Project Report - Week 1

SI100B Project Report - Crawler

Workload Division

- 颜毅恒（yanyh1@shanghaitech.edu.cn）& 彭琬迪（pengwd@shanghaitech.edu.cn）:

  Find how to send requests to the web, understand the meaning of the data crawled form the web.

- 苏慧哲（suhzh@shanghaitech.edu.cn or vhtmscyo@gmail.com）:

  Write python program and the report.

Preliminary Comments

Difficulties and Solutions

We want to write a crawler to get the data from the website. First, we should understand how normal people or the browser get the information from the website.

Understand How Browser Get the information

After searching the web, we find that browsers send HTTP messages to communicate with the server. HTTP is the abbriviation of Hypertext Transfer Protocol. We can find the definition of HTTP messages in MDN[1]:

> HTTP messages are how data is exchanged between a server and a client. There are two types of messages: requests sent by the client to trigger an action on the server, and responses, the answer from the server.

Since we want to sent requests to the server to ask for information, we should understand the requests message.

A request message is compose of three parts: start line, headers and body.

In the status line, an HTTP method, the request target and the HTTP version should be given.

Headers contains some useful information to be used by the server. Headers should follow the same basic structure:

> a case-insensitive string followed by a colon (`':'`) and a value whose structure depends upon the header.

Some requests need to fetch data from or send data to the server. These data are in body part. Not all requests have a body.

There're eight types of requests in total[2]:

> - `GET`
>
>   The `GET` method requests a representation of the specified resource. Requests using `GET` should only retrieve data.
> - `HEAD`
>
>   The `HEAD` method asks for a response identical to that of a `GET` request, but without the response body.
> - `POST`
>
>   The `POST` method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.
> - `PUT`
>
>   The `PUT` method replaces all current representations of the target resource with the request payload.
> - `DELETE`
>
>   The `DELETE` method deletes the specified resource.
> - `CONNECT`
>
>   The `CONNECT` method establishes a tunnel to the server identified by the target resource.
> - `OPTIONS`

> The `OPTIONS` method is used to describe the communication options for the target resource.
>
> - `TRACE`
>
>   The `TRACE` method performs a message loop-back test along the path to the target resource.
>
> - `PATCH`
>
>   The `PATCH` method is used to apply partial modifications to a resource.

`GET` and `POST` are the most commenly use two requests. In this project, we only need to understand `get` method so that we can get resources from the server.

After the requests accepted by the server, it will give back an HTTP response. It is also composed of three parts: status line, headers and body.

The status line contains the the protocol version, a status code and status text. The status code and the status text indicates whether the request is successed or failed. According to MDN[4]:

> Responses are grouped in five classes:
>
> 1. Informational responses (`100` – `199`)
> 2. Successful responses (`200` – `299`)
> 3. Redirects (`300` – `399`)
> 4. Client errors (`400` – `499`)
> 5. Server errors (`500` – `599`)

And following are some commonly seen status[5]:

> 200 OK - The request has succeeded.
>
> 204 No Content - There is no content to send for this request, but the headers may be useful.
>
> 301 Moved Permanently - The URL of the requested resource has been changed permanently. The new URL is given in the response.
>
> 304 Not Modified - This is used for caching purposes. It tells the client that the response has not been modified, so the client can continue to use the same cached version of the response.
>
> 404 Not Found - The server can not find the requested resource.
>
> 500 Internal Server Error- The server has encountered a situation it doesn't know how to handle.

The headers are similar to the headers of the HTTP requests. The structure are strictly the same, but the type of the headers may be different.

The body contains the data given by the server. The information we want should be in the body.

It not enough to understand the HTTP messages only if we want to actually send the massege. We should also understand the URLs.

