# Basic Web Scraping Application Documentation

*Release 1.0.0*

**Guide: Viplav Vimal, DE, BEL**
**Student: K Venkat Ramnan, PES1201801319, PESU**

**Jun 08, 2020**

# CONTENTS:

# CHAPTER ONE

## CERTIFICATE

This is to certify that **K.Venkat Ramnan, PES1201801319,** 4th Semester, Electronics and Communication Department, PESU has completed the 30days project “Basic Web scraper Application” from 13 May 2020 to 11 June 2020 at Network & Cyber Security / MilCom Strategic Business Unit (MilCom SBU), Bharat Electronics Limited (BEL), Bangalore.

**Shri. Viplav Vimal, DE, BEL**

**Shri. Mrityunjaya. P. Hegde, Manager, BEL**

**Smt. Umadevi B,DGM, BEL**

CHAPTER

# TWO

# ACKNOWLEDGEMENT

Thanks to Smt. Umadevi B, DGM for providing opportunity to do the project. Thanks to Shri. Mrityunjaya P Hegde, Manager and Shri.Viplav Vimal, Deputy Engineer for the guidance. Thanks to BEL for permitting to do 30 days Project work.

Thanks to Professor Smt. Anuradha.M, Head, Electronics and Communication Department, PESU.

CHAPTER

# THREE

# INTRODUCTION

## 3.1 Web scraping

Web scraping, web harvesting, or web data extraction is data scraping used for extracting data from websites. Web scraping software may access the World Wide Web directly using the Hypertext Transfer Protocol, or through a web browser. Web scraping extracts the data specific to the needs of the programmer. The data may be used for price comparisons, stock market comparsions etc.

## 3.2 Web scraping Vs Web crawling

The Web scraping is different from Web crawling. In Web crawling the web data is copied and used for indexing to improve the fastness of the search engine. The web crawling is the processing intensive operation and it crawls over the world Wide Web in an extensive way. The web scraping extracts information for specific purpose from the specific urls.

## 3.3 Project Work

For this student project of 30 days duration the chosen topic was “Basic Web Scraping Application”. The objective is to write a basic software application which scrapes the given keyword from the list of given web sites.

CHAPTER

FOUR

# DESIGN

## 4.1 High Level design

The urls to be scraped are entered into the text file. This test file is read and url reading and parsing is done. The keyword which is the word to be scraped is obtained from the web interface. Using the libraries the scraping is done.The extracted information from the Table Contents of the scraped websites is saved as a cvs file. The href links of the scraped web sites are saved as a text file.

## 4.2 Programming Language

For the basic web scraping the Python language is selected. We can also use C, C++, Node.js, Ruby and Java. Python is also having a library BeautifulSoup and a framework Scrapy. So for this project Python Programming Language is used.

## 4.3 Python Web scraping libraries / Framework

There are two libraries useful for Web scraping are Beautifulsoup and Scrapy.

- **BeatifulSoup**
  - Beautiful Soup is a Python library for pulling data out of HTML and XML files.
  - It creates a parse tree for parsed pages that can be used to extract data from HTML.
  - Extracting data from HTML is useful for web scraping.
  - Easy to learn, plenty of examples and books.
  - Single threaded, Synchronous, takes time to do task.
  - needs other libraries to perform scraping and it is not a framework.
  - better suited for simple projects.
- **Scrapy**
  - Scrapy is a free and open-source web-crawling framework written in Python.
  - Originally designed for web scraping.
  - It can also be used to extract data using APIs or as a general-purpose web crawler.
  - It is a framework.
  - Asynchronous and fast.

  - Difficult to understand and a deep learning curve.
  - Best suited for complex project and web crawling projects.

In this project Beautifulsoup is used because of its simplicity. The Html parser is used for extracting the data.

## 4.4 Url Handling in Python

The Python packages for opening, reading and parsing the urls is required for the project. The package urllib have the modules for opening, reading and parsing the urls and is used in this project.

## 4.5 RESTful API

The web scraping function is using BeautifulSoup. This functionality is converted into RESTful API using Flask,Flask-Restful and Flask-CORS. The Representational state transfer is a software architectural style that defines a set of constraints to be used for creating Web services. Web services that conform to the REST architectural style, called RESTful Web services, provide interoperability between computer systems on the Internet. Flask is a micro web framework written in Python. Flask-RESTful is an extension for Flask that adds support for quickly building REST APIs. Flask-CORS Flask extension for handling Cross Origin Resource Sharing (CORS), making cross-origin AJAX possible.

**Note:** Flask is micro framework with plenty of capabilities. We are using it only for creating Rest API.

## 4.6 Vue.js and Axios

The project provide simple user interface. The html and Vue.js is used for the front end User Interface. Vue.js is an open-source JavaScript framework for building user interfaces and single-page applications.For asynchronous calls the Axios Library is used. The Axios is the http client.

**Note:** Instead of vue-resource of Vue.js the Axios http client is used.

## 4.7 Operating System

Ubuntu OS is used.

## 4.8 Integrated Development Environment (IDE)

- Visual studio code IDE and Spyder IDE are used for writing code.

## 4.9 Documentation

- The sphnix document tool is used for documenting the project as the project is python based.

## 4.10 Libraries and their version

The below is the table of SW/libs and their version used in this project.

| **Description** | **Version** |
|---|---|
| Ubuntu | 18.04.4.LTS |
| Python | 3.6.9 |
| pip | 20.1.1 |
| beautiful soup | 4.8.2 |
| bs4 package | 0.0.1 |
| Flask | 1.1.2 |
| Flask - Cors | 3.0.3 |
| Flask - Restful | 0.3.8 |
| urlib | 1.22 |
| vue/cli | 4.4.1 |

CHAPTER

FIVE

# PROTOTYPE CODE

## 5.1 Proof of Concept Code

The following python program requests html data from the urls given in the list. It then uses BS4 to parse the html pages and hence specifically scraps all href links. If the word "tender" is found in the scraped links it is written to text file.

The code is listed below:

```
webscraper.py

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May 13 12:50:41 2020

@author: venkat
"""

from bs4 import BeautifulSoup
import urllib.request
import re

urls_list= ["https://eprocure.gov.in/cppp/","https://eprocure.gov.in/cppp/
↪latestactivetendersnew/cpppdata"]#This takes          list of urls required
for url in urls_list:
    page = urllib.request.urlopen(url)#This line requests the url for the HTML page

    try:
        page = urllib.request.urlopen(url)
    except:
        print("An error occured.")

    soup = BeautifulSoup(page, 'html.parser')#This parses the HTMl
    #print(soup.prettify())# This prints the whole webpage in Html format

    content_list=[]

    for link in soup.find_all('a', href=True):#This finds all href link tags
    #print(link['href'])
        content_list.append(str(link['href']))#All the href links are saved inside␣
↪the content_list
    f=open("tender.txt",'a')#We are creating a text file
    for i in content_list:
```

(continues on next page)

(continued from previous page)

```
        if 'tender' in i:#We are searching for the keyword. I Have directly hardcoded␣
↪here but it can also be placed                     in a variable or in a list
            f.write('%s\n' % i)#I am writing the results to the text file
            print(i)
    f.close()
```

The following python program requests html data from the url assigned to variable. It is then written to file in a byte format. The key word to be found is also given in byte format. If keyword it is printed on the terminal screen.

The code is listed below:

```
websearch.py
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May 13 17:51:21 2020

@author: venkat
"""
'''
This program differs from the web_scraper program in such a way that it searches the␣
↪given keyword from the requested HTML and prints all the tags related in bytestring␣
↪form
'''
import sys
import re
import urllib

link="https://eprocure.gov.in/cppp/"
lines = urllib.request.urlopen(link).readlines()
f=open("all.txt",'a')
for line in lines:
    if b'tender' in line:
        f.write('%s\n' % line)
        #print(line)
f.close()
```

The following python program shows a flask based web application. At the back end python program does scraping.

The code is listed below:

```
flask_app.py:
from flask import Flask,render_template,request
from web_scraper import web_scraper

app=Flask(__name__)
app.config["DEBUG"]=True

@app.route('/', methods=["GET","POST"])
def scraper_page():
    if request.method == "POST":
        URL1=None

        keyword=None
        URL1=str(request.form["URL1"])

        keyword=str(request.form["keyword"])
```

(continues on next page)

(continued from previous page)

```
        if URL1 is not None and keyword is not None:
            result1=web_scraper(URL1,keyword)

            return render_template('result.html').format(result1=result1)
    return render_template('index.html')
if __name__ == '__main__':
   app.run(debug = True)




web_scraper.py:

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May 13 12:50:41 2020

@author: venkat
"""

from bs4 import BeautifulSoup
import urllib.request
import re

#urls_list= ["https://eprocure.gov.in/cppp/","https://eprocure.gov.in/cppp/
↪latestactivetendersnew/cpppdata"]#This takes list of urls required
def web_scraper(given_url,keyword):

    page = urllib.request.urlopen(given_url)#This line requests the url for the HTML␣
↪page

    try:
        page = urllib.request.urlopen(given_url)
    except:
        print("An error occured.")

    soup = BeautifulSoup(page, 'html.parser')#This parses the HTMl
    #print(soup.prettify())# This prints the whole webpage in Html format

    content_list=[]
    keyword_list=[]

    for link in soup.find_all('a', href=True):#This finds all href link tags
    #print(link['href'])
        content_list.append(str(link['href']))#All the href links are saved inside␣
↪the content_list

    #return content_list

    for i in content_list:
        if keyword in i:
            keyword_list.append(i)

    return keyword_list

index.html:
```

