasa *item-container*. To znači da ću svim podacima artikla moći pristupiti, ako pronađem sve instance *item-container* klase.



Slika 1. Prikaz klase item-container koja sadrži podatke o proizvodima

Dohvatiti sve oznake div s klasom *item-container* moguće je koristeći petlju, ali Beautiful Soup u sebi ima nekoliko ugrađenih funkcija koje automatiziraju ovaj proces. Funkcija *findAll()* omogućava automatizaciju procesa prikupljanja svih istoimenih klasa. Prvi od dva parametra koje ću proslijediti funkciji je HTML oznaka, u ovom slučaju div, a drugi parametar je klasa koja dodatno opisuje oznaku div te ga je nužno proslijediti kako ne bih preuzeo druge div oznake koji ne sadrže podatke koji su mi potrebni.

```
item_containers = parsed_html.find('div', {'class':'item-container'})
```

Kako bi softver za praćenje cijena video igara bio koristan potrebni su mi ime video igre, cijena, popust ukoliko postoji i poveznica za kupnju. Sve te informacije se nalaze u svakoj div oznaci, preciziranoj klasom *item-container*. Sada su podaci o video igrama pohranjeni u varijabli *itemContainers* u obliku polja. Svaka igra zauzima jedno mjesto u polju. Kako bih

pristupio svakoj stavki pojedinačno, kako bih dobio pojedinačne podatke, koristit ću for petlju. Prvo je potrebno definirati jedan zaseban itemContainer u polju itemContainers koji sadrži sve *item-container* klase u HTML datoteci, a nakon toga proći kroz svaki pojedinačno koristeći petlju.

```
item_container = item_containers[0]
for item_container in item_containers:
```

Sljedeći korak je pronaći sve informacije koje su mi potrebne u svakoj instance klase. Funkciju *findAll*() sam već prethodno prikazao, ali također postoji još jedan način pronalaženja podataka. Ovakav način pretrage brži je od pretraživanja funkcijom *findAll*(), ali od programera iziskuje više čitanja HTML kôda. Naime, podacima možemo pristupiti i na način da vidimo gdje je u strukturi datoteke ugnježđena HTML oznaka te upisujemo putanju.

```
<div class="item-action">
    <ul>
        <li>
            <a class="item-img" href="https://www.newegg.com/Product/
            Product.aspx?Item=N82E16832203313"></a>
        </li>
</div>
```

U prethodnom primjeru poveznicu iz oznake *a* mogu preuzeti koristeći findAll() funkciju:

```
example.findAll('a', {'class':'item-img'})
```

Isti rezultat mogu dobiti i na ovaj način:

```
example.div.ul.li.a['href']
```

Cilj ovog projekta bio je prikupiti podatke o nazivu, cijeni, popustu i poveznici računalne igre te kako bih to uspio upisat ću sljedeće naredbe u for petlju:

```
game_title = item_container.findAll('a', {'class':'item-title'})
game_price = item_container.findAll('li', {'class':'price-current'})
game_discount = item_container.findAll('span', {'class':'price-save-
percent'})
game_link = item_container.a['href']
```

4. Pohrana podataka

Sada su svi traženi podaci pohranjeni u varijable u Pythonu. Kako bi ih pregledavao ili njima upravljao raspoložive su tri mogućnosti.

4.1. Ispis podataka u konzolu

Najjednostavniji način pregleda podataka je ispisivanjem podataka u konzolu koristeći funkciju *print()* unutar iste for petlje korištene za prikupljanje podataka.

```
print ('NAME: ' + str(game_title[0].text))
    try:
        print('PRICE: ' + str(game_price[0].strong.text +
str(game_price[0].sup.text)))
    except AttributeError:
        print('PRICE: NOT AVAILABLE')
    try:
        print('DISCOUNT: ' + str(game_discount[0].text))
    except IndexError:
        print('DISCOUNT: NOT AVAILABLE')
    print('LINK: ' + str(game_link))
```

Prilikom ispisa podataka može doći do pojedinih grešaka povezanih sa strukturom HTML podataka. Na primjer na web stranici www.newegg.com s koje prikupljam sve podatke neki artikli imaju napisanu cijenu dok neki, koji nisu raspoloživi nemaju. Prilikom ispisa cijene artikala, program će izbaciti grešku – *AttributeError*. Do njega dolazi jer pokušavamo ispisati podatak kao string, a on ima vrijednost *None*. Kako bih izbjegao grešku i rušenje programa primjenio sam na početku navedeni try i except pristup. Sada će program pokušati ispisati cijenu video igre, a ukoliko ona nije navedena i dođe do AttributeError poruke, ispisat će se da cijena nije navedena i program će nastaviti izvoditi sljedeće linije kôda.

Slika 2. Ispis podataka u konzolu

4.2. CSV datoteke

CSV (engl. Comma-Separated Values) datoteke su datoteke u koje se spremaju tabularni podaci (tekst i brojevi) u tekstualnom obliku. Svaka pojedina linija kôda predstavljan jedan zapis. Svaki se zapis sastoji od jednog ili više polja koja su odvojena zarezima. Problem CSV datoteka je što format nije standardiziran. Osnovna ideja je jednostavna, svaki podatak je odvojen zarezima, ali datoteka postaje nejasna ukoliko podaci isto tako sadrže zareze ili ukoliko se moraju prikazati u više redaka [13].

U ovom programu, nova CSV datoteka kreirat će se prvi puta i izmjeniti svaki sljedeći put kada se program pokrene. Datoteku ću kreirati koristeći *open()* funkciju koja se izvorno nalazi u programskom jeziku Python. Funkcija *open()* otvara datoteku i vraća odgovarajući objekt. Kako bi tablica bila pregledna i jasna korisnicima, potrebno je upisati naslove svakog stupca u tablicu. Kako bismo unijeli naslove u tablicu dovoljno ih je napisati kao jedan string te odvojiti zarezom. CSV datoteka zarez čita kao prijelaz u novi stupac. Ovo je potrebno napraviti prije petlje jer se taj tekst mora pojaviti samo u prvom retku.

```
filename = 'videoGames.csv'
fOpen = open(filename, 'w')
headers = 'NAME, PRICE, DISCOUNT, LINK \n'
fOpen.write(headers)
```

Za upis podataka definirao sam funkciju *print_to_csv*() koja kao argumente prima podatke koje sam prethodno prikupio: naslov, cijenu, popust i link na stranicu. Isto kao i naslove tablice potrebno je argumente odvojiti zarezom kako bi se svaki podatak zapisao u zasebni stupac.

```
def print_to_csv(title, price, discount, link):
    fOpen.write(title.replace(',', '|') + ',' + price + ',' + discount + ',' +
link + '\n')
```

Kod naslova sam iskoristio funkciju *replace()* jer neki naslovi video igara sadrže zarez u imenu pa se zbog toga jedan dio naslova upiše u jedan stupac, a drugi dio naslova u drugi stupac. Funkcija replace prima dva argumenta gdje drugi argument mijenja prvi. CSV datoteku moguće je otvoriti u Microsoft Excelu i sličnim programima te se zbog toga može brzo formatirati kako bi bila pregledna.