Design a Valid URL

We search the wikipedia and find the syntax of a valid URL (Since a URL is a URI points to the resources on a network, it's syntax is the same as the URI). It should be composed of[6]:

- A non-empty **scheme** + `:`
- `//` + an **authority** component (optional)
  - A **userinfo** subcomponent + `@` (optional)
    - username
    - `:` + password (optional)
  - A **host** subcomponent
  - `:` + A **port** subcomponent (optional)
- A **path** component
- `?` + A **query** component (optional)
- `#` + A **fragment** component (optional)

There is also a more clear picture:

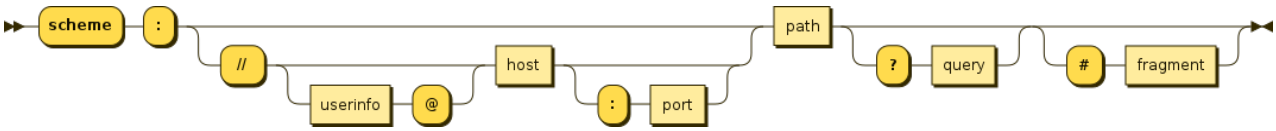

Fig.1 URI Syntanx[6]

Get the Information Using Python

In order to use python to send HTTP requests and parse HTTP responses, we decide to use requests library. Two of our team member failed to connect to flightradar24, so we decided to set fliaghtaware as our target website.

According to the first week's document of the project, before we sent the requests, we should first find the tocken on on the website. Through some simple searching, we can find that the token is written in the html file.