(continues on next page)

(continued from previous page)

```
<html>

    <body>
    <p>Enter the URLs on which Web Scraping should take place</p>
    <form method="POST" action=".">
        <p><input name="URL1"/></p>

        <p>Enter the keyword you want to search</p>
        <p><input name="keyword"/></p>
        <p><input type='submit' value='Get Data!!' /></p>
    </form>
    </body>
</html>

result.html:
<html>

<body>
    <p>The result of first URL is {result1}</p>

    <p><a href="/">Click here to perform again</a>
</body>

</html>
```

## 5.2 Prototype code

The prototype code used for the final project code is listed below

The below code scrapes the tables in the url and stores them as csv file:

```
many_info.py:
#import libraries
import requests
import pandas as pd
import csv
import urllib.request
from bs4 import BeautifulSoup
url = 'https://eprocure.gov.in/cppp/latestactivetendersnew/cpppdata'
response = requests.get(url)
print(response.status_code)
soup = BeautifulSoup(response.text,"html.parser")
table = soup.findAll('table')[0]
tr = table.findAll(['tr'])[1:11]
csvFile = open("eprocure.csv",'wt',newline='', encoding='utf-8')
writer = csv.writer(csvFile)
try:
    for cell in tr:
        th = cell.find_all('th')
        th_data = [col.text.strip('\n') for col in th]
        td = cell.find_all('td')
        row = [i.text.replace('\n','') for i in td]
        writer.writerow(th_data+row)
```

(continues on next page)

(continued from previous page)

```
finally:
    csvFile.close()
```

This code is used for scraping multiple url pages and find the given key word:

```
multiurl.py:
#This code is for the first update
#The problem to be addressed is to add multiple pages without hardcoding
#This code is for multiple pages within the link

#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed May 13 12:50:41 2020

@author: venkat
"""

from bs4 import BeautifulSoup
import urllib.request


urls_list_central=[]
urls_list_state=[]
words_to_be_searched_list=[]

for i in range(2,10,1):
    urls_list_state.append("https://eprocure.gov.in/cppp/latestactivetendersnew/
↪cpppdata?page="+str(i))
#urls_list= ["https://eprocure.gov.in/cppp/","https://eprocure.gov.in/cppp/
↪latestactivetendersnew/cpppdata"]#This takes list of urls required
for url in urls_list_state:
    page = urllib.request.urlopen(url)#This line requests the url for the HTML page

    try:
        page = urllib.request.urlopen(url)
    except:
        print("An error occured.")

    soup = BeautifulSoup(page, 'html.parser')#This parses the HTMl
    #print(soup.prettify())# This prints the whole webpage in Html format

    content_list=[]

    for link in soup.find_all('a', href=True):#This finds all href link tags
    #print(link['href'])
        content_list.append(str(link['href']))#All the href links are saved inside␣
↪the content_list
    f=open("Tender.txt",'a')#We are creating a text file
    for i in content_list:
        if 'Tender' in i:#We are searching for the keyword. I Have directly hardcoded␣
↪here but it can also be placed in a variable or in a list
            f.write('%s\n' % i)#I am writing the results to the text file
            print(i)
    f.close()
```

This code scraps multiple urls given in a text file and scraps for multiple keywords given in the list:

```
multiurl_part2:
#This code uses the concept of files.
import urllib
from bs4 import BeautifulSoup
contents=["Improvement","Construction","water","TRANSPORTATION"]
URL_list=[]

#Here we upload a file containing all the URLs
file1=open("URL.txt","r")
line=file1.readlines()
#print(line)



for url in line:
        html = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(html,'html.parser')
        links = soup.find_all('a')
        for l in links:
        #print(l.text)
                for c in contents:
                        if c in l.text:
                                print(c+":")
                                print(l)
```

This program is used for recursively searching for href links:

```
recursiv.py:
from bs4 import BeautifulSoup
import requests
import multiprocessing

s=set()

def get_links(url):
    response = requests.get(url)

    data = response.text
    soup = BeautifulSoup(data, 'lxml')

    links = set()
    for link in soup.find_all('a'):
        link_url = link.get('href')

        if link_url is not None and link_url.startswith('http') :
            links.add(link_url + '\n')


    write_to_file(links)
    return links


def write_to_file(links):
    with open('data.txt', 'a') as f:
        f.writelines(links)


```

(continues on next page)

(continued from previous page)

```
def get_all_links(url):
    for link in get_links(url):
        get_all_links(link)


r = 'https://eprocure.gov.in/cppp'
processes=[]
write_to_file([r])
for i in range(0,10):
    p = multiprocessing.Process(target=get_all_links(r), args=(i,))
    processes.append(p)
    p.start()

for process in processes:
    process.join()
write_to_file([r])
#get_all_links(r)
```

The below program uses the Scrapy to achieve recursive search for href links:

```
Spider (Recursive code):
from scrapy.spiders import CrawlSpider
from scrapy.spiders import Rule
from scrapy.linkextractors import LinkExtractor

from scrapy import Item
from scrapy import Field


class UrlItem(Item):
    url = Field()


class WikiSpider(CrawlSpider):
    name = 'eprocure'
    #allowed_domains = ['wikipedia.org']
    start_urls = ["https://eprocure.gov.in/cppp/latestactivetendersnew/cpppdata"]

    rules = (
        Rule(LinkExtractor(), callback='parse_url'),
    )

    def parse_url(self, response):
        item = UrlItem()
        item['url'] = response.url

        return item

#After doing this execute inside the webcrawler folder ==>  "scrapy crawl eprocure -o␣
↪eprocure.csv -t csv"


spider: multiscrap.py:
from scrapy.spiders import CrawlSpider
from scrapy.spiders import Rule
from scrapy.linkextractors import LinkExtractor
```

(continues on next page)

(continued from previous page)

```python
from scrapy import Item
from scrapy import Field


class UrlItem(Item):
    url = Field()


class WikiSpider(CrawlSpider):
    name = 'wiki'
    allowed_domains = ['wikipedia.org']
    start_urls = ['https://en.wikipedia.org/wiki/Main_Page/']

    rules = (
        Rule(LinkExtractor(), callback='parse_url'),
    )

    def parse_url(self, response):
        item = UrlItem()
        item['url'] = response.url

        return item
```

CHAPTER

SIX

# FINAL CODE

## 6.1 Final Code

The below program uses flask micro framework to create a RESTful API for the web scraper.The web scraper function is taken from a different python program Complete Scraper.py. The curl command for the flask program takes arguments of the content to be searched in the urls given in the text file.

The code is listed below:

```
Api.py

import os
from flask import Flask
from flask_restful import Resource, Api, reqparse
from flask_cors import CORS
from CompleteScraper import CompleteScraper

app=Flask(__name__)
CORS(app)
api=Api(app)

parser=reqparse.RequestParser()


parser.add_argument("content")

class scraper(Resource):
    def post(self):
        args=parser.parse_args()


        content=args["content"]
        output=CompleteScraper(content=content)
        return {"Output": output}

api.add_resource(scraper, "/scraper")

if __name__ == "__main__":
    app.run(debug=True)

# In the terminal run "curl http://localhost:5000/scraper -d "content="WHEAT"" -X␣
↪POST -v"
```

The following code defines a function which is used to scrap the href links as well as the tables from the given urls in the text file. If content (argument to the function) is present it will print links to a text file and table data to a csv file.

The code is listed below:

```
CompleteScraper.py:
from bs4 import BeautifulSoup
import urllib.request
import csv

def remove_html_tags(text):
    """Remove html tags from a string"""
    import re
    clean = re.compile('<.*?>')
    return re.sub(clean, '', text)

def CompleteScraper(content):
    file1=open("URL.txt","r")
    line=file1.readlines()

    headers=[]
    required_lines=[]
    for url in line:
        html = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(html,'html.parser')
        table = soup.findAll('table')[0]
        tr = table.findAll(['tr'])[0:11]
        links=soup.find_all('a')
        csvFile = open("OutputData.csv",'a',newline='', encoding='utf-8')
        writer = csv.writer(csvFile)
        f=open("OutputLinks.txt","a")
        try:
            for l in links:
                if content in l.text:
                    required_lines.append(str(l))
                    f.write(str(l))



            for cell in tr:
                th = cell.find_all('th')
                th_data = [col.text.strip('\n') for col in th]
                td = cell.find_all('td')
                row = [i.text.replace('\n','') for i in td]
                #print(row[4])
                if len(th_data)!=0:
                    required_lines.append(th_data)
                    writer.writerow(th_data)
                for i in row:
                    if content in i:
                        required_lines.append(row)
                        writer.writerow(row)

        finally:
            csvFile.close()
            f.close()
    return required_lines
```

This vue program presents the web application's Html file on the web page. This program also connect to the CompleteScraper.vue which contains backend. The code is listed below:

```
App.vue:
<template>
  <v-app>
    <v-app-bar app>
      <v-toolbar-title class="headline text-uppercase">
        <span class="font-weight-light">Web Scraper</span>
      </v-toolbar-title>
    </v-app-bar>
    <v-content>
      <CompleteScraper/>
    </v-content>
  </v-app>
</template>

<script>
import CompleteScraper from './components/CompleteScraper';

export default {
  name: 'App',

  components: {
    CompleteScraper,
  },

  data: () => ({
    //
  }),
};
</script>
```