NAME	PRICE	DISCOUNT	LINK
FINAL FANTASY XV Windows Edition [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832166309
Human: Fall Flat [Online Game Code]	7.49	50%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832847005
Puyo Puyo Tetris [Online Game Code]	16.99	15%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832203313
Warhammer: Vermintide 2 [Online Game Code]	24.99	17%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832982020
Final Fantasy XIV: Stormblood PC [Game Download]	29.99	25%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832166257
Sid Meier's Civilization VI [Online Game Code]	49.99	17%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832205315
Grand Theft Auto V : Criminal Enterprise Starter Pack [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832137090
Prey [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832136181
BATTLETECH - Digital Deluxe Edition [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832021305
Northgard [Online Game Code]	19.99	33%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832843012
Chrono Trigger Limited Edition [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832166313
Gravel [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832698003
Cities: Skylines [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832123025
Grand Theft Auto V [PC Download] with GTA Online	49.99	17%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832137064
Injustice 2 [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832777213
Middle Earth: Shadow of War - Gold Edition [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832777185
Fallout 4 VR [Online Game Code]	51.99	13%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832136202
Borderlands 2 Game of the Year Edition [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832205123
Vampyr [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832844075
Nier: Automata [Online Game Code]	49.99	17%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832166272
Bayonetta [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832203281
Tom Clancy's The Division [Online Game Code]	41.99	16%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832138398
LA Noire: The VR Case Files [Online Game Code]	24.99	17%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832205425
Life is Strange: Before the Storm Deluxe Edition [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832166279
WORLD OF FINAL FANTASY [Online Game Code]	32.99	18%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832166299
LEGO Worlds [Online Game Code]	14.99	50%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832777179
Kerbal Space Program [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832205414
Wolfenstein II: The New Colossus [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832136198
Alien: Isolation: The Collection [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832203048
XCOM 2: Collection [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832205427
Warhammer 40|000: Dawn Of War III [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832203219
Football Manager Touch 2018 [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832203302
Endless Space 2 [Online Game Code]	NOT AVAILABLE	NOT AVAILABLE	https://www.newegg.com/Product/Product.aspx?Item=N82E16832203280
Sonic Mania [Online Game Code]	16.99	15%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832203284
Fallout 4 [Online Game Code]	25.99	13%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832136086
XCOM 2: War of the Chosen [Online Game Code]	32.99	18%	https://www.newegg.com/Product/Product.aspx?Item=N82E16832205394

Tablica 1. Prikaz podataka u CSV datoteci

4.3. Baza podataka

Najbolji način pohrane podataka koji su prikupljeni s weba je baza podataka. Baza podataka je skup povezanih podataka gdje se pod podacima podrazumjevaju činjenice koje su poznate i imaju implicitno određeno značenje [14]. U ovom konkretnom slučaju podaci koje ću spremati u bazu podataka biti će podaci o računalnim igrama, kao što je prije navedeno: ime, cijena, popust i poveznica na web stranicu na kojoj se računalna igra može kupiti. U ovom projektu kao tehnologiju za razvoj baza podataka koristit ću NoSQL (engl. *No Structured Query Language*) bazu podataka MongoDB. Relacijske odnosno SQL baze podataka razlikuju se od NoSQL baza podataka u tome što je za relacijske baze podataka potrebna

shema koja mora biti zadana unaprijed kako bi se u bazu podataka mogli pohranjivati podaci. Na primjer, ja želim napraviti bazu podataka koja će sadržavati podatke o video igrama te se odlučim za tradicionalnu relacijsku bazu podataka. To znači da moram unaprijed napraviti shemu za bazu i tek tada u nju mogu unositi podatke. Ukoliko nakon nekog vremena želim dodati novi tip podatka u bazu, moram dizajnirati novu shemu, s novim stupcem za tu vrstu podatka te nakon toga preseliti sve podatke iz početne baze podataka u novu, nadograđenu bazu. Što znači da ako u nekom trenutku želim dodati mišljenje korisnika o video igri u bazu podataka uz već postojeće podatke o nazivu, cijeni, popustu i poveznici na web stranicu, ne mogu na postojeću bazu podataka dodati stupac u koji ću upisivati cijene već moram napraviti novu shemu baze podataka s novim stupcem te preseliti sve prethodno prikupljene podatke u nju. Uz to, još jedan razlog zbog kojeg sam se odlučio za korištenje NoSQL baze podataka je što će se podaci u bazi podataka često mijenjati. Kako bi projekt imao smisla onda se podaci o cijenama video igara u bazi podataka moraju mijenjati svaki puta kada se promijene na web stranicama s kojih su prikupljeni. U SQL bazi podataka ovaj proces bi trajao znatno duže nego u NoSQL bazi podataka. Osim toga, kako su podaci koje prikupljam s weba nestrukturirani i nepoznati unaprijed, SQL bazu podataka bilo bi puno teže implementirati. NoSQL baze podataka su napravljene na način da dopuštaju umetanje podataka bez da postoji unaprijed definirana shema što znači da je razvoj baze podataka brži i jednostavniji, brže se ažuriraju podaci i jednostavnije je unošenje nestrukturiranih podataka [15]. MongoDB prihvaća podatke zapisane u JSON formatu. JSON (*engl. JavaScript Object Notation*) je format za razmjenu podataka koji je lako razumljiv te zbog toga pogodan za čitanje i pisanje. Osim što pogoduje ljudima, strojevi ga isto jednostavno parsiraju i generiraju [16]. Dakle, kako je standardizirani zapis podataka MongoDB baza podataka u JSON formatu, moram izmjeniti svoj kôd na način da izlaz bude zapisan u JSON formatu. Napravio sam funkciju *toJson* koja izlazne varijable *title*, *price*, *discount* i *link* u Python Dictionary obliku unosi u listu data funkcijom *append()*, a zatim tu listu pretvara u JSON format te ju zapisuje u datoteku data.json. Funkcija se poziva unutar for petlje u glavnom programu.

```
import json
data = []
def toJson(data, title, price, discount, link):
    data.append({
        'NAME':title,
        'PRICE':price,
```

```
        'DISCOUNT':discount,
        'LINK':link
    })

    with open('data.json', 'w') as outfile:
        json.dump(data, outfile)
```

Lista se prevodi u JSON format funkcijom *dump()* koja je dio json biblioteke u programskom jeziku Python.


```
[{"NAME": "Human: Fall Flat [Online Game Code]", "PRICE": "7.49", "DISCOUNT":
"50%", "LINK":
"https://www.newegg.com/Product/Product.aspx?Item=N82E16832847005"},
{"NAME": "FINAL FANTASY XV Windows Edition [Online Game Code]", "PRICE": "NOT
AVAILABLE", "DISCOUNT": "NOT AVAILABLE", "LINK":
"https://www.newegg.com/Product/Product.aspx?Item=N82E16832166309"},
{"NAME": "Puyo Puyo Tetris [Online Game Code]", "PRICE": "16.99", "DISCOUNT":
"15%", "LINK":
"https://www.newegg.com/Product/Product.aspx?Item=N82E16832203313"}]
```


Svaka stavka u JSON formatu odvojena je od ostalih stavki u listi vitičastim zagradama, a svaki podatak je par koji ima ključ s lijeve strane, a vrijednost s desne. Između ključa i vrijednosti nalazi se dvotočka. Svaka stavka se odvaja zarezom. Sljedeći korak je uvoženje podataka iz datoteke u bazu podataka. Bazu podataka se nakon instalacije pokreće iz command line interpretera. Kako bi se baza podataka pokrenula potrebno je locirati direktorij u kojem je MongoDB instaliran i pokrenuti mongod.exe datoteku. Kako bi se pokrenuo mongo shell potrebno je u istom folderu pokrenuti mongo.exe. U mongo shellu, komandom „*db*“ ispisuje se ime baze podataka koja je trenutno aktivna.

```
> db
videoGames
```

Naredbom *use* nakon koje slijedi ime baze podataka (npr. *use videoGames*) otvara se postojeća ili stvara nova baza podataka ukoliko baza s tim imenom već ne postoji.

```
> use videoGames
switched to db videoGames
```