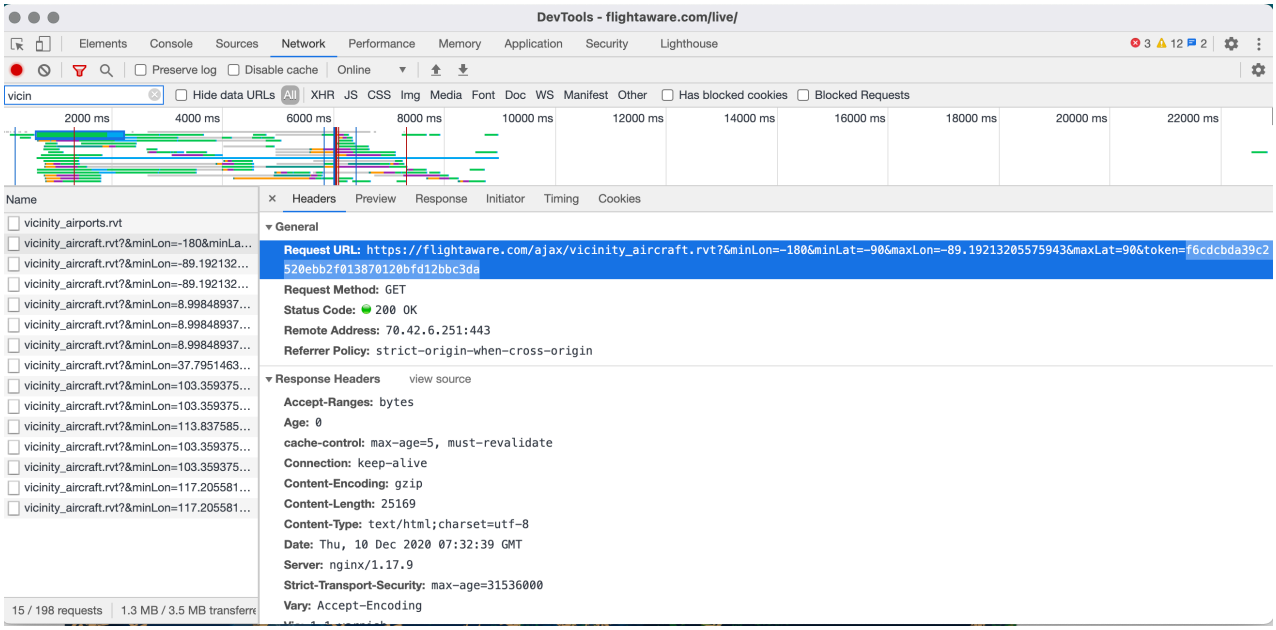

Fig.2 How to find the token - first step

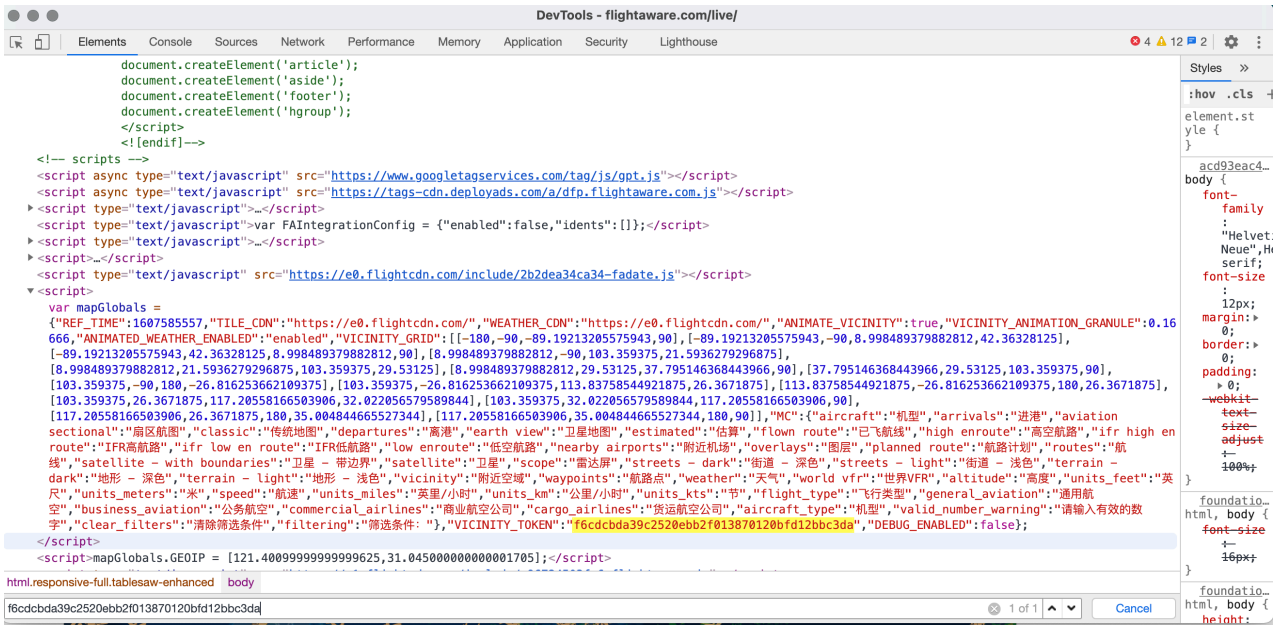

Fig.3 How to find the token - second step

We can first request the html file and find the token use regex[8].

```
import requests as rq
import re #regex pakage of python

tokenregx = '\"VICINITY_TOKEN\":\"([A-Za-z0-9]*)\"'

html = rq.get("https://flightaware.com/live/", timeout=5)

testurl = "https://flightaware.com/ajax/vicinity_aircraft.rvt"

token = re.search(tokenregx,html.text)
print(token.group(1))
```

Fig.4 Find token

Once we got the token we can send HTTP requests using requests:

```
estparam =
{"minLon":100,"minLat":11,"maxLon":143,"maxLat":51,"token":token.group(1)}

testquery = rq.get(testurl,params = testparam)

print(testquery)
t = testquery.json()
print(type(t))
print(t.keys())
```

Fig.5 Get data

`minLon` means the minimal longitude, `maxLon` is maximal longitude. Likewise, `minLat` is minimal Latitude and `maxLat` is maximal Latitude.

The whole data is too large, so we won't write it in this report, nor will we show it directly to the user. In this case, we should parse the data and display them in a human-friendly way.