This program designs the input field as well as submit button. It also has the axios component in order to get data from the api produced by the flask program. It also has the code for the loading spinner. The code is listed below:

```
CompleteScraper.vue:
<template>
  <v-container>
    <v-layout text-xs-center wrap>
      <v-flex>
        <!-- IMPORTANT PART! -->
        <form>
          <v-text-field v-model="content" label="Type the keyword to be searched"␣
↪required></v-text-field>
          <v-btn @click="submit">submit</v-btn>
          <v-btn @click="clear">clear</v-btn>
        </form>
        <br />


        <p> Note 1: The list of URLS at which scraping is happening is at URL.txt␣
↪file in current working directory where api.py exists</p>
        <p> Note 2: The output table is stored in OutputData.csv file in the current␣
↪working directory where api.py exists</p>
        <p> Note 3: The output links are stored in OutputLinks.txt file in current␣
↪working directory </p>
        <br />

    <div>
```

(continues on next page)

(continued from previous page)

```
        <div v-if="loading">
            <vue-simple-spinner size="huge" message="Scraping Now .. Pls Wait"/>
        </div>

        <div v-else>
            <h3>DONE!!!</h3>
        </div>

    </div>
        <br />



        <br />
        <br />
        <div v-for='items in outputClass' v-bind:key="items.id">
          <div v-for="elements in items" v-bind:key="elements.id">
            <p>{{ elements }}</p>
            <hr>
          </div>
          </div>


        <!-- END: IMPORTANT PART! -->
      </v-flex>
    </v-layout>
  </v-container>
</template><script>
import axios from "axios"
import VueSimpleSpinner from 'vue-simple-spinner'
export default {


  name: "CompleteScraper",
  data: () => ({
    loading:false,
    content: '',
    outputClass: []
  }),
  methods: {
    submit() {
      this.loading=true
      axios.post("http://127.0.0.1:5000/scraper", {

          content: this.content
        })
        .then((response) => {
          console.log(response.data);
          this.outputClass = response.data;

        })
        .catch(error => console.log(error))
        .finally(() => (this.loading = false))

    },
    clear() {
```

(continues on next page)

(continued from previous page)

```
      this.content = ""
      this.outputClass=[]
    }
  },
components: {
      VueSimpleSpinner
   }
};



</script>
```

CHAPTER

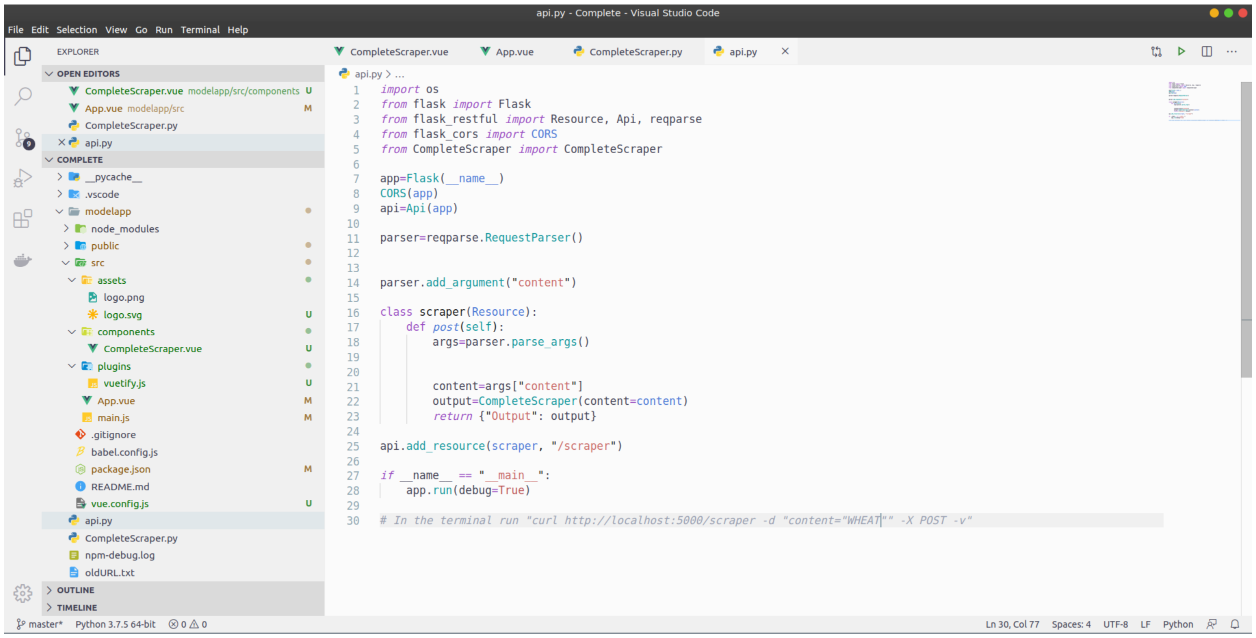
# SEVEN

# SNAPSHOTS

## 7.1 IDE Snapshots

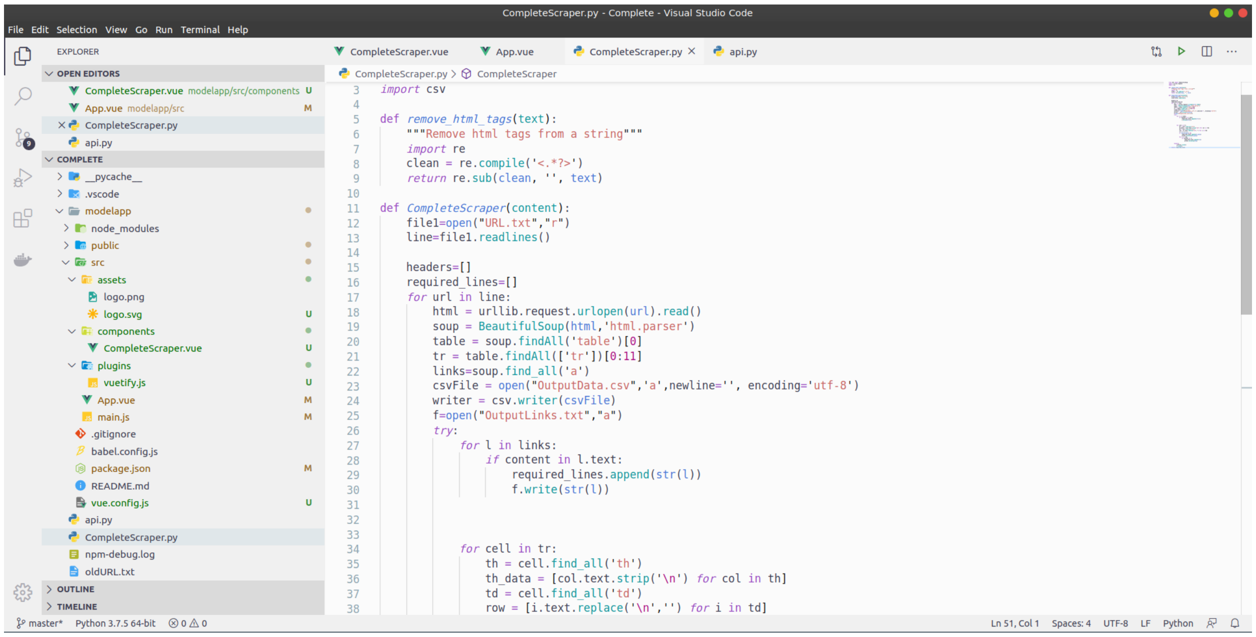
The Directory structure of the project is shown in the below screenshots.

The screen shot of api.py is shown below.