Kako bi se podaci pohranjivali u aktivnu bazu potrebno je stvoriti kolekciju (engl. *collection*). Kolekciju ću stvoriti funkcijom *createCollection()* kojoj je prvi argument ime kolekcije, a drugi datoteka u kojoj je zapisana konfiguracija kolekcije (npr. db.createCollection(games, opcije)). MongoDB pohranjuje dokumente u kolekcije. Kolekcije u bazama podataka koje nisu relacijske odgovaraju tablicama u relacijskim bazama podataka. Kako bi se datoteka s podacima unijela u bazu podataka koristi se naredba mongoimport s tri parametra, --db koji označava bazu podataka, --collection koja označava kolekciju unutar te baze podataka --file koja označava datoteku koja se unosi. Ispis podataka u kolekciji omogućen je funkcijom *find()*, a ispis se uljepšava funkcijom *pretty()*: *db*.*games*.*find*().*pretty*().

```
> db.games.find().pretty()
{
        "_id" : ObjectId("5aa7df357a4d52b897032440"),
        "NAME" : "Human: Fall Flat [Online Game Code]",
        "PRICE" : "7.49",
        "DISCOUNT" : "50%",
        "LINK" : "https://www.newegg.com/Product/Product.aspx?Item=N82E16832847005"
}
{
        "_id" : ObjectId("5aa7df357a4d52b897032441"),
        "NAME" : "Grand Theft Auto V : Criminal Enterprise Starter Pack [Online Game Code]",
        "PRICE" : "NOT AVAILABLE",
        "DISCOUNT" : "NOT AVAILABLE",
        "LINK" : "https://www.newegg.com/Product/Product.aspx?Item=N82E16832137090"
}
{
        "_id" : ObjectId("5aa7df357a4d52b897032442"),
        "NAME" : "Warhammer: Vermintide 2 [Online Game Code]",
        "PRICE" : "24.99",
        "DISCOUNT" : "17%",
        "LINK" : "https://www.newegg.com/Product/Product.aspx?Item=N82E16832982020"
}
{
        "_id" : ObjectId("5aa7df357a4d52b897032443"),
        "NAME" : "Northgard [Online Game Code]",
        "PRICE" : "19.99",
        "DISCOUNT" : "33%",
        "LINK" : "https://www.newegg.com/Product/Product.aspx?Item=N82E16832843012"
}
{
        "_id" : ObjectId("5aa7df357a4d52b897032444"),
        "NAME" : "Chrono Trigger Limited Edition [Online Game Code]",
        "PRICE" : "NOT AVAILABLE",
        "DISCOUNT" : "NOT AVAILABLE",
        "LINK" : "https://www.newegg.com/Product/Product.aspx?Item=N82E16832166313"
}
{
        "_id" : ObjectId("5aa7df357a4d52b897032445"),
        "NAME" : "Hearts of Iron IV: Waking the Tiger [Online Game Code]",
        "PRICE" : "16.99",
        "DISCOUNT" : "15%",
        "LINK" : "https://www.newegg.com/Product/Product.aspx?Item=N82E16832021308"
}
{
        "_id" : ObjectId("5aa7df357a4d52b897032446"),
        "NAME" : "Gravel [Online Game Code]",
        "PRICE" : "NOT AVAILABLE",
        "DISCOUNT" : "NOT AVAILABLE",
        "LINK" : "https://www.newegg.com/Product/Product.aspx?Item=N82E16832698003"
}
```

Slika 3. Prikaz podataka u json formatu

5. Usporedba biblioteka lxml i Beautiful Soup

Lxml je biblioteka napisana u programskom jeziku C što ju čini znatno bržom u odnosu na biblioteku BeautifulSoup, ali isto tako može izazivati veće poteškoće prilikom instalacije zbog toga što nije napisana u Pythonu (za razliku od biblioteke Beautiful Soup). Kako bih usporedio brzinu izvođenja preveo sam program, napisan primjenom biblioteke Beautiful Soup, kojeg sam opisao na početku ovoga rada u program koji se koristi bibliotekom lxml. Sintaksa je slična u obje biblioteke te kôd ima skoro jednak broj linija. Najveća razlika između ova dva programa je parser koji se koristi i metode kojima se pronalaze parsirani podaci. Sljedeći isječak kôda je isti kao i na početku ovog rada samo je u njemu umjesto Beautiful Soup biblioteke korišten lxml.

```
from lxml import html

import requests

def print_to_csv(title, price, link):

    fOpen.write(title.replace(',', '|') + ',' + price + ',' + ',' + link +
'\n')

website_url =
'https://www.newegg.com/Product/ProductList.aspx?Submit=ENE&IsNodeId=1&N=10002
1392%20600489407%20600489408%20600538759%20600560190&name=Steam-PC-Download'
page_html = ''
try:

    r = requests.get(website_url)

    page_html = r.content

    html_element = html.document_fromstring(page_html)

    r.close()

except requests.exceptions.RequestException as e:

    print ('Error: ' + str(e))

    r = None

filename = 'videoGames.csv'
fOpen = open(filename, 'w')
headers = 'NAME, PRICE, LINK \n'
fOpen.write(headers)
```

```
names = html_element.cssselect('.item-title')

prices1 = html_element.cssselect('.price-current strong')

prices2 = html_element.cssselect('.price-current strong')


for name, price1, price2 in zip(names, prices1, prices2):

    name_text = name.text_content()

    price_text = price1.text_content() + '.' + price2.text_content()

    link_text =  name.attrib['href']

    print_to_csv(name_text, price_text, link_text)
```

Kako bih vidio koji je pristup brži i efikasniji, stavio sam oba kôda u while petlju koja će se izvesti pedeset puta.

```
i = 0
start_time = time.time()
while(i < 50):

    #Beautiful Soup i lxml kôd
    i+=1

print("--- %s seconds ---" % (time.time() - start_time))
```

Rezultati izvođenja programa na 50 HTML stranica su pokazali da je lxml brži za oko 5 sekundi što je relevantna razlika ukoliko uzmemo u obzir da je vrijeme izvođenja programa bilo samo dvadesetak sekundi. Na većim projektima poput onog kojeg sam napisao u okviru Scrapy koji se izvodio oko 3 sata razlika bi bila znatna.

```
Lxml: --- 20.819999933242798 seconds ---

Beautiful Soup: --- 25.001999855041504 seconds ---
```

Richard Lawson je u svojoj knjizi „Web Scraping with Python“ usporedio brzinu izvođenja biblioteka Beautiful Soup, lxml i regularnih izraza. Primjenio je svaku od metoda na jednostavnu, statičnu web stranicu 1000 puta. Modul regularnih izraza koji je poput lxml biblioteke napisan u programskom jeziku C parsirao je stranice u 5.5 sekundi, lxml u 7.06 sekundi,a Beautiful Soup u 42.84 sekunde što ju čini daleko najsporijom [3] . U ovom radu sam testirao i usporedio četiri alata za prikupljanje podataka s weba koje sam koristio u ovom

radu. Testiranje se sastojalo od prikupljanja podataka o video igrama s iste stranice www.newegg.com koje sam ponovio 50 puta za svaki alati. Rezultati usporedbe alata u brzini prikazani su u Tablici 2.

Tehnologija za web struganje	Vrijeme izvođenja
lxml	17.81s
Beautiful Soup	25.00s
Scrapy	7.38s
Selenium	26.39s

Tablica 2. Usporedba brzine izvođenja alata za web struganje

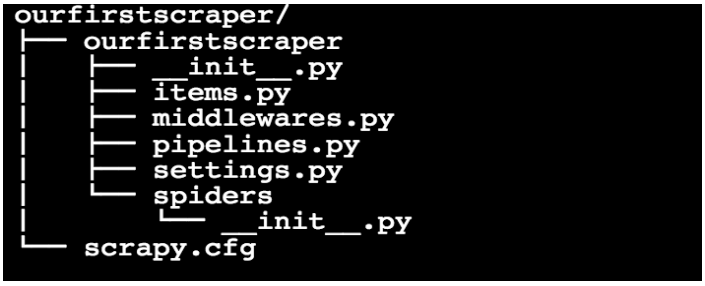
Što se brzine izvođenja tiče, nakon što sam sva četiri alata primjenio na isti skup podataka, došao sam do zaključka kako je daleko najbrži alat, specijalizirani okvir za web struganje Scrapy. Okvir Selenium je najsporiji, ali njegova prednost je da može skupljati podatke s najvećeg broja različitih web stranica. Osim toga, Selenium se može koristiti u kombinaciji s bilo kojim od ostalih okvira i biblioteka za prikupljanje dinamički generiranih podataka. Između ostalog često se koristi za izbjegavanje *captcha* upita, unosa korisničkog imena i lozinke i ispunjavanja formi. Kako Selenium pruža funkcionalnost koju ostale biblioteke i okviri nemaju, brzina izvođenja nije presudno svojstvo kako bi bio koristan. BeautifulSoup je sporiji od okvira Scrapy i biblioteke lxml, a nema velik broj ostalih funkcionalnosti kao Selenium pa je najveća motivacija za korištenje ove biblioteke jednostavnost pisanja kôda i pregršt dokumentacije i literature za učenje. Zbog toga je BeautifulSoup najbolja biblioteka za učenje web struganja. Biblioteka lxml je najbrža biblioteka za parsiranje HTML i XML kôda pa je, ako zanemarimo okvir Scrapy, najbolji izbor. Lxml je pisan u programskom jeziku C zbog čega brže parsira datoteke, ali je zbog toga otežana instalacija na pojedinim operacijskim sustavima i distrubucijama. Isto tako za lxml postoji manje dokumentacije u usporedbi s bibliotekom BeautifulSoup. Mana ove dvije biblioteke je što zahtjevaju korištenje drugih biblioteka što otežava i usporava proces prikupljanja podataka s weba.

