Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Кафедра Штучного інтелекту

Дисципліна: «Інтелектуальні технології в Internet та Semantic Web»

Лабораторна робота № 1

на тему: «РОЗРОБКА XML-ДОКУМЕНТІВ ТА ВІДПОВІДНИХ XML-СХЕМ У РЕДАКТОРІ XML»

|  |  |  |
| --- | --- | --- |
| Виконали:  ст. гр. ІТКН-17-7  Савенков Д.В. |  | Перевірив:  Бібічков І. Є. |
|  |  |  |

Харків 2020

1 РОЗРОБКА XML-ДОКУМЕНТІВ ТА ВІДПОВІДНИХ XML-СХЕМ У РЕДАКТОРІ XML

* 1. **Мета роботи**

Отримання даних із віддаленого WEB-ресурсу та їх обробка. Ознайомлення із роботою у редакторі XMLSpy або в аналогу – середовищі для роботи з кількома XML-технологіми. Розробка XML-схеми, XML-документа, його валідація.

**1.2 Хід роботи**

**Варіант -** 'https://djinni.co/set\_lang?code=en&next=/developers/' (резюме ІТ-спеціалістів)

**Посилання на гіт-репозиторій:** https://github.com/denys-savenkov/Savenkov-Semantic-Web

**1.2.1 Аргументація вибору мови програмування та засобів розробки**

Для виконання поставленого практичного завдання була обрана мова програмування Python за її легкість, зручність, наявність великої кількості бібліотек та добре складеної документації. Серед засобів розробки була обрана IDE PyCharm за її функціонал, зокрема, інтеграції механізмів та автоматизації роботи із XML та XSD файлами.

Для добування даних з веб ресурсу була обрана бібліотека Scrapy v2.0 (детальна інформація та документація: https://scrapy.org/). Однією з переваг саме цієї бібліотеки є розпаралелювання процесу «скрапінгу» та можливість зберігати дані у форматі XML.

**1.2.2 Створення XSD-схеми**

Після аналізу веб-ресурсу була створена наступна XSD-схема для подальшої валідації отриманих парсером даних:

<?xml version="1.0" encoding="UTF-8"?>

<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="candidates" type="candidatesType"/>

<xs:complexType name="locationsType">

<xs:sequence>

<xs:element type="xs:string" name="value" maxOccurs="unbounded" minOccurs="0"/>

</xs:sequence>

</xs:complexType>

<xs:complexType name="candidateType">

<xs:sequence>

<xs:element type="xs:string" name="url"/>

<xs:element type="xs:string" name="position"/>

<xs:element type="locationsType" name="locations"/>

<xs:element type="xs:string" name="salary"/>

<xs:element type="xs:float" name="years\_of\_experience"/>

<xs:element name="english\_level">

<xs:simpleType>

<xs:restriction base="xs:string">

<xs:enumeration value="Beginner/Elementary"/>

<xs:enumeration value="Advanced/Fluent"/>

<xs:enumeration value="Intermediate"/>

<xs:enumeration value="Pre-Intermediate"/>

<xs:enumeration value="Upper Intermediate"/>

<xs:enumeration value=""/>

</xs:restriction>

</xs:simpleType>

</xs:element>

<xs:element type="xs:string" name="experience"/>

<xs:element type="xs:string" name="skills"/>

<xs:element type="xs:string" name="highlights"/>

<xs:element type="xs:string" name="expectations"/>

</xs:sequence>

</xs:complexType>

<xs:complexType name="candidatesType">

<xs:sequence>

<xs:element type="candidateType" name="candidate" maxOccurs="unbounded" minOccurs="0"/>

</xs:sequence>

</xs:complexType>

</xs:schema>

**1.2.3 Створення веб-павуків**

Бібліотека Scrapy дає зручний інструментарій для роботи з веб-павуками, наприклад, обирати формат збереження даних або обробляти декілька сайтів водночас.

Хоча вбудований механізм роботи із XML непоганий, але він має певні обмеження, наприклад, неможливість кореневим тегам (які створюються автоматично) створити атрибути та записати у них значення, наприклад ID або, у нашому випадку, URL резюме.

Для реалізації павуків спочатку необхідно модифікувати клас XmlItemExporter (клас Scrapy, що експортує дані у XML), а саме змінити назви кореневих тегів:

from scrapy.exporters import XmlItemExporter

class DjinniXmlItemExporter(XmlItemExporter):

def \_\_init\_\_(self, file, \*\*kwargs):

super(DjinniXmlItemExporter, self).\_\_init\_\_(file,

root\_element="candidates",

item\_element="candidate",

\*\*kwargs)

Після цього створимо клас DjinniItem, що буде зберігати необхідні дані зі сторінок:

import scrapy

class DjinniItem(scrapy.Item):

url = scrapy.Field()

position = scrapy.Field()

locations = scrapy.Field()

salary = scrapy.Field()

years\_of\_experience = scrapy.Field()

english\_level = scrapy.Field()

experience = scrapy.Field()

skills = scrapy.Field()

highlights = scrapy.Field()

expectations = scrapy.Field()

Зробивши ці кроки, можемо перейти до створення веб-павука, який буде включати в себе два парсера: для сторінок з переліком посилань на резюме та для сторінок самих резюме. Для вибору необхідних елементів сторінки були використані CSS Selectors (альтернативою може бути XPath) та регулярні вираження.