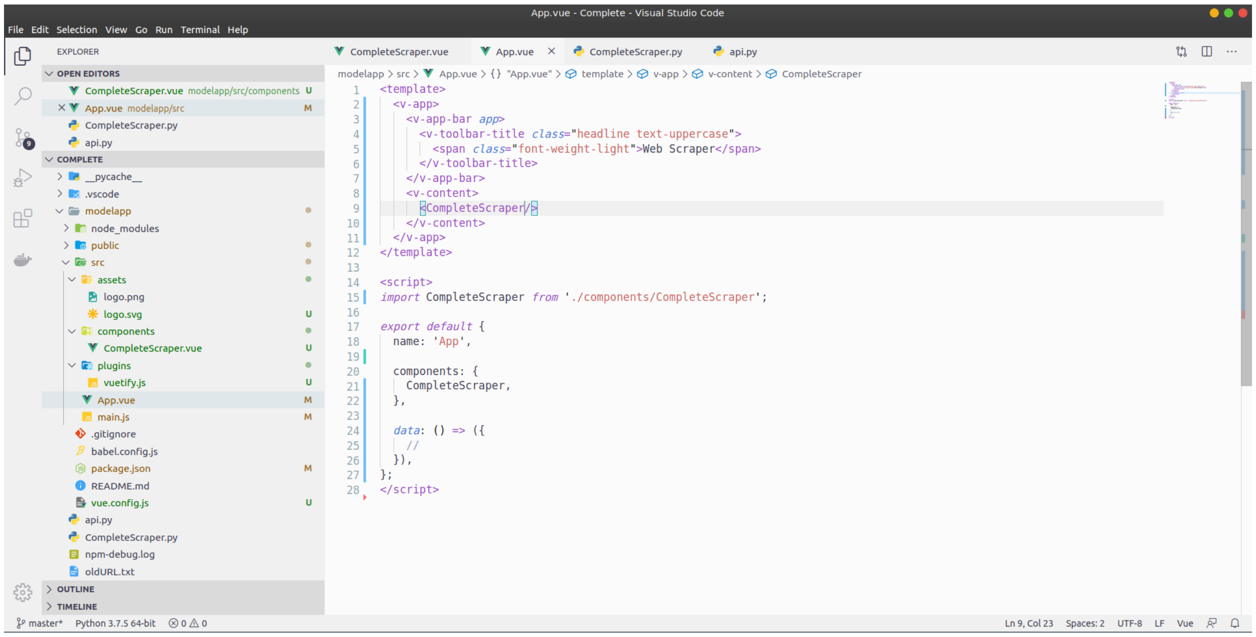
```python
import os
from flask import Flask
from flask_restful import Resource, Api, reqparse
from flask_cors import CORS
from CompleteScraper import CompleteScraper

app=Flask(__name__)
CORS(app)
api=Api(app)

parser=reqparse.RequestParser()


parser.add_argument("content")

class scraper(Resource):
    def post(self):
        args=parser.parse_args()


        content=args["content"]
        output=CompleteScraper(content=content)
        return {"Output": output}

api.add_resource(scraper, "/scraper")

if __name__ == "__main__":
    app.run(debug=True)

# In the terminal run "curl http://localhost:5000/scraper -d "content="WHEAT"" -X POST -v"
```

The screen shot of Complete scraper.py is shown below.



```python
import csv

def remove_html_tags(text):
    """Remove html tags from a string"""
    import re
    clean = re.compile('<.*?>')
    return re.sub(clean, '', text)

def CompleteScraper(content):
    file1=open("URL.txt","r")
    line=file1.readlines()

    headers=[]
    required_lines=[]
    for url in line:
        html = urllib.request.urlopen(url).read()
        soup = BeautifulSoup(html,'html.parser')
        table = soup.findAll('table')[0]
        tr = table.findAll(['tr'])[0:11]
        links=soup.find_all('a')
        csvFile = open("OutputData.csv",'a',newline='', encoding='utf-8')
        writer = csv.writer(csvFile)
        f=open("OutputLinks.txt","a")
        try:
            for l in links:
                if content in l.text:
                    required_lines.append(str(l))
                    f.write(str(l))



            for cell in tr:
                th = cell.find_all('th')
                th_data = [col.text.strip('\n') for col in th]
                td = cell.find_all('td')
                row = [i.text.replace('\n','') for i in td]
```

The screen shot of App.vue is shown below.

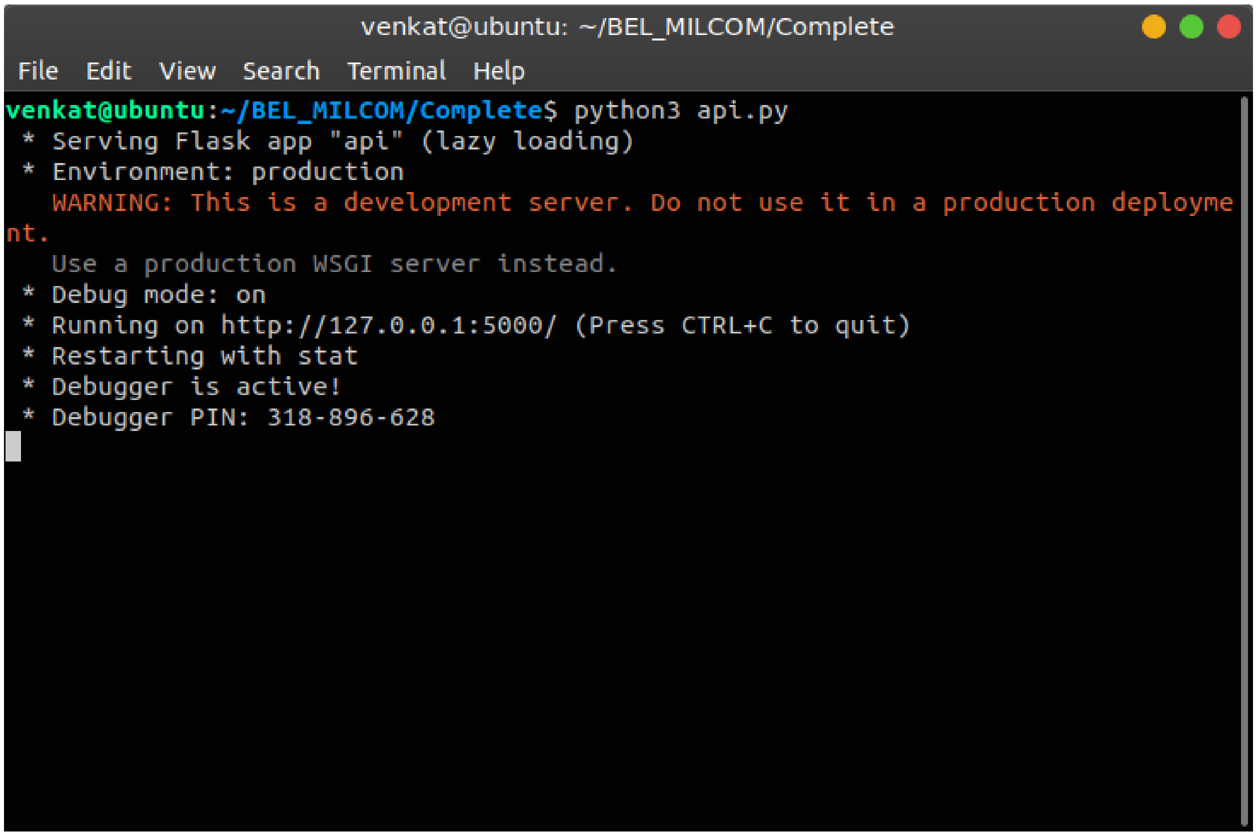


```html
<template>
  <v-app>
    <v-app-bar app>
      <v-toolbar-title class="headline text-uppercase">
        <span class="font-weight-light">Web Scraper</span>
      </v-toolbar-title>
    </v-app-bar>
    <v-content>
      <CompleteScraper/>
    </v-content>
  </v-app>
</template>

<script>
import CompleteScraper from './components/CompleteScraper';

export default {
  name: 'App',

  components: {
    CompleteScraper,
  },

  data: () => ({
    //
  }),
};
</script>
```

## 7.2 Starting of the Backend RestAPI

- Make sure that the port 8080 is not currently used by any other application
- open a new terminal and type python3 api.py at the directory where api.py exists.

The screenshot of the terminal on starting the RestAPI is shown below.

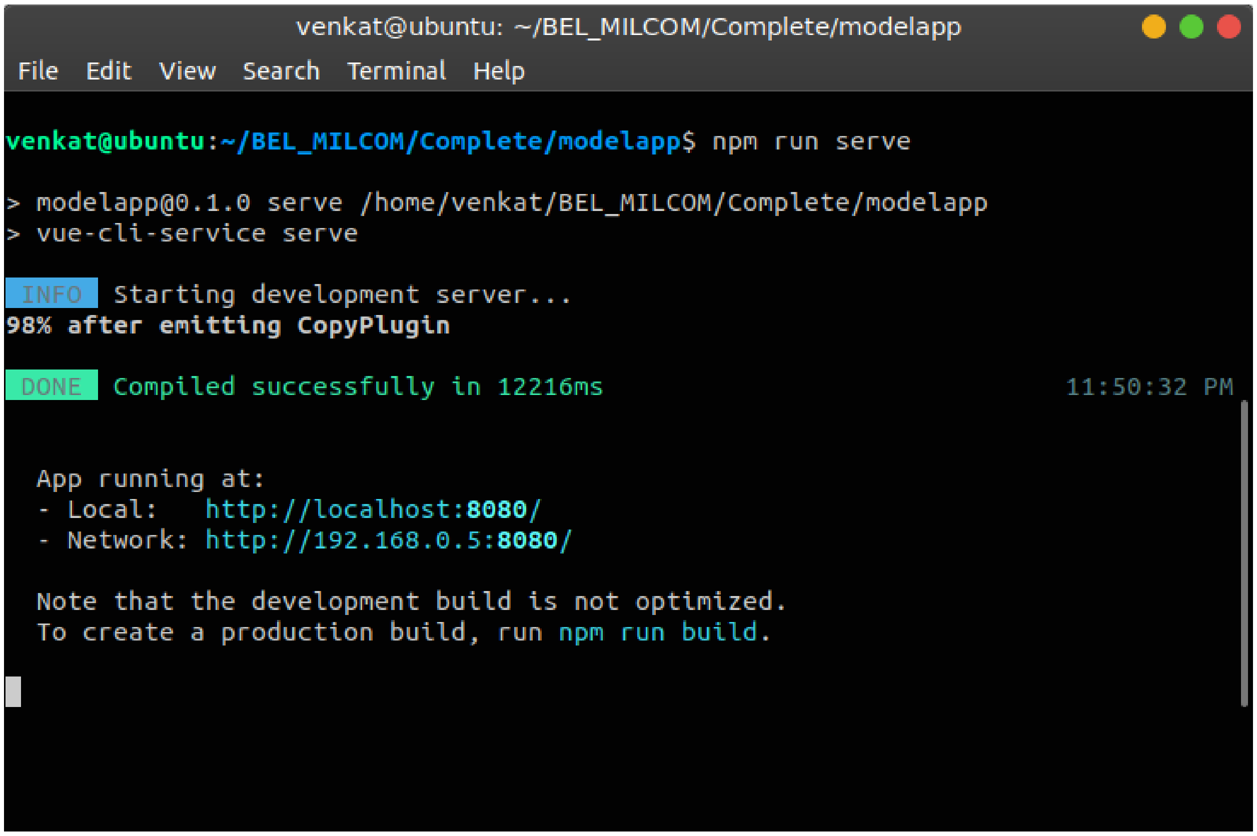

venkat@ubuntu: ~/BEL_MILCOM/Complete
File Edit View Search Terminal Help

```
venkat@ubuntu:~/BEL_MILCOM/Complete$ python3 api.py
 * Serving Flask app "api" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployme
nt.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 318-896-628
```