6. Prikupljanje podataka web okvirom „Scrapy“

Za sada program ispravno pohranjuje sve tražene informacije u bazu podataka, ali problem je što prikuplja podatke samo s prve stranice trgovine. Program koji prikuplja podatke o proizvodima samo s prve stranice neke velike trgovine za kupnju putem weba nije praktičan jer korisnik neće dobiti informacije o većini raspoloživih proizvoda. Stoga je sljedeći zadatak unaprijediti postojeći program tako da prati poveznice na ostale stranice trgovine i prikupi podatke o svim proizvodima koji su u trgovini. BeautifulSoup je biblioteka koja predstavlja najbolji izbor kada se prikuplja manja količina podataka, ali kod velike količine podataka koji se nalaze na mnoštvu stranica, praktičnije je koristiti aplikacijski web okvir za web struganje: Scrapy. Uz pomoć okvira Scrapy preuzet ću jednu po jednu stranicu online trgovine www.newegg.com te sa svake pojedinačno preuzeti podatke na isti način na koje sam ih preuzeo prethodno s početne stranice što će rezultirati bazom podataka u kojoj će se nalaziti informacije o svim proizvodima online trgovine. BeautifulSoup i lxml su biblioteke za parsiranje html i xml datoteka, a Scrapy je aplikacijski okvir za pisanje „pauka“ koji prolaze kroz web stranice i s njih prikupljaju podatke. Scrapy koristi mehanizme selektora za prikupljanje podataka, ali je za zadatak prikupljanja podataka također dozvoljeno korištenje biblioteka BeautifulSoup i lxml. Selektori dolaze do traženih podataka tako što pretražuju HTML datoteku na temelju XPath i CSS izraza. XPath se koristi za navigaciju kroz elemente i atribute XML dokumenata, ali se može primjeniti i na HTML dokumentima, a CSS (engl. *Cascading Style Sheets*) je jezik koji opisuje stil HTML dokumenata. Koristan je za prikupljanje podataka s weba zbog korištenja klasa pomoću kojih je jednostavnije selektirati pojedine HTML tagove. U slučaju korištenja biblioteka BeautifulSoup i lxml umjesto selektora, Scrapy služi samo za navigaciju kroz web stranicu.

6.1. Kreiranje projekta

Kako bih kreirao projekt dovoljno je upisati komandu „scrapy startproject“ te nakon toga ime projekta. Ova komanda pokreće niz procesa okvira Scrapy te automatski kreiraju strukturu direktorija koji u sebi imaju unaprijed zadane datoteke.



Slika 4. Prikaz datotečnog sustava 1

Najvažnija Python datoteka je settings.py u koju se unose i spremaju sve postavke „pauka“ koji prolaze kroz web stranicu, a najvažniji direktorij je /*spiders* direktorij u kojem se pohranjuju i iz kojeg se pokreću svi pauci koje programer napiše [17].

6.2. Postavke

U *settings* datoteci se definiraju mnoge stvari koje su nužne za pravilno i moralno prikupljanje podataka s web stranica. U settings datoteci se definira varijabla *user agent* čijom se vrijednošću korisnik ili u ovom slučaju softver za prikupljanje podataka s weba predstavlja web stranici s koje prikuplja podatke. Najbolja praksa je user agent nazvati konfiguracijom web preglednika kojeg osoba koristi.

```
USER_AGENT = 'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/64.0.3282.186 Safari/537.36'
```

Osim toga, u ovoj se datoteci zapisuje hoće li softver poštovati pravila stranica u vidu robots.txt datoteke u kojoj su zapisana „pravila ponašanja“ softvera.

```
ROBOTSTXT_OBEY = True
```

Isto tako u settings.py datoteci se definira format u kojem će se prikupljeni podaci pohranjivati. Ukoliko programer želi podatke pohraniti u csv datoteku onda će u settings.py datoteku dodati sljedeće:

```
FEED_FORMAT = "csv"
FEED_URI = "videoGames.csv"
```

U okviru Scrapy postoji *AutoThrottle* ekstenzija te je dobra praksa da ju se uključi jer ona automatski podešava brzinu prikupljanja podataka na temelju trenutnog opterećenja Scrapy servera i servera stranice s koje se prikupljaju podaci. Uključenjem *AutoThrottle* ekstenzije

postupam „pristojnije“ prema prometu web stranice i ne riskiram mogućnost blokiranja svog programa.

```
AUTOTHROTTLE_ENABLED = True
AUTOTHROTTLE_TARGET_CONCURRENCY = 4.0
```

Iz istog razloga treba uključiti HTTP Cache kako prilikom razvijanja i testiranja softvera za prikupljanje podataka s weba ne opterećujem server.

```
HTTPCACHE_ENABLED = True
HTTPCACHE_EXPIRATION_SECS = 0
```

6.3. Prikupljanje podataka s jedne stranice

Prvi „pauk“ kojeg ću napraviti prikupljati će podatke o proizvodima samo s prve stranice, isto kao i prethodni program napravljen korištenjem biblioteka BeautifulSoup i Requests. Korištenjem okvira Scrapy ne postoji više potreba za korištenjem biblioteke Requests jer Scrapy u sebi ima ugrađenu funkciju koja preuzima HTML kôd, a nije potrebno koristiti ni biblioteku BeautifulSoup jer Scrapy također sam parsira HTML kôd. Stoga je dovoljno samo uključiti biblioteku scrapy. Scrapy se sastoji od mnoštva klasa te se mora definirati klasa za svakog napisanog „pauka“. Unutar klase moraju se definirati tri varijable: *ime* (koje označava ime „pauka“), dozvoljena domena (ime domene kroz koju „pauk“ smije prolaziti) i početna URL adresa (adresa s koje pauka počinje prikupljati podatke). Svakim pokretanjem programa, program vraća objekt *response*. U njemu su sadržani svi podaci o HTML dokumentu. Ova funkcionalnost okvira Scrapy slična je biblioteci Requests. Nakon što sam definirao varijable potrebne da bi se program pravilno izvodio, definirat ću funkciju *parse(self, response)* u kojoj ću pomoću CSS ili XPath selektora „dohvatiti“ podatke koji su mi potrebni. Funkcija će se pozvati svaki puta kada se stvori novi objekt response u kojem je pohranjena HTML datoteka.

```
import scrapy

class NeweggSpider(scrapy.Spider):
    name = 'neweggscraper'
    allowed_domains = ['www.newegg.com']
    start_urls =
['https://www.newegg.com/Product/ProductList.aspx?Submit=ENE&IsNodeId=1&N=1000
21392%20600489407%20600489408%20600538759%20600560190&name=Steam-PC-Download']
```

```
    def parse_product(self, response):
        name = response.css('.item-title::text').extract()
        price = response.css('.price-current strong::text').extract()
        price2 = response.css('.price-current sup::text').extract()
        link = response.css('.item-img::attr(href)').extract()
```

Sve podatke koji su mi potrebni dohvatio sam koristeći CSS selektore. Uzmem li za primjer varijablu *link* vidjet ću da točka na početku imena označava da se radi o CSS klasi s nazivom *item-img*, a oznaka *attr(href)* govori programu da želim prikupiti vrijednosti atributa href, odnosno poveznicu.