Display the Information

First, take a look at the raw data:

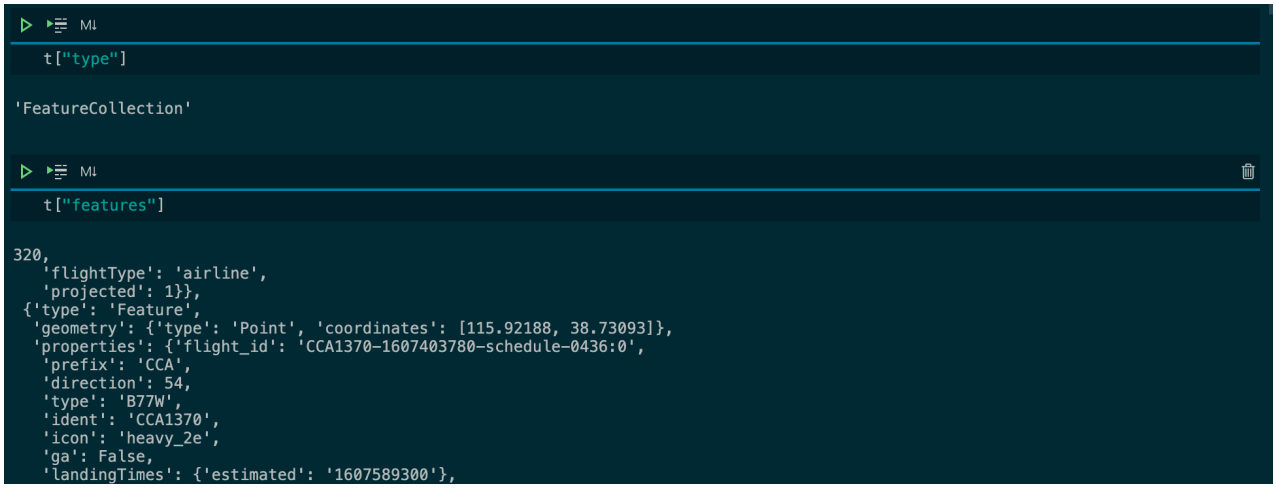

Fig.6 What's inside

We can easily find that the useful information is in the "feature".

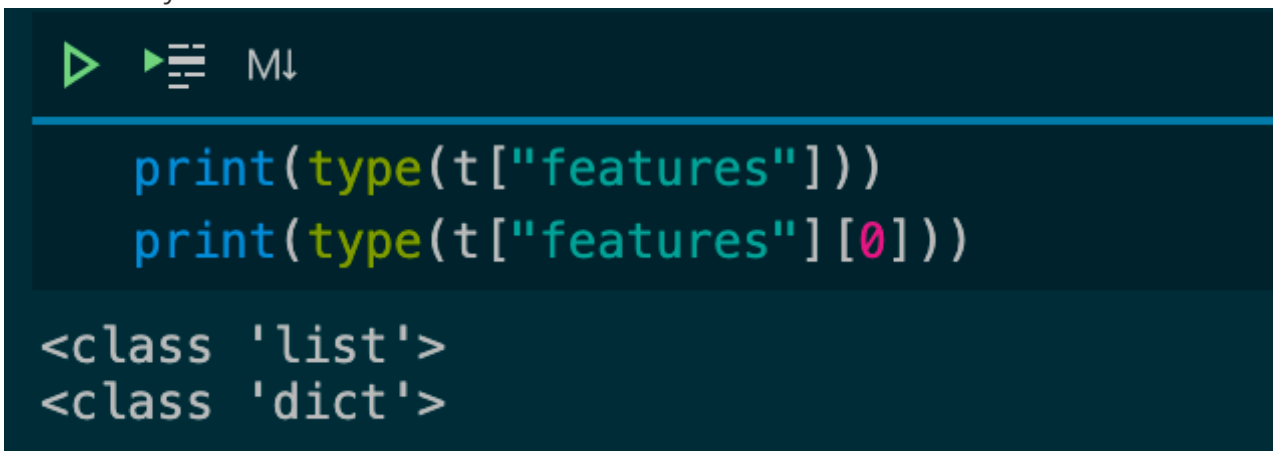

Fig.6 Type

The information are stored in a list of dictionaries. Here's an examplar of the data:

```
{'type': 'Feature',
 'geometry': {'type': 'Point',
              'coordinates': [121.33802, 31.16602]},
 'properties': {'flight_id': 'CSN3557-1607409360-schedule-0316:0',
                'prefix': 'CSN',
                'direction': 358,
                'type': 'A321',
                'ident': 'CSN3557',
                'icon': 'airliner',
                'ga': False,
                'landingTimes': {'estimated': '1607588305'},
                'origin': {'icao': 'ZGSZ',
                           'iata': 'SZX',
                           'isUSAirport': False},
                'destination': {'icao': 'ZSSS',
                                'iata': 'SHA',
                                'TZ': ':Asia/Shanghai',
                                'isUSAirport': False},
                'prominence': 118,
```

```
                'flightType': 'airline',
                'projected': 0, 'altitude': 2,
                'altitudeChange': 'D',
                'groundspeed': 125}
}
```

We can guess the meaning of the data based on the keys of the dictionary.