## 7.3 Starting of the Vue App using npm

- Make sure that the port 8080 is not currently used by any other application
- open a new terminal and type npm run serv at the directory where modelapp exists.

The screenshot of the terminal on starting of the Vue using npm is shown below:

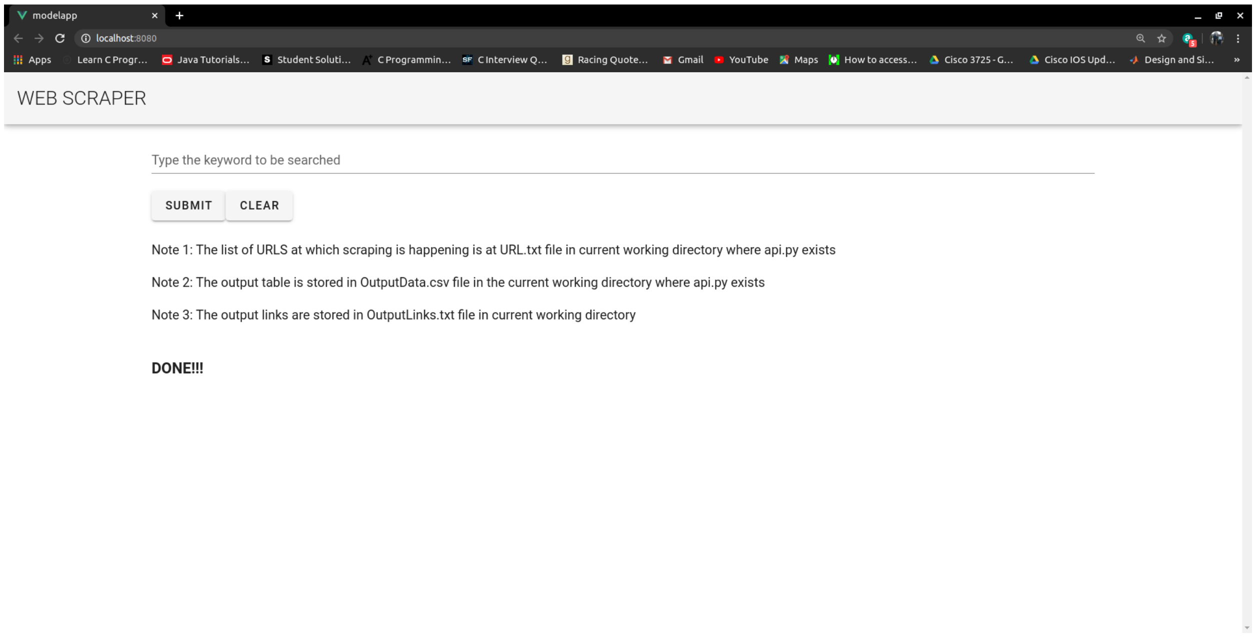

venkat@ubuntu: ~/BEL_MILCOM/Complete/modelapp
File Edit View Search Terminal Help

```
venkat@ubuntu:~/BEL_MILCOM/Complete/modelapp$ npm run serve

> modelapp@0.1.0 serve /home/venkat/BEL_MILCOM/Complete/modelapp
> vue-cli-service serve

 INFO  Starting development server...
98% after emitting CopyPlugin

 DONE  Compiled successfully in 12216ms                                  11:50:32 PM


  App running at:
  - Local:   http://localhost:8080/
  - Network: http://192.168.0.5:8080/

  Note that the development build is not optimized.
  To create a production build, run npm run build.
```

## 7.4 Opening the project web application at the browser

- Make sure Backend RestAPI running in a terminal.
- Make sure Vue App is running in another teminal.
- Make sure the text file containing the list of url is present in the require directory.

At the browser enter http://localhost:8080 The first page of the web application is shown below:



WEB SCRAPER

Type the keyword to be searched

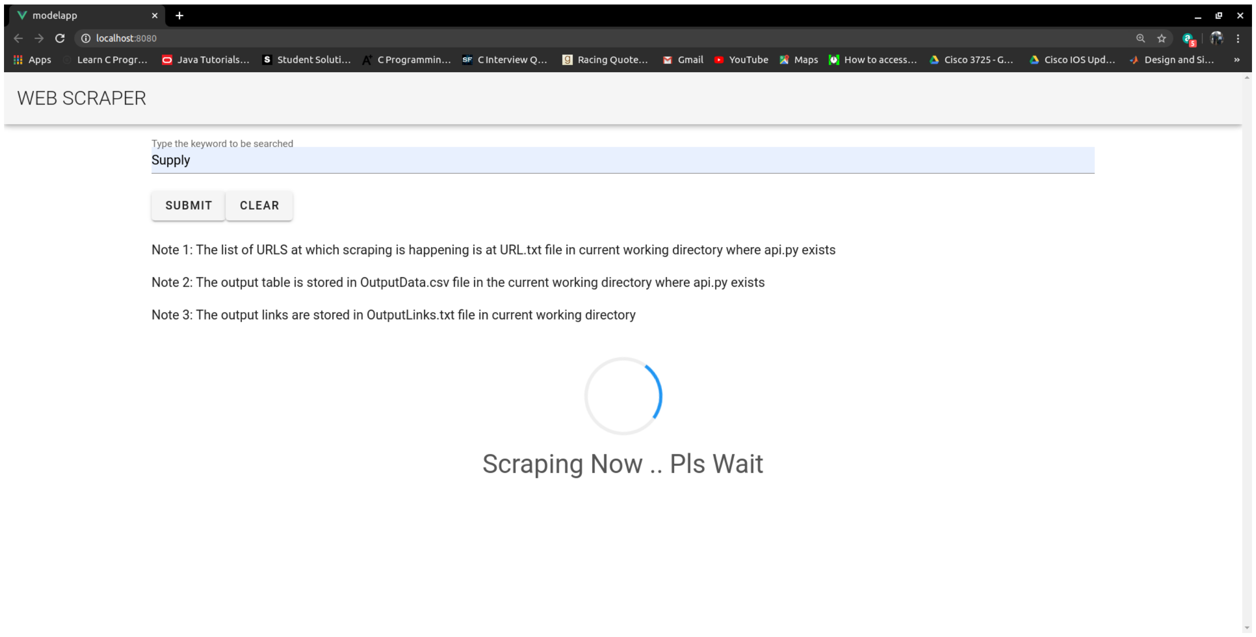
Note 1: The list of URLS at which scraping is happening is at URL.txt file in current working directory where api.py exists

Note 2: The output table is stored in OutputData.csv file in the current working directory where api.py exists

Note 3: The output links are stored in OutputLinks.txt file in current working directory

**DONE!!!**

Then enter the keyword. Then submit. The scraping - loading page appears.