U funkciju *parse()* ću još ugnijezditi jednu for petlju koja će svaki puta kada se podaci pohrane u varijable, zapisati ih u Python dictionary obliku.

```
for item in zip(name, price, price2, link):
            scrapedInfo = {
                'NAME' : item[0],
                'PRICE' : item[1] + item[2],
                'LINK' : item[3],
            }

            yield scrapedInfo
```

Kako bi se definirao jedinstveni izlaz iz programa, Scrapy pruža mogućnosti unaprijed definirane klase *Item*. Objekti klase Item su jednostavni spremnici u koje se pohranjuju prikupljeni podaci. Podaci o postojećim objektima klase Item nalaze se u datoteci items.py koja se automatski generira prilikom započimanja novog Scrapy projekta. Ključna riječ yield šalje prikupljene podatke iz varijable *scrapedInfo* na daljnje procesiranje.

```
import scrapy


class ScrapeneweggItem(scrapy.Item):
    name = scrapy.Field()
    price = scrapy.Field()
    link = scrapy.Field()
```

Objekti *Field()* se koriste kako bi specificirali meta podatke (podatke koji sadrže informacije o podacima koje prikupljamo), za svaki zaseban tip podatka. Objekt *Field()* postoji za svaki tip podatka: naziv, vrijednost i poveznicu.

6.4. Prikupljanje podataka s cijele stranice

Nakon što sam napravio program koji prikuplja tražene podatke s jedne stranice, sljedeći korak je prikupiti podatke sa svake stranice online trgovine. Postoje dva načina na koja se može pristupiti ovom problemu. Prvi pristup je da se pronađe uzorak u samoj adresi web stranice te da se konstruira petlja koja iterira kroz sve adrese online trgovine i na svakoj od njih pojedinačno primjenjuje algoritam prikupljanja podataka. U adresi prve stranice online trgovine s koje prikupljam podatke postoji parametar „page“ (podcrtan i obojan u crveno).

```
https://www.newegg.com/Product/ProductList.aspx?Submit=ENE&N=100021392%20600489407%20600489408%20600538759%20600560190&IsNodeId=1&page=1&bop=And&PageSize=36&order=BESTMATCH
```

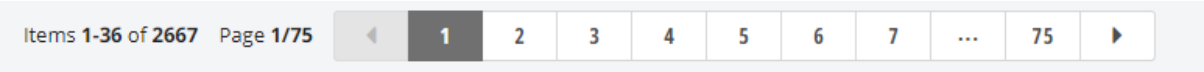
Prelaskom na nekoliko sljedećih stranica online trgovine, primjetio sam da je jedino što se mijenja brojka koja označava broj stranice trgovine pa sam konstruirao while petlju koja taj broj uvećava za jedan te na svakoj od tih stranica primjenjuje algoritam koji sam prethodno objasnio. Kako bi se implementirao ovaj pristup dovoljno je na početku programskog kôda napisati sljedeće:

```
start_urls =
['https://www.newegg.com/Product/ProductList.aspx?Submit=ENE&N=100021392%20600489407%20600489408%20600538759%20600560190&IsNodeId=1&page=1&bop=And&PageSize=36&order=BESTMATCH']
i = 2
while(i<75):
start_urls.append('https://www.newegg.com/Product/ProductList.aspx?Submit=ENE&N=100021392%20600489407%20600489408%20600538759%20600560190&IsNodeId=1&page='+str(i)+'&bop=And&PageSize=36&order=BESTMATCH')
```

Prije samog prikupljanja podataka, u start_urls listu u kojoj se zapisuju adrese s kojih će se prikupljati podaci se zapisuju sve adrese od 2 do 75 (koliko ima stranica trgovine) te se nakon toga podaci počinju prikupljati.

Drugi pristup nudi bolje, ali implementacijski kompleksnije rješenje. Naime, rješenje se sastoji od praćenja poveznica na druge stranice na isti način na koji bi čovjek pregledavao online trgovinu. U ovom slučaju primjena takvog rješenja nije bila potrebna jer postoje obrasci u adresi online trgovine i jer je broj stranica trgovine unaprijed poznat i određen, ali postoje situacije u kojima želimo prikupiti podatke o proizvodima koji se ne nalaze na predvidljivoj domeni. Ja sam kao projekt zadao prikupljanje podataka o video igrama za

osobna računala, ali kada bih htio prikupiti podatke o video igrama koje su dizajnirane za različite platforme ovakav pristup ne bi bio moguć iz razloga što bi adresa za svaku platformu bila različita iako je dio iste domene. Prvi pristup sam implementirao na stranici www.newegg.com te sam njime prikupio podatke sa svih 75 stranica online trgovine pa ću stoga drugi pristup implementirati na drugoj stranici koja ima mnogo više stavki (oko 40,000) u svojoj trgovini: http://store.steampowered.com



Slika 5. Prikaz navigacijskog sučelja

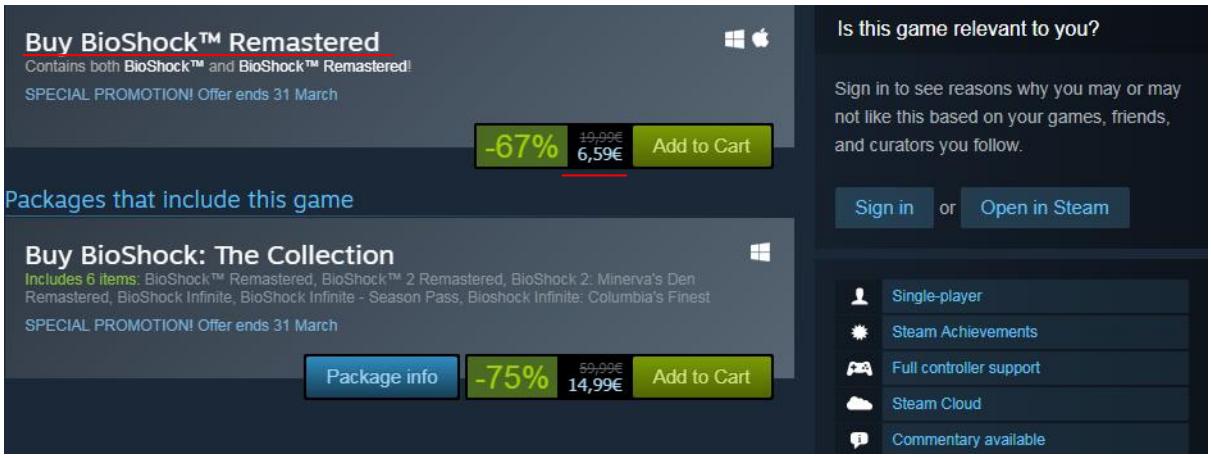
Web okvir „Scrapy“ u sebi ima nekoliko unaprijed uključenih „generičnih pauka“. Ti su „pauci“ klase koji sadrže metode koje se mogu koristiti pri izgradnji programa za prikupljanje podataka s weba. Korištenjem tih metoda programer ne mora sam pisati često korištene funkcije poput parsiranja XML/CSV feeda ili onoga za što će primarno meni biti potrebne, a to je praćenje poveznica. Za praćenje poveznica koristit ću najčešće korištenog „pauka“ za prikupljanje podataka s weba: *CrawlSpider*. Korištenjem ove klase otvara se mogućnost definiranja polja u koje se zapisuju pravila prema kojima program prolazi kroz poveznice neke stranice. Osim pravila, ova klasa uvodi i *Link Extractor* objekt koji definira na koji način će se ekstrahirati poveznice sa svake stranice kroz koju je program prošao [10].Link Extractor je dio lxml biblioteke i implementiran je pomoću robusnog lxml HTML parsera.

Iako se svi potrebni podaci nalaze na početnoj stranici gdje su izlistani svi artikli s nje je nemoguće podatke prikupljati korištenjem okvira Scrapy ili bibliotekom BeautifulSoup jer se njen sadržaj dinamički generira korištenjem skriptnog jezika Javascript.

Slika 6. Prikaz podataka za prikupljanje

Jedino rješenje problema je praćenje poveznica i ulazak u svaku zasebnu web stranicu koja je

posebna za svaki artikl te prikupljanje traženih podataka s te, novootvorene stranice. Za programe koji prikupljaju podatke s weba je ovo i bolje rješenje jer postoji veći broj podataka koji je dostupan na zasebnoj stranici svakog artikla.