The `coordinates` (longitude and latitude), `direction` (heading), `altitude`, `ident`, `groundspeed` (flight number), `iata` of `origin` and `iata` of `destination` are informations we need.

`altitude` and `groundspeed` may not appear in some pieces of data.

However, the squawk number and the registration number are missing.

Note that the 'filight_id' is composed of flight number of the airline and a unix timestamp, so it should be distinct and can be used to get rid of the duplicated data.

We create a dictionary to store the data. The key is the 'flight_id' and the value is a list of dictionaries contains the selected data.

However when we implment the data-selection part, we find that one query per time may not enough.

Cross the Antimeridian

The website can only accept the query when minLat and minLon are smaller than maxLat and maxLon respectively (if not, the result would be the same as that case). So if the square area we ask has cross the antimeridian, we should seperate the query into two parts and merge the two query results.

Who's Flying over ShanghaiTech?

Finally, we could combine all the parts together and write a program to get airplane information around ShanghaiTech using the class we wrote. The program will crawl the information every 10 seconds and display five data crawled. By default, The data will be stored in `/tmp/data.json` on Linux. The program can also accept customer settings using command lines.

Usage

Move to the projectf folder.

For the first time use, you have to grant the program execute permission.

```
cd python-project/
sudo chmod a+x shtech.py
./shtech.py
```

You can use `-h` to check the help.

```
./shtech.py -h
```

Output:

```
orano_akia@kongyeqiuyadeMacBook-Pro python-project % ./shtech.py -h
-i <time>, --interval=<time>
  set an interval to crawl the website. The value would be set to 1 if the input time is less than 1. Default is 10.

-s <filename>, --saveto=<filename>
  Give a file to save the data, it should be a json file.Omitt the option indicates that the data will be stored in default file. (/tmp/data.json)

-n, --nosave
Ask the program not to save the data. Will override -s

-m <loops>, --maxloop=<loops>
  Give a number of loop times. If loops is less or equal to 0, then the program will loop infinite times.Default the times of looping is 5.

-d <number>, --display=<number>
  Give a number of items to display. If the number is less or equal to 0, then the data won't be diplayed. Default is 5 items
```

Another example of use:

```
./shtech.py -m 0 -d 2 -i 5 -n
```

This meanings that:

- The program won't stop until being halted by the user.
- Display 2 data each time
- Crawl(update) the data at least 2s intervals
- Don't save the data

References

[1] HTTP messages-HTTP|MDN

[2] HTTP request methods-HTTP|MDN

[3] Get-HTTP|MDN

[4] HTTP response status codes-HTTP|MDN

[5] HTTP状态码 | 菜鸟教程

[6] Uniform Resources Identifier - Wikipedia

[7] 快速上手 - Requests 2.18.1 文档

[8] 正则表达式手册

[9] HTTP 协议的 8 种请求类型

[10] HTTP Headers-HTTP | MDN

[11] 如何检查URL的合法性？ _ Goldrick的杂记-CSDN博客

[12] URL的语法-图灵社区

[13] 什么是URL？学习Web开发 | MDN

[14] re--正则表达式操作—Python 3.9.1文档

Data Source

URLs requested

`https://flightaware.com/ajax/vicinity_aircraft.rvt?&minLon=5.2150726318359375&minLat=48.1640625&maxLon=35.15625&maxLat=90&token=9f5a4f7273d920a6a7f12e1fbdf1c9e0102878d4`

The `https:` is the protocol or scheme, indicating which protocol should be used. Here we use HTTP protocol or the HyperText Transfer Protocol.