WEB SCRAPER

Type the keyword to be searched

Supply

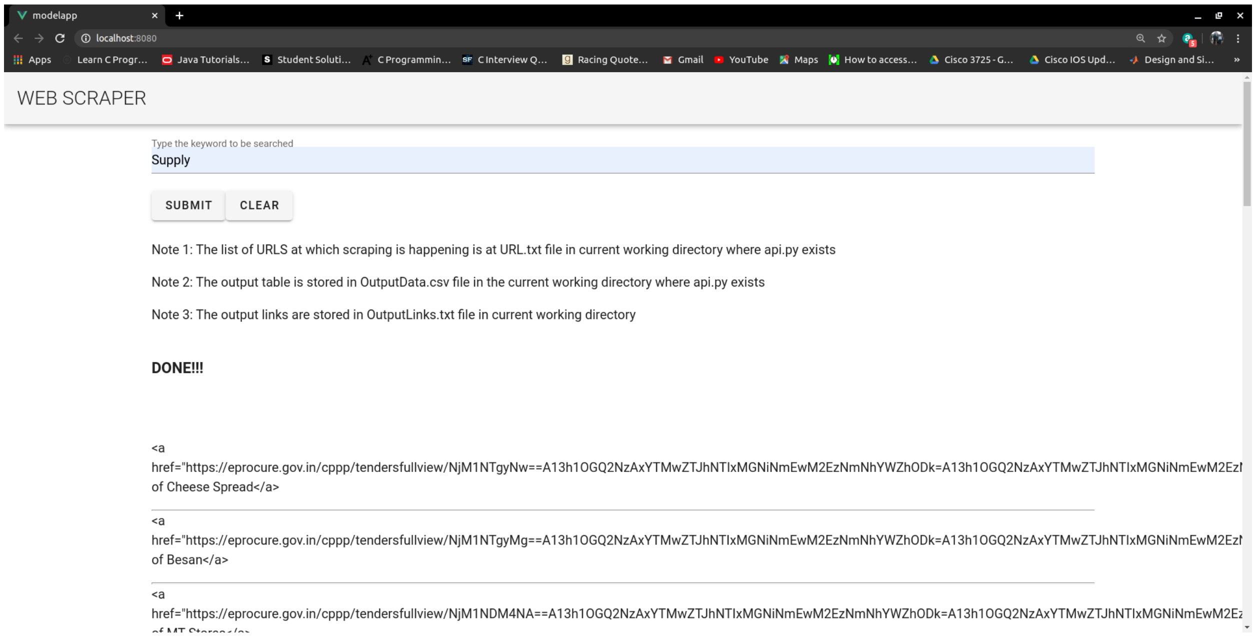
Note 1: The list of URLS at which scraping is happening is at URL.txt file in current working directory where api.py exists

Note 2: The output table is stored in OutputData.csv file in the current working directory where api.py exists

Note 3: The output links are stored in OutputLinks.txt file in current working directory

Scraping Now .. Pls Wait

After the scraping is done the the extracted data is displayed.



WEB SCRAPER

Type the keyword to be searched
Supply

Note 1: The list of URLS at which scraping is happening is at URL.txt file in current working directory where api.py exists

Note 2: The output table is stored in OutputData.csv file in the current working directory where api.py exists

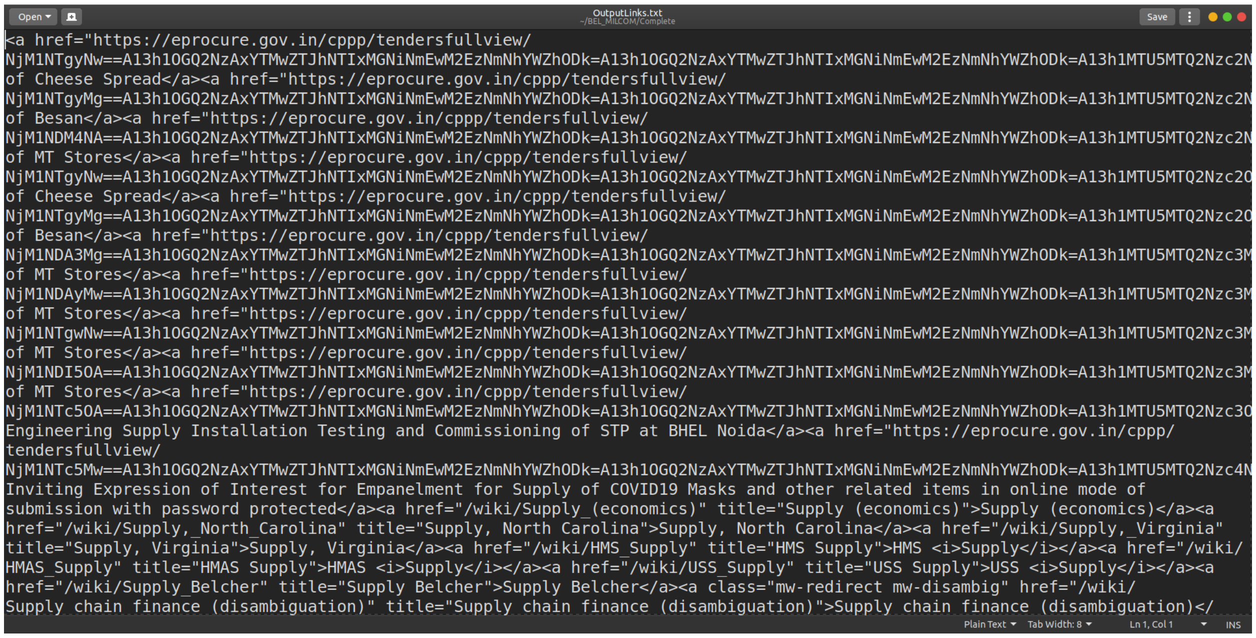
Note 3: The output links are stored in OutputLinks.txt file in current working directory

**DONE!!!**

```
<a
href="https://eprocure.gov.in/cppp/tendersfullview/NjM1NTgyNw==A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzN
of Cheese Spread</a>

<a
href="https://eprocure.gov.in/cppp/tendersfullview/NjM1NTgyMg==A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzN
of Besan</a>

<a
href="https://eprocure.gov.in/cppp/tendersfullview/NjM1NDM4NA==A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2Ez
```

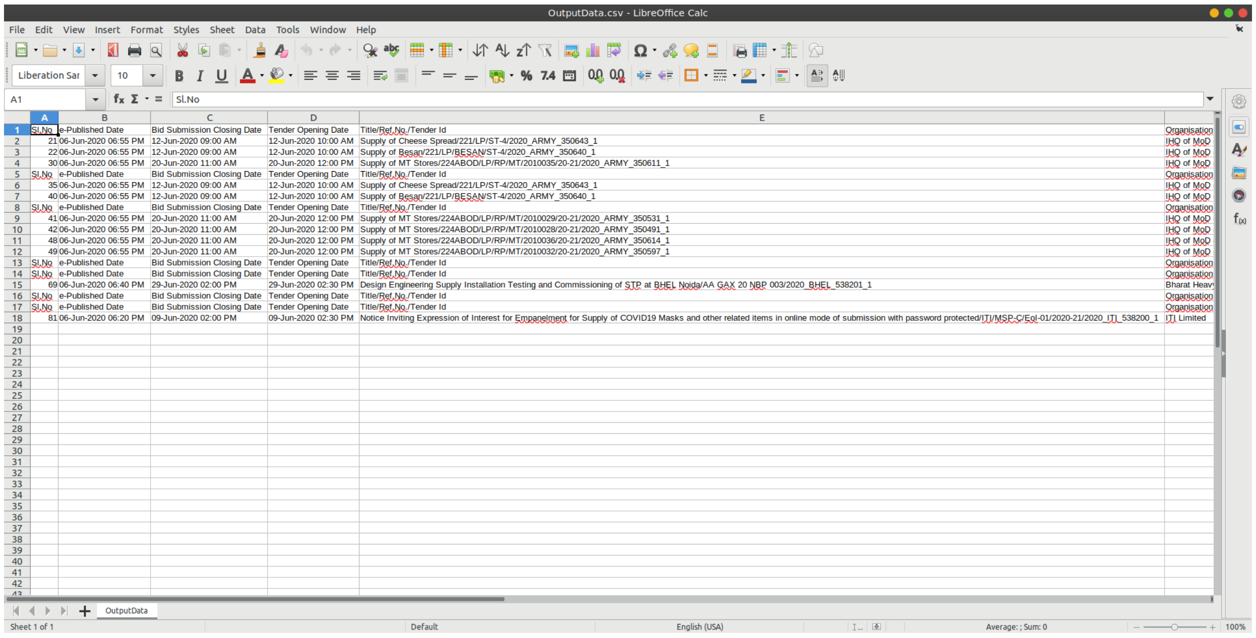
## 7.5 Output Files

The href scraped from the urls are stored in a text file. The screen shot of the file is shown below.



```
<a href="https://eprocure.gov.in/cppp/tendersfullview/
NjM1NTgyNw==A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1MTU5MTQ2Nzc2N
of Cheese Spread</a><a href="https://eprocure.gov.in/cppp/tendersfullview/
NjM1NTgyMg==A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1MTU5MTQ2Nzc2N
of Besan</a><a href="https://eprocure.gov.in/cppp/tendersfullview/
NjM1NDM4NA==A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1MTU5MTQ2Nzc2N
of MT Stores</a><a href="https://eprocure.gov.in/cppp/tendersfullview/
NjM1NTgyNw==A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1MTU5MTQ2Nzc2O
of Cheese Spread</a><a href="https://eprocure.gov.in/cppp/tendersfullview/
NjM1NTgyMg==A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1MTU5MTQ2Nzc2O
of Besan</a><a href="https://eprocure.gov.in/cppp/tendersfullview/
NjM1NDA3Mg==A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1MTU5MTQ2Nzc3M
of MT Stores</a><a href="https://eprocure.gov.in/cppp/tendersfullview/
NjM1NDAyMw==A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1MTU5MTQ2Nzc3M
of MT Stores</a><a href="https://eprocure.gov.in/cppp/tendersfullview/
NjM1NTgwNw==A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1MTU5MTQ2Nzc3M
of MT Stores</a><a href="https://eprocure.gov.in/cppp/tendersfullview/
NjM1NDI5OA==A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1MTU5MTQ2Nzc3M
of MT Stores</a><a href="https://eprocure.gov.in/cppp/tendersfullview/
NjM1NTc5OA==A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1MTU5MTQ2Nzc3O
Engineering Supply Installation Testing and Commissioning of STP at BHEL Noida</a><a href="https://eprocure.gov.in/cppp/
tendersfullview/
NjM1NTc5Mw==A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1OGQ2NzAxYTMwZTJhNTIxMGNiNmEwM2EzNmNhYWZhODk=A13h1MTU5MTQ2Nzc4N
Inviting Expression of Interest for Empanelment for Supply of COVID19 Masks and other related items in online mode of
submission with password protected</a><a href="/wiki/Supply_(economics)" title="Supply (economics)">Supply (economics)</a><a
href="/wiki/Supply,_North_Carolina" title="Supply, North Carolina">Supply, North Carolina</a><a href="/wiki/Supply,_Virginia"
title="Supply, Virginia">Supply, Virginia</a><a href="/wiki/HMS_Supply" title="HMS Supply">HMS <i>Supply</i></a><a href="/wiki/
HMAS_Supply" title="HMAS Supply">HMAS <i>Supply</i></a><a href="/wiki/USS_Supply" title="USS Supply">USS <i>Supply</i></a><a
href="/wiki/Supply_Belcher" title="Supply Belcher">Supply Belcher</a><a class="mw-redirect mw-disambig" href="/wiki/
Supply_chain_finance_(disambiguation)" title="Supply chain finance (disambiguation)">Supply chain finance (disambiguation)</
```

The contents of the tables which are scraped from the urls are stored in a cvs file. The screen shot of the file is shown below.