Програмний код виглядає наступним чином:

import re

import scrapy

from scrapper\_items import DjinniItem

class DjinniSpider(scrapy.Spider):

name = 'djinni'

start\_urls = ['https://djinni.co/set\_lang?code=en&next=/developers/']

max\_candidates = 10000

def start\_requests(self):

self.current\_candidates = 0

yield scrapy.Request(self.start\_urls[0], self.parse\_main\_page)

def parse\_main\_page(self, response):

ten\_candidates = ['https://djinni.co/' + x for x in

response.css('div.searchresults a.profile::attr(href)').extract()]

for candidate\_url in ten\_candidates:

if self.current\_candidates > self.max\_candidates:

return

self.current\_candidates += 1

yield scrapy.Request(candidate\_url, self.parse\_candidate)

# next page

next\_page = response.css('li.next a::attr(href)').extract\_first()

if next\_page:

yield response.follow(url=next\_page, callback=self.parse\_main\_page)

def parse\_candidate(self, response):

candidate = DjinniItem(url=response.url)

candidate['position'] = response.css('h1::text').extract\_first().strip()

candidate['locations'] = [location.strip() for location in

response.css('div.main-profile-details::text').extract\_first().split(',')[:-1]]

candidate['salary'] = response.css(

'span.profile-details-salary::text').extract\_first().strip()

rx = re.compile(

r'[-+]? (?: (?: \d\* \. \d+ ) | (?: \d+ \.? ) )'

r'(?: [Ee] [+-]? \d+ ) ?',

re.VERBOSE)

years\_of\_experience = rx.findall(

response.css('p.before-hint::text').extract\_first().strip())

years\_of\_experience = 0 if len(years\_of\_experience) == 0 else float(years\_of\_experience[0])

candidate['years\_of\_experience'] = years\_of\_experience

candidate['english\_level'] = response.css('p.before-hint::text').extract\_first().split('English')[1].strip()

html\_tags = re.compile(r'<.\*?>')

main\_body = {x.strip(): html\_tags.sub('', y.strip()) for x, y in

zip(response.css('h4 \*::text').extract(),

response.css('p.profile').extract())}

candidate['experience'] = main\_body['Experience'] if 'Experience' in main\_body.keys() else ''

candidate['skills'] = main\_body['Skills'] if 'Skills' in main\_body.keys() else ''

candidate['highlights'] = main\_body['Highlights'] if 'Highlights' in main\_body.keys() else ''

candidate['expectations'] = main\_body['Looking for'] if 'Looking for' in main\_body.keys() else ''

yield candidate

if \_\_name\_\_ == '\_\_main\_\_':

from scrapy.crawler import CrawlerProcess

with open('djinni.xml', 'w') as f:

pass

process = CrawlerProcess(settings={

'FEED\_EXPORTERS': {

'xml': 'exporter.DjinniXmlItemExporter',

},

'FEED\_FORMAT': 'xml',

'FEED\_URI': 'djinni.xml'

})

process.crawl(DjinniSpider)

process.start() # the script will block here until the crawling is finished

**1.2.4 Валідація отриманих даних**

У результаті роботи реалізованого павука було отримано майже 10.000 записів резюме ІТ-спеціалістів наступної структури

<?xml version="1.0" encoding="utf-8"?>

<candidates>

<candidate><url>https://djinni.co//q/e1cbb5ef52/</url><position>Розробник PHP</position><locations><value>Kyiv</value></locations><salary>$400</salary><years\_of\_experience>0</years\_of\_experience><english\_level>Pre-Intermediate</english\_level><experience>Development of Symfony-based web service to automation online bidding process for premium aero services (administration system and client side).Technology: PHP 7.1 + Symfony 3.2 + Doctrine ORM / PDO (PostgreSQL) + Git + Git-CI + phpunit + Composer</experience><skills>PHP, Git, Linux, MVC, HTML, CSS, jQuery, MySQL, Symfony, webpack, Bootstrap</skills><highlights>create CRUD and Blog, fixed errors, updated modules, configured tests</highlights><expectations>skills development in PHP</expectations></candidate>

…

</candidates>

Щоб швидко та ефективно перевірити валідність даних, можемо скористатися розробленою раніше схемою XSD. Але спочатку, необхідно створити для цього програмний код:

from lxml import etree

xmlschema\_doc = etree.parse('djinni.xsd')

xmlschema = etree.XMLSchema(xmlschema\_doc)

# parse xml

try:

doc = etree.parse('djinni.xml')

print('XML well formed, syntax ok.')

# check for file IO error

except IOError:

print('Invalid File')

# check for XML syntax errors

except etree.XMLSyntaxError as err:

print('XML Syntax Error, see error\_syntax.log')

with open('error\_syntax.log', 'w') as error\_log\_file:

error\_log\_file.write(str(err.error\_log))

quit()

except:

print('Unknown error, exiting.')

quit()

По виконанню цього коду отримуємо отримали результат «XML well formed, syntax ok.», що свідчить про валідність отриманого XML документу.

* 1. **Висновки**

Під час виконання цієї практичної роботи було опрацьовано теоретичні знання та були набуті практичні навички щодо отримання даних з зовнішніх веб ресурсів, щодо роботи із XML документами та їх валідацією. Також була розглянута бібліотека для парсингу Scrapy, були виявленні її переваги та недоліки

У результаті був отриманий додаток, що добуває валідні дані резюме ІТ-спеціалістів з сайту djinni.co, що можна використовувати у інших проектах та практичних роботах