Slika 7. Prikaz podataka za prikupljanje

Pravila se definiraju instanciranjem Rule() objekata, a u njima se nalaze Link Extractor objekti koji će u sebi imati određene smjernice prema kojima će prikupljati poveznice. Sljedećim naredbama ću uvesti klasu CrawlSpider, pravila koja se nalaze u toj klasi i Link Extractor objekte:

```
from scrapy.spiders import CrawlSpider, Rule
from scrapy.linkextractor import LinkExtractor
```

```
rules = [
        Rule(
            LinkExtractor(
                allow='/app/(.+)/',
                restrict_css='#search_result_container'
            ),
            callback='parse_product'
        ),
        Rule(
            LinkExtractor(
                allow='page=(\d+)',
                restrict_css='.search_pagination_right')
        )
    ]
```

Oba Link Extractor objekta koja sam definirao imaju dva parametra. Parametru *Allow* se dodjeljuje regularni izraz koji se mora podudarati s apsolutnom URL adresom s koje se

prikupljaju poveznice, a parametar restrict_css sadrži CSS selektor specifičnog područja s kojeg će se poveznice preuzimati [18]. Pomoću prvog pravila, program ulazi u stranicu svakog zasebnog, a pomoću drugog pravila, program pronalazi poveznicu na sljedeću stranicu trgovine i prelazi na nju nakon čega se nastavlja prikupljanje podataka omogućeno prvim pravilom [19].

Ova pravila u kombinaciji s funkcijom *parse_product()* dovoljna su za prikupljanje podataka sa stranice http://store.steampowered.com, ali mnogi se podaci neće prikupiti što će rezultirati nepotpunom listom. Na nekim stranicama zasebnih artikala postoje mehanizmi kontrole pristupa koji korisniku onemogućuju ulaz u stranicu bez da ispuni određenu formu. Kada program pokuša ući na stranicu koja ima ovaj mehanizam biti će preusmjeren na drugu stranicu gdje će mu biti ponuđena forma. Jedini način da uđe u željenu stranicu je da ispuni ponuđenu formu.

Prvi mehanizam kontrole pristupa zahtjeva od korisnika da unese svoj datum rođenja. Forma je jednostavna, korisniku se prikažu tri padajuća izbornika i tipka za unos.

Slika 8. Prikaz prvog mehanizma kontrole pristupa

Kako bih mogao doći do željenih podataka moram napisati funkciju koja će unijeti valjani datum i nakon što ga unese pritisnuti tipku za unos. Kada program pokuša pristupiti adresi http://store.steampowered.com/app/8870/BioShock_Infinite/ preusmjerava se na drugu adresu: http://store.steampowered.com/agecheck/app/8870/ i to vrijedi za sve igre koje imaju implementiran ovaj mehanizam kontrole pristupa. Kako bi program mogao automatizmom prolaziti kroz ovu kontrolnu točku implementirat ću u već postojeću funkciju *parse_product()* dio koda koji će se izvesti samo ako se u adresi stranice na koju je program preusmjeren

nalazi *„/agecheck/app“*. Ovaj se izraz nalazi samo na toj stranici pa zbog toga javlja programu da može početi s ispunjavanjem forme. Za samo popunjavanje forme koristit ću klasu FormRequest koja je specijalizirana verzija bazične klase Request koja u sebi ima ugrađene funkcije za ispunjavanje HTML formi. Koristi lxml.html forme kako bi unaprijed popunila polja podacima koje ću joj proslijediti. Prije nego što mogu ispuniti formu moram je dohvatiti. Sve potrebne elemente forme ću dohvatiti korištenjem css selektora na isti način na koji sam prethodno dohvatio cijenu, naziv i poveznicu artikla u trgovini. U varijablu *action* se pohranjuje action atribut forme koji kao vrijednost ima poveznicu koja određuje gdje se šalju podaci koje korisnik unese u formu, a u varijabli *name* i *value* se pohranjuju podaci o istoimenim atributima HTML oznake input.

```
if '/agecheck/app' in response.url:
            form = response.css('#agecheck_box form')
            action = form.css('::attr(action)').extract_first()
            name = form.css('input ::attr(name)').extract_first()
            value = form.css('input ::attr(value)').extract_first()
```

Nakon što sam dohvatio podatke mogu u formu proslijediti datum definiran u varijalbi formdata i nakon toga podnijeti zahtjev.

```
formdata = {
                name: value,
                'ageDay': '1',
                'ageMonth': '1',
                'ageYear': '1900'
            }

            yield FormRequest(
                url=action,
                method='POST',
                formdata=formdata,
                callback=self.parse_product
            )
```

U „else dio“ programskog kôda se upisuje prethodno napisana funkcija za prikupljanje podataka [19].

Drugi mehanizam kontrole pristupa upozorava korisnika da sadržaj koji se nalazi na stranici kojoj pokušava pristupiti nije primjeren za svaku dob te ga pita želi li svejedno pristupiti

stranici ili se vratiti natrag na popis svih igara. Kako bi omogućio programu da uđe u stranicu i nastavi prikupljati podatke simulirat ću pritisak na tipku „View Page“, a to ću učiniti korištenjem okvira za automatizaciju web preglednika: Selenium.

Slika 9. Prikaz drugog mehanizma kontrole pristupa

Selenium je robustan alat za prikupljanje podataka s weba iako je originalno osmišljen kao alat za automatizaciju web preglednika u svrhu testiranja web aplikacija. Danas je velika količina sadržaja web stranica dinamički generirana korištenjem skriptnog jezika Javascript što predstavlja najveću prepreku kod prikupljanja podataka s weba. Selenium rješava taj problem na način da samostalno otvara web preglednik, učitava stranicu i prikuplja s nje podatke. Selenium nije web preglednik pa ga je potrebno integrirati u neki od podržanih web preglednika poput Mozilla Firefox i Google Chrome. Ako osoba pokrene Selenium uz Google Chrome, otvorit će se web preglednik, na njemu će se učitati tražena stranica, a zatim će se automatski u pregledniku dogoditi sve što je programer napisao u programskom kôdu [2].

Prije pisanja naredbi potrebno je preuzeti Google Chrome webdriver, uvesti webdriver modul iz biblioteke Selenium, spremiti apsolutnu putanju prethodno preuzetog Chrome webdrivera u varijablu i povezati ju sa okvirom Selenium.

```
from selenium import webdriver
def __init__(self):
   chromePath = r'C:\..\..\..\ChromeDriver\chromedriver.exe'
   self.driver = webdriver.Chrome(chromePath)
```

Adresa ove kontrole točke pristupa razlikuje se od adrese prve točke kontrole pristupe pa je moguće riješenje izvođenje sljedećeg programskog kôda ukoliko u adresi stranice točke kontrole pristupa bude sljedeći uzorak:

```
elif not re.findall('app/(.*)/agecheck', redirected.url):
```

U samom kôdu je prvo potrebno proslijediti url adresu selenium webdriveru, a potom selektorom dohvatiti tipku „*View Page*“ (slika 9).

```
self.driver.get(response.url)
tipka =
self.driver.find_element_by_xpath("""//*[@id="app_agegate"]/div[3]/a[1]""")
```

U okviru Selenium postoji velik broj funkcija kojima se nastoje simulirati radnje čovjeka, a u ovoj konkretnoj situaciji mi je potrebno izvođenje pristikanja lijeve tipke miša na dohvaćenju tipku. Kako bih to postigao dovoljno je pozvati funkciju na varijablu koja sadrži lokaciju same tipke i nakon toga ću novootvorenu stranicu proslijediti natrag parseru koji će nastaviti postupak prikupljanja podataka.

```
tipka.click()
yield Request(url=self.driver.current_url, callback=self.parse_product)
```

Nakon što sam implementirao mehanizme za prolazak točki kontrole pristupa, ako pokrenem program, on će prikupiti podatke o svim proizvodima online trgovine http://store.steampowered.com.

Okvir napravljen isključivo za web struganje Scrapy je robustan okvir koji velikom brzinom prikuplja podatke s weba i uz pomoć tzv. „cjevovoda“ pohranjuje podatke u CSV datoteku ili bazu podataka. Idealan je izbor za prikupljanje podataka s weba zbog velikih mogućnosti uređivanja izvornog kôda i brzine izvođenja, ali zbog raznih mogućnosti koje nudi najteži je za korištenje u usporedbi s ostalim alatima.