OutputData.csv - LibreOffice Calc

|  | A | B | C | D | E |  |
|---|---|---|---|---|---|---|
| 1 | Sl.No | e-Published Date | Bid Submission Closing Date | Tender Opening Date | Title/Ref.No./Tender Id | Organisation |
| 2 | 21 | 06-Jun-2020 06:55 PM | 12-Jun-2020 09:00 AM | 12-Jun-2020 10:00 AM | Supply of Cheese Spread/221/LP/ST-4/2020_ARMY_350643_1 | IHQ of MoD |
| 3 | 22 | 06-Jun-2020 06:55 PM | 12-Jun-2020 09:00 AM | 12-Jun-2020 10:00 AM | Supply of Besan/221/LP/BESAN/ST-4/2020_ARMY_350640_1 | IHQ of MoD |
| 4 | 30 | 06-Jun-2020 06:55 PM | 20-Jun-2020 11:00 AM | 20-Jun-2020 12:00 PM | Supply of MT Stores/224ABOD/LP/RP/MT/2010035/20-21/2020_ARMY_350611_1 | IHQ of MoD |
| 5 | Sl.No | e-Published Date | Bid Submission Closing Date | Tender Opening Date | Title/Ref.No./Tender Id | Organisation |
| 6 | 35 | 06-Jun-2020 06:55 PM | 12-Jun-2020 09:00 AM | 12-Jun-2020 10:00 AM | Supply of Cheese Spread/221/LP/ST-4/2020_ARMY_350643_1 | IHQ of MoD |
| 7 | 40 | 06-Jun-2020 06:55 PM | 12-Jun-2020 09:00 AM | 12-Jun-2020 10:00 AM | Supply of Besan/221/LP/BESAN/ST-4/2020_ARMY_350640_1 | IHQ of MoD |
| 8 | Sl.No | e-Published Date | Bid Submission Closing Date | Tender Opening Date | Title/Ref.No./Tender Id | Organisation |
| 9 | 41 | 06-Jun-2020 06:55 PM | 20-Jun-2020 11:00 AM | 20-Jun-2020 12:00 PM | Supply of MT Stores/224ABOD/LP/RP/MT/2010029/20-21/2020_ARMY_350531_1 | IHQ of MoD |
| 10 | 42 | 06-Jun-2020 06:55 PM | 20-Jun-2020 11:00 AM | 20-Jun-2020 12:00 PM | Supply of MT Stores/224ABOD/LP/RP/MT/2010028/20-21/2020_ARMY_350491_1 | IHQ of MoD |
| 11 | 48 | 06-Jun-2020 06:55 PM | 20-Jun-2020 11:00 AM | 20-Jun-2020 12:00 PM | Supply of MT Stores/224ABOD/LP/RP/MT/2010036/20-21/2020_ARMY_350614_1 | IHQ of MoD |
| 12 | 49 | 06-Jun-2020 06:55 PM | 20-Jun-2020 11:00 AM | 20-Jun-2020 12:00 PM | Supply of MT Stores/224ABOD/LP/RP/MT/2010032/20-21/2020_ARMY_350597_1 | IHQ of MoD |
| 13 | Sl.No | e-Published Date | Bid Submission Closing Date | Tender Opening Date | Title/Ref.No./Tender Id | Organisation |
| 14 | Sl.No | e-Published Date | Bid Submission Closing Date | Tender Opening Date | Title/Ref.No./Tender Id | Organisation |
| 15 | 69 | 06-Jun-2020 06:40 PM | 29-Jun-2020 02:00 PM | 29-Jun-2020 02:30 PM | Design Engineering Supply Installation Testing and Commissioning of STP at BHEL Noida/AA GAX 20 NBP 003/2020_BHEL_538201_1 | Bharat Heav |
| 16 | Sl.No | e-Published Date | Bid Submission Closing Date | Tender Opening Date | Title/Ref.No./Tender Id | Organisation |
| 17 | Sl.No | e-Published Date | Bid Submission Closing Date | Tender Opening Date | Title/Ref.No./Tender Id | Organisation |
| 18 | 81 | 06-Jun-2020 06:20 PM | 09-Jun-2020 02:00 PM | 09-Jun-2020 02:30 PM | Notice Inviting Expression of Interest for Empanelment for Supply of COVID19 Masks and other related items in online mode of submission with password protected/ITI/MSP-C/EoI-01/2020-21/2020_ITI_538200_1 | ITI Limited |