`//flightaware.com` is a host. It will be resolved into address which points to the target server.

`/ajax/vicinity_aircraft.rvt` is the path of target on the server. `.rvt` indicates it's a data file.

`?minLon=5.2150726318359375&minLat=48.1640625&maxLon=35.15625&maxLat=90&token=9f5a4f7273d920a6a7f12e1fbdf1c9e0102878d4` is the query componet. It contains a query string which can be understand by the server to indicate what we are asking.

The meaning of each parameters are:

Params	Meaning
minLon	Minimal Longitude
minLat	Minimal Latitude
maxLon	Maximal Longitude
maxLat	Maximal Latitude
token	token

Responses

The body of the responses are json files.

We can use `json()` method of the requests to convert the data into python dictionary class.

```
converted_responces = responces.json()
```

The following is an example of the HTTP requests:

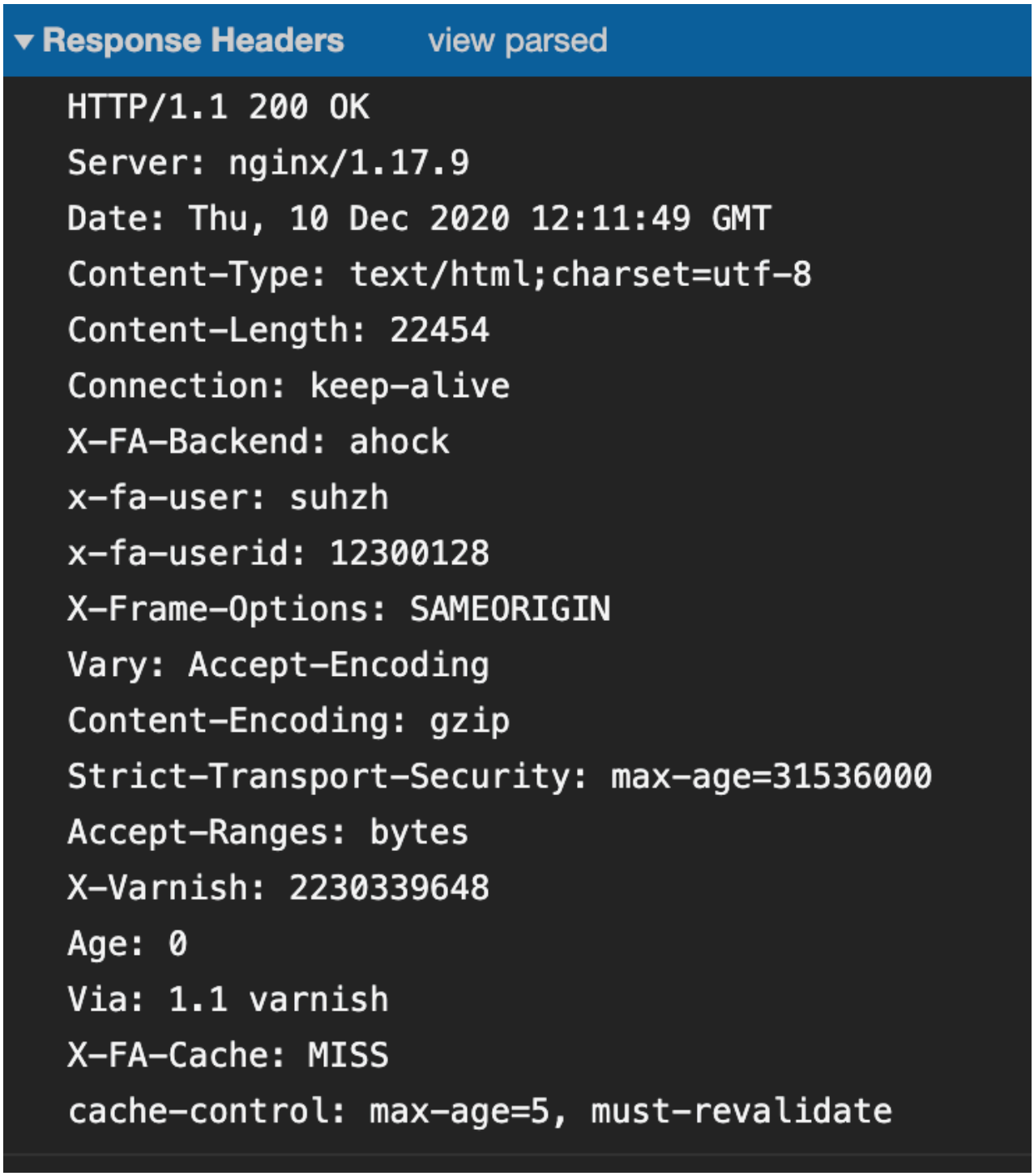

Fig.7 HTTP responce

The first line is the status line, and the rest are headers.

And following is a part of the respoonce body:

{"type":"FeatureCollection","features":[{"type":"Feature","geometry":{"type":"Point","coordinates":[5.46362,50.64927]},"properties":{"flight_id":"OOPMV-1607599990-adhoc-0","prefix":"OOP","direction":208,"type":"","ident":"OO-PMV","icon":"unknown","ga":true,"landingTimes":{"estimated":"0"},"origin":{"icao":"51.07N/5.11E","iata":null,"isUSAirport":false},"destination":{"icao":"---","iata":null,"TZ":"","isUSAirport":false},"prominence":12,"flightType":"ga","projected":0,"altitude":12,"altitudeChange":"D","groundspeed":156}},{"type":"Feature","geometry":{"type":"Point","coordinates":[15.81348,50.06898]},"properties":{"flight_id":"OKYUL50-1607601684-adhoc-0:0","prefix":"OKY","direction":44,"type":"","ident":"OKYUL50","icon":"cessna","ga":true,"landingTimes":{"estimated":"0"},"origin":{"icao":"50.06N/15.80E","iata":null,"isUSAirport":false},"destination":{"icao":"---","iata":null,"TZ":"","isUSAirport":false},"prominence":10,"flightType":"ga","projected":0,"altitude":10,"altitudeChange":"C","groundspeed":67}},{"type":"Feature","geometry":{"type":"Point","coordinates":[4.95689,52.30110]},"properties":{"flight_id":"CES7152-1607593414-eb-0001:6","prefix":"CES","direction":42,"type":"","ident":"CES7152","icon":"heavy_4e","ga":false,"landingTimes":{"estimated":"0"},"origin":{"icao":"EHAM","iata":"AMS","isUSAirport":false},"destination":{"icao":"---","iata":null,"TZ":"","isUSAirport":false},"prominence":105,"flightType":"airline","projected":0,"altitude":105,"altitudeChange":"C","groundspeed":307}},{"type":"Feature","geometry":{"type":"Point","coordinates":[6.42064,49.94357]},"properties":{"flight_id":"CSA2DZ-1607585747-7-5-233:2","prefix":"CSA","direction":247,"type":"AT72","ident":"CSA760","icon":"twin_large","ga":false,"landingTimes":{"estimated":"1607606580"},"origin":{"icao":"LKPR","iata":"PRG","isUSAirport":false},"destination":{"icao":"LFPG","iata":"CDG","TZ":":Europe/Paris","isUSAirport":false},"prominence":14400,"flightType":"airline","projected":0,"altitude":180,"altitudeChange":"-","groundspeed":265}},{"type":"Feature","geometry":{"type":"Point","coordinates":[16.39413,48.55510]},"properties":{"flight_id":"VPC12-1607595292-adhoc-0:0","prefix":"VPC","direction":56,"type":"G150","ident":"VPC12","icon":"jet_swept","ga":true,"landingTimes":{"estimated":"0"},"origin":{"icao":"LEBL","iata":"BCN","isUSAirport":false},"destination":{"icao":"---

Fig.8 Responce body

Most useful information are explained in the Display the Information. The titles of each data field are quite simple, so it is possible to understand the meaning of the data.

Implementation

Package used

- re - To use regex to find the tocken.
- time - To set intervals.
- requests - To send HTTP requests and parse the HTTP responces.
- json - To save the python dictionary data in json file.
- os - To