7. Prikupljanje podataka okvirom „Selenium“

Kao što sam prethodno napisao, Selenium se može koristiti za mnoge stvari poput zaobilaženja Captcha autentikacije koje se pojavljuju ako web stranica primjeti da ju pretražuje „robot“, a ne čovjek, simuliranje unosa korisničkog imena i lozinke ukoliko web stranica traži od korisnika da se prijavi kako bi nastavio pretraživati stranicu, ispunjavanje različitih formi, ali najvažnija funkcionalnost Seleniuma je mogućnost izvođenja Javascript kôda. Često su podaci koje bismo željeli prikupiti „skriveni“ unutar oznake *script* u HTML kôdu i ne mogu se isčitati preuzimanjem izvornog kôda web stranice. U ovoj situaciji Selenium uvelike pomaže jer on izvodi Javascript kôd i samim time omogućuje iščitavanje podataka. Kada prikupljamo podatke s weba koristeći Beautiful Soup, lxml ili Scrapy prvo preuzimamo objekt, najčešće izvorni kôd te iz njega preuzimamo podatke. Ukoliko se podaci nalaze u oznaci *script* biti će vidljivi u web pregledniku, ali ne i u izvornom kôdu.

```
<script>
  window.Humble = window.Humble || {}; // Create Humble info struct if not
present.
  window.Humble.title = 'The Humble Store: Great games. Fantastic prices.
Support charity.'; // Expose title to frontend application.
</script>
```

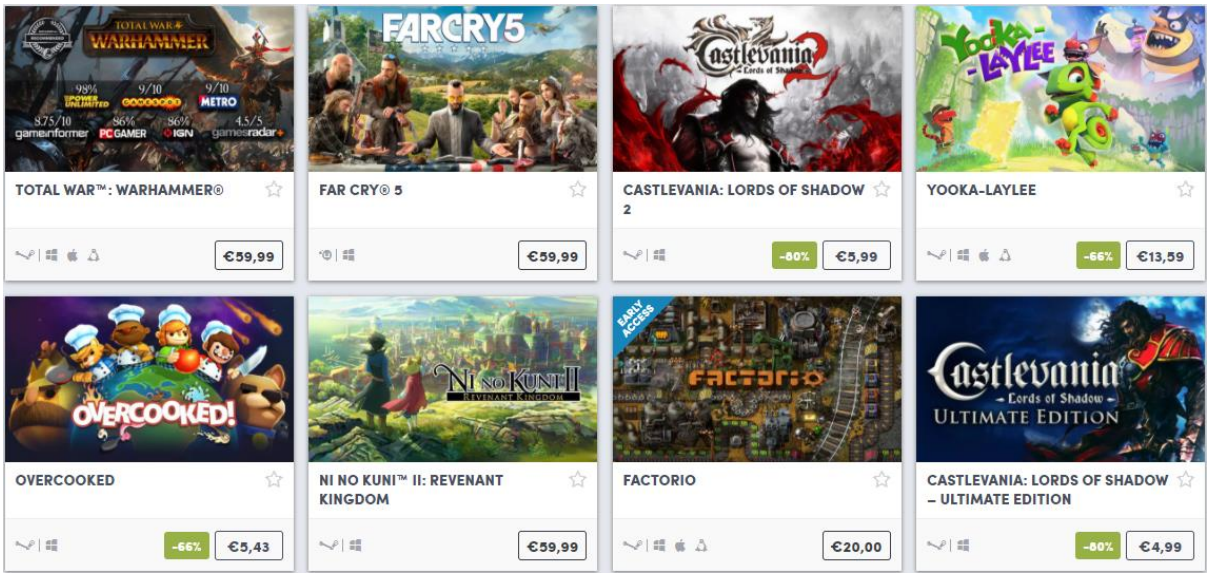
Svi podaci o artiklima zapisani su unutar oznake *script* poput ove. Selenium otvara preglednik i direktno iz njega, nakon što se Javascript učita, izvlači podatke. Sljedeće što ću prikazati je preuzimanje podataka sa stranice iz koje je ovaj isječak kôda preuzet kako bih demonstrirao prikupljanje podataka koji se dinamički generiraju i koristit ću samo Selenium.

Prije svega potrebno je definirati jedan od webdrivera. Ja ću za ovaj primjer koristit Mozilla Firefox webdriver koji se u sintaksi minimalno razlikuje od Google Chrome webdrivera kojeg sam koristio u prethodnom primjeru. Osim toga, kako se Selenium ponaša poput čovjeka koji pretražuje web stranicu na njega utječu certifikati koji na stranici postoje radi sigurnosti. Ukoliko ne zaobiđem certifikate na stranici, neću moći prikupljati podatke jer mi stranica neće dozvoliti ulaz. Konkretno, na stranici s koje prikupljam podatke: https://www.humblebundle.com/ je implementiran SSL certifikat. SSL certifikati se koriste na stranicama gdje korisnici ostavljaju svoje osobne podatke poput broja kreditne kartice. SSL certifikat osigurava da broj kartice bude sigurno prenesen do poslužitelja.

```
profile = webdriver.FirefoxProfile()
profile.accept_untrusted_certs = True
```

```
driver = webdriver.Firefox(firefox_profile=profile)
```

Podaci o video igrama su dinamički generirani isto kao i na stranici http://store.steampowered.com (Slika 10.)



Slika 10. Prikaz podataka za prikupljanje

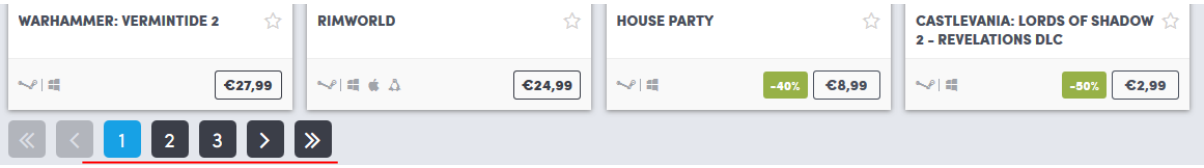
Kada sam prikupljao podatke sa http://store.steampowered.com problemu sam pristupio na način da sam ulazio u svaku zasebnu stranicu svake pojedine video igre, s nje prikupljao podatke, vraćao se natrag na početnu stranicu i ponavljao taj proces. Zbog toga je proces prikupljanja podataka o svim video igrama trajao nekoliko sati. Selenium je sporiji zbog toga što za razliku od okvira Scrapy i biblioteka Beautiful Soup i lxml izvodi Javascript i djeluje slično kao i čovjek. Ipak, Selenium će zadatak odraditi nešto brže zbog toga što može prikupljati dinamički generirane podatke i ne mora ulaziti u svaku zasebnu stranicu svake pojedine video igre.

```
def ScrapeHumbleBundle(url):
    driver.get(url)
    timeout = 20
    try:
        WebDriverWait(driver,
timeout).until(EC.visibility_of_element_located((By.XPATH,
"/html/body/div[1]/div/div[5]/div[2]/section/div[2]/div[2]/div/div/div[2]/div/
div[2]/div[1]/ul/li[1]/div/div/a/div/img")))
        print ("Page is ready!")
    except TimeoutException:
        print('Timed out waiting for page to load')
        driver.quit()
```

Prvo što je potrebno napraviti je otvoriti stranicu i čekati da se ona otvori odnosno pričekati da se elementi koje želim prikupiti učitaju na stranici. U ovom konkretnom slučaju ja sam pričekao da se učitaju slike pošto su one najzahtjevniji elementi stranice. Drugim riječima, ako se učitaju slike, učitala se i čitava stranica. U slučaju da se stranica ne uspije učitati u 20 sekundi, program će se ugasiti.

```
    names = driver.find_elements_by_class_name('entity-title')
    prices = driver.find_elements_by_class_name('price')
    links = driver.find_elements_by_css_selector(div:nth-child(1) > a:nth-
            child(1)')
    for name, price, link in zip(names, prices, links):
        #pohrana ili ispis podataka
        print('NAME: ' + name.text + ' || PRICE: ' + price.text + ' || LINK: +
link.get_attribute('href'))
```

Varijable *names*, *prices* i *links* predstavljaju polja u koja se pohranjuju podaci, a potom ih kôdom unutar for petlje pohranjujemo u bazu podataka, CSV datoteku ili u ovom slučaju ispisujemo u konzolu. Ovim kôdom se prikupe i pohrane svi podaci s jedne stranice. Kako bih prešao na iduću stranicu moram, kao da sam pretražujem stranicu, pronaći tipku koja vodi na stranicu na tu sljedeću stranicu.