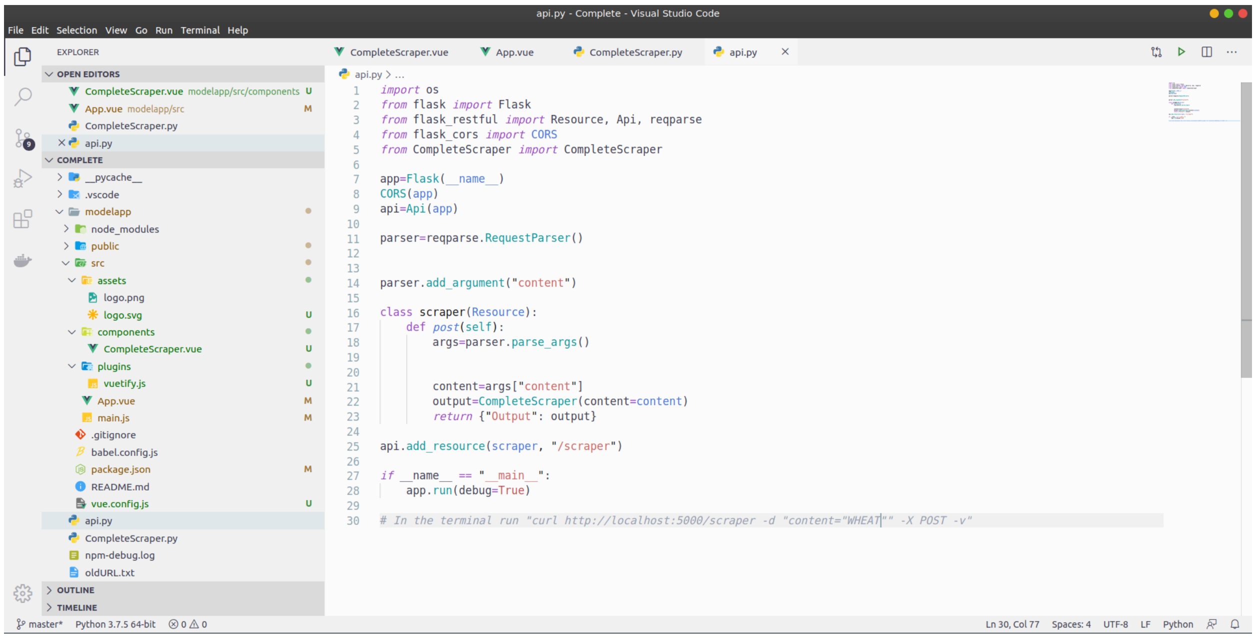
CHAPTER

# EIGHT

# TROUBLESHOOTING

## 8.1 Coding and Compiling

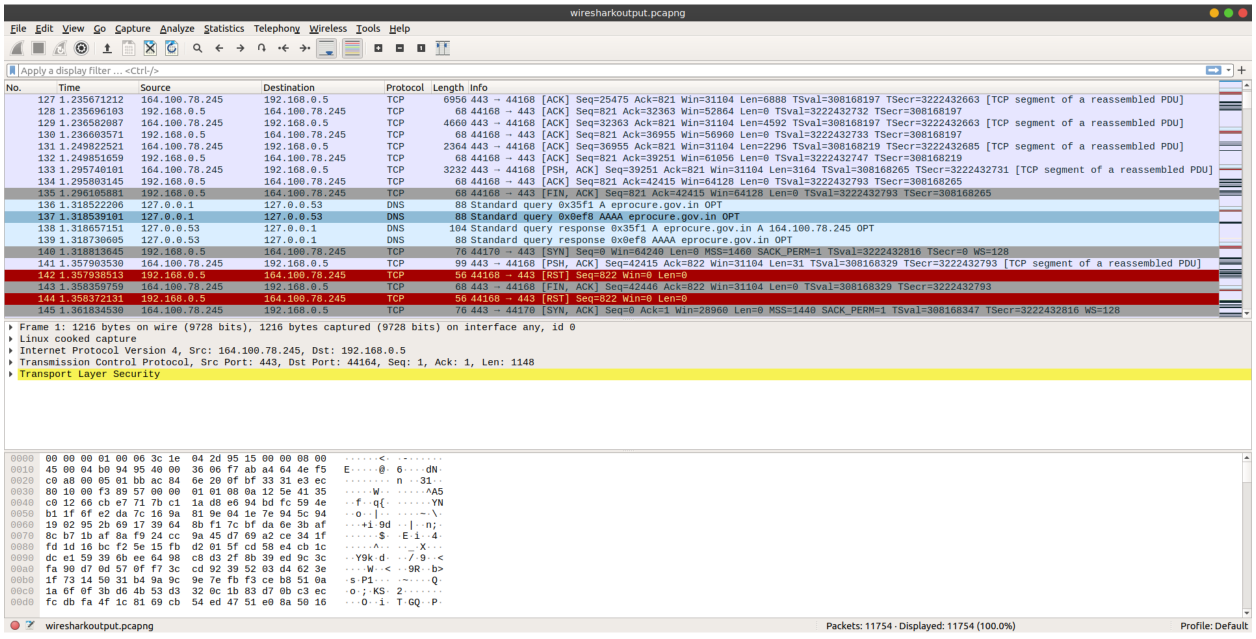
The Visual Code IDE and Spider IDE are used for coding.



```python
import os
from flask import Flask
from flask_restful import Resource, Api, reqparse
from flask_cors import CORS
from CompleteScraper import CompleteScraper

app=Flask(__name__)
CORS(app)
api=Api(app)

parser=reqparse.RequestParser()


parser.add_argument("content")

class scraper(Resource):
    def post(self):
        args=parser.parse_args()


        content=args["content"]
        output=CompleteScraper(content=content)
        return {"Output": output}

api.add_resource(scraper, "/scraper")

if __name__ == "__main__":
    app.run(debug=True)

# In the terminal run "curl http://localhost:5000/scraper -d "content="WHEAT"" -X POST -v"
```

## 8.2 Network Analysis

The tools such as Wireshark or tcpdump are used for network analysis.


wiresharkoutput.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
wiresharkoutput.pcapng
Packets: 11754 · Displayed: 11754 (100.0%)
Profile: Default

```
No.   Time         Source          Destination     Protocol Length Info
  127 1.235671212  164.100.78.245  192.168.0.5     TCP      6956 443 → 44168 [ACK] Seq=25475 Ack=821 Win=31104 Len=6888 TSval=308168197 TSecr=3222432663 [TCP segment of a reassembled PDU]
  128 1.235696103  192.168.0.5     164.100.78.245  TCP        68 44168 → 443 [ACK] Seq=821 Ack=32363 Win=52864 Len=0 TSval=3222432732 TSecr=308168197
  129 1.236582087  164.100.78.245  192.168.0.5     TCP      4660 443 → 44168 [ACK] Seq=32363 Ack=821 Win=31104 Len=4592 TSval=308168197 TSecr=3222432663 [TCP segment of a reassembled PDU]
  130 1.236603571  192.168.0.5     164.100.78.245  TCP        68 44168 → 443 [ACK] Seq=821 Ack=36955 Win=56960 Len=0 TSval=3222432733 TSecr=308168197
  131 1.249822521  164.100.78.245  192.168.0.5     TCP      2364 443 → 44168 [ACK] Seq=36955 Ack=821 Win=31104 Len=2296 TSval=308168219 TSecr=3222432685 [TCP segment of a reassembled PDU]
  132 1.249851659  192.168.0.5     164.100.78.245  TCP        68 44168 → 443 [ACK] Seq=821 Ack=39251 Win=61056 Len=0 TSval=3222432747 TSecr=308168219
  133 1.295740101  164.100.78.245  192.168.0.5     TCP      3232 443 → 44168 [PSH, ACK] Seq=39251 Ack=821 Win=31104 Len=3164 TSval=308168265 TSecr=3222432731 [TCP segment of a reassembled PDU]
  134 1.295803145  192.168.0.5     164.100.78.245  TCP        68 44168 → 443 [ACK] Seq=821 Ack=42415 Win=64128 Len=0 TSval=3222432793 TSecr=308168265
  135 1.296105881  192.168.0.5     164.100.78.245  TCP        68 44168 → 443 [FIN, ACK] Seq=821 Ack=42415 Win=64128 Len=0 TSval=3222432793 TSecr=308168265
  136 1.318522206  127.0.0.1       127.0.0.53      DNS        88 Standard query 0x35f1 A eprocure.gov.in OPT
  137 1.318539101  127.0.0.1       127.0.0.53      DNS        88 Standard query 0x0ef8 AAAA eprocure.gov.in OPT
  138 1.318657151  127.0.0.53      127.0.0.1       DNS       104 Standard query response 0x35f1 A eprocure.gov.in A 164.100.78.245 OPT
  139 1.318730605  127.0.0.53      127.0.0.1       DNS        88 Standard query response 0x0ef8 AAAA eprocure.gov.in OPT
  140 1.318813645  192.168.0.5     164.100.78.245  TCP        76 44170 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=3222432816 TSecr=0 WS=128
  141 1.357903530  164.100.78.245  192.168.0.5     TCP        99 443 → 44168 [PSH, ACK] Seq=42415 Ack=822 Win=31104 Len=31 TSval=308168329 TSecr=3222432793 [TCP segment of a reassembled PDU]
  142 1.357938513  192.168.0.5     164.100.78.245  TCP        56 44168 → 443 [RST] Seq=822 Win=0 Len=0
  143 1.358359759  164.100.78.245  192.168.0.5     TCP        68 443 → 44168 [FIN, ACK] Seq=42446 Ack=822 Win=31104 Len=0 TSval=308168329 TSecr=3222432793
  144 1.358372131  192.168.0.5     164.100.78.245  TCP        56 44168 → 443 [RST] Seq=822 Win=0 Len=0
  145 1.361834530  164.100.78.245  192.168.0.5     TCP        76 443 → 44170 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1440 SACK_PERM=1 TSval=308168347 TSecr=3222432816 WS=128
```

```
▸ Frame 1: 1216 bytes on wire (9728 bits), 1216 bytes captured (9728 bits) on interface any, id 0
▸ Linux cooked capture
▸ Internet Protocol Version 4, Src: 164.100.78.245, Dst: 192.168.0.5
▸ Transmission Control Protocol, Src Port: 443, Dst Port: 44164, Seq: 1, Ack: 1, Len: 1148
▸ Transport Layer Security
```

```
0000  00 00 00 01 00 06 3c 1e  04 2d 95 15 00 00 08 00   ······<· ·-······
0010  45 00 04 b0 94 95 40 00  36 06 f7 ab a4 64 4e f5   E·····@· 6····dN·
0020  c0 a8 00 05 01 bb ac 84  6e 20 0f bf 33 31 e3 ec   ········ n ··31··
0030  80 10 00 f3 89 57 00 00  01 01 08 0a 12 5e 41 35   ·····W·· ·····^A5
0040  c0 12 66 cb e7 71 7b c1  1a d8 e6 94 bd fc 59 4e   ··f··q{· ······YN
0050  b1 1f 6f e2 da 7c 16 9a  81 9e 04 1e 7e 94 5c 94   ··o··|·· ····~·\·
0060  19 02 95 2b 69 17 39 64  8b f1 7c bf da 6e 3b af   ···+i·9d ··|··n;·
0070  8c b7 1b af 8a f9 24 cc  9a 45 d7 69 a2 ce 34 1f   ······$· ·E·i··4·
0080  fd 1d 16 bc f2 5e 15 fb  d2 01 5f cd 58 e4 cb 1c   ·····^·· ··_·X···
0090  dc e1 59 39 6b ee 64 98  c8 d3 2f 8b 39 ed 9c 3c   ··Y9k·d· ··/·9··<
00a0  fa 90 d7 0d 57 0f f7 3c  cd 92 39 52 03 d4 62 3e   ····W··< ··9R··b>
00b0  1f 73 14 50 31 b4 9a 9c  9e 7e fb f3 ce b8 51 0a   ·s·P1··· ·~····Q·
00c0  1a 6f 0f 3b d6 4b 53 d3  32 0c 1b 83 d7 0b c3 ec   ·o·;·KS· 2·······
00d0  fc db fa 4f 1c 81 69 cb  54 ed 47 51 e0 8a 50 16   ···O··i· T·GQ··P·
```

## 8.3 Frequently happening lessons

- Some other application will be running on port number 8080.
- api.py is not started in the new terminal or that terminal is closed.
- vue application is not started in the new terminal or that terminal is closed.
- The firewall is in the ON condition.
- The read/write permissions not properly given for the user.

CHAPTER

NINE

# CONCLUSION

The basic web scraping using python, beautifulsoup, csv, urllib requests, Vue.JS, flask, flask restful API, flask CORS, axios , arg parser, html parser and html was implemented.