Slika 11. Prikaz navigacijskog sučelja

Tipka se nalazi na samom dnu stranice nakon izlistanih artikala. Pohranit ću njen Xpath u varijablu i na nju pozvati funkciju *click()* nakon čega će web preglednik automatski učitati sljedeću stranicu. Nakon što se stranica otvori, pozvat ću ponovno istu funkciju s adresom novootvorene stranice čime se postiže rekurzija. Kada program dođe do posljednje stranice, sljedeća stranica se nikada neće učitati i funkcijom *quit()* se program zatvara.

```
goNext =
driver.find_element_by_xpath('/html/body/div[1]/div/div[5]/div[2]/section/div[
2]/div[2]/div/div/div[2]/div/div[3]/div/a[314]')
    goNext.click()
    time.sleep(3)
    print('CURRENT URL: ' + driver.current_url)
    return ScrapeHumbleBundle(driver.current_url)
```

Okvir Selenium najbolje je koristiti u situacijama u kojima se podaci na stranici dinamički generiraju ili su napisani korištenjem okvira Ajax gdje se podaci ažuriraju nakon što se stranica učita i slično. Koristi se za razne stvari, uglavnom za automatizaciju web preglednika, a kada se radi o web scrapingu, najveća prednost je prikupljanje dinamički generiranih podataka i jednostavnost korištenja dok je što se brzine tiče sporiji od okvira Scrapy i biblioteka lxml i BeautifulSoup.

8. Zaključna razmatranja

U ovome radu dao sam uvid u najkorištenije tehnologije za prikupljanje podataka s weba u programskom jeziku Python. Iznio sam prednosti i nedostatke svake tehnologije, međusobno ih usporedio te prikazao njihovu primjenu na nekoliko tematski povezanih projekata na području internetske trgovine. Prikupljanje podataka s weba ili web struganje je korisno u širokom spektru djelatnosti od elektronske trgovine do znanstvenih istraživanja jer omogućava subjektu da dobije saznanja o tržištu ili o polju istraživanja s lokacije koja sadrži najveću količinu nestrukturiranih podataka – Internetu. Napsiao sam nekoliko programa koji su prikupljali i pohranjivali podatke o digitalnim proizvodima koji se prodaju i kupuju isključivo putem weba. Ideja svakog programa kojeg sam napisao je da olakša potrošaču na način da mu pomogne donijeti najbolju odluku. Na tržištu već postoje mnoge web stranice koji implementiraju tehnologije poput Beautiful Soup i lxml biblioteka i okvira Scrapy i Selenium kako bi uspoređivale cijene različitih artikala i obrazovale potrošače. Beautiful Soup primarno se koristi u manjim projektima gdje se ne prikuplja velika količina podataka. Ovu biblioteku krasi jednostavnost sintakse te mogućnost prikupljanja podataka čak i sa stranica koje sadrže loše napisan HTML kôd. S druge strane njoj najsličnija biblioteka lxml također ima jednostavnu sintaksu i znantno brže mogućnosti parsiranja, ali na mnogim platformama može doći do problema s instalacijom. Okvir Scrapy predstavlja najbolje riješenje za prikupljanje velike količine podataka i zbog toga je najpopularniji izbor za komercijalne proizvode. Sintaksa je složenija te je potrebno naučiti sve mogućnosti okvira prije nego što se mogu potpuno iskoristiti sve njegove mogućnosti. Iako je to razvijen okvir za web struganje, otvorenog je kôda pa postoji mnogo prostora za osobne izmjene. Selenium, kao drugi okvir kojeg sam koristio, kao prednost ima izvođenje Javascript kôda naušrtb brzine. Smatram kako je to jedan od najboljih načina za web struganje gdje brzina nije apsolutno najvažnije svojstvo, a korištenjem web preglednika poput PhantomJS otvara se mogućnost izvođenja u pozadini. Isto tako, kako dinamički generirane stranice postaju standard u razvoju web stranica, korištenje okvira Selenium će postati nužnost. Web struganje koristi se i u različitim znanostima u kojima znanstvenici ovise o velikoj količini podataka kako bi dokazali određene teorije. Primjerice, Richard N. Landers u svojem članku "Internet Scraping for Research" navodi kako je najveći problem istraživanja na polju psihologije pronalazak osoba koje bi sudjelovale u istraživanju te naglašava kako je često rješenje upravo u web struganju. Danas kada živimo u doba informacija, u doba kada informacije često vrijede jednako ili više nego materijalna dobra, poznavanje tehnologija

prikupljanja podataka s weba i pretvaranja istih u korisne informacije znači veliku prednost u svakoj znanstvenoj ili gospodarskoj djelatnosti.

9. Literatura

[1] WWW Foundation, »History of the Web,« 2016. [Mrežno]. Available: http://webfoundation.org/about/vision/history-of-the-web/. [Pokušaj pristupa 13 March 2018].

[2] R. Mitchell, Web Scraping with Python, USA: O'Reilly Media, Inc., 2015.

[3] R. Lawson, Web Scraping with Python, Birmingham: PACKT publishing, 2016.

[4] C. K, MongoDB: The Definitive Guide, Sebastopol: O'Reilly Media, 2013.

[5] M. Rouse, »What Is,« 2005. [Mrežno]. Available: http://whatis.techtarget.com/definition/Python. [Pokušaj pristupa 5 April 2018].

[6] M. Lutz, Learning Python, Sebastopol: O' Reilly Media, Inc, 2013.

[7] DataHut, »Beginner's guide to Web Scraping with Python lxml,« 2016. [Mrežno]. Available: http://blog.datahut.co/beginners-guide-to-web-scraping-with-python-lxml/. [Pokušaj pristupa 8 April 2018].

[8] Python Requests, »Docs. Quickstart,« 2018. [Mrežno]. Available: http://docs.python-requests.org/en/master/user/quickstart/#make-a-request. [Pokušaj pristupa 14 April 2018].

[9] S. Avasarala, Selenium WebDriver Practical Guide, PACKT publishing, 2014.

[10] Doc.scrapy.org, »Spiders - Scrapy 1.5.0 documentation,« 2016. [Mrežno]. Available: https://doc.scrapy.org/en/latest/topics/spiders.html#crawling-rules. [Pokušaj pristupa 26 March 2018].

[11] MongoDB, »What is MongoDB?,« 2018. [Mrežno]. Available: https://www.mongodb.com/what-is-mongodb. [Pokušaj pristupa 9 April 2018].

[12] Business Dictionary, »What is parsing? Definition and meaning.,« 2018. [Mrežno]. Available: http://www.businessdictionary.com/definition/parsing.html . [Pokušaj pristupa 13 April 2018].

[13] Wikipedia, »Comma-Separated Values,« 2018. [Mrežno]. Available: https://en.wikipedia.org/wiki/Comma-separated_values. [Pokušaj pristupa 15 April 2018].

[14] R. Elmasri i S. Navathe, Fundamentals of database systems, Addison-Wesley, 2010.

[15] MongoDB, »NoSql explained,« 2015. [Mrežno]. Available: https://www.mongodb.com/nosql-explained?jmp=footer . [Pokušaj pristupa 28 March 2018].

[16] JSON, »Json,« 2018. [Mrežno]. Available: https://www.json.org/. [Pokušaj pristupa 28 March 2018].

[17] Analytics Vidhya, »Web Scraping in Python using Scrapy,« 2018. [Mrežno]. Available: https://www.analyticsvidhya.com/blog/2017/07/web-scraping-in-python-using-scrapy/. [Pokušaj pristupa 15 March 2018].

[18] Doc.Scrapy.org, »Link Extractors - Scrapy 1.5.0 documentation,« 2018. [Mrežno]. Available: https://doc.scrapy.org/en/latest/topics/link-extractors.html . [Pokušaj pristupa 27 March 2018].

[19] A. Perunicic, »Scraping the Steam Game Store with Scrapy,« The Scrapinghub Blog, 2017. [Mrežno]. Available: https://blog.scrapinghub.com/2017/07/07/scraping-the-steam-game-store-with-scrapy/. [Pokušaj pristupa 28 March 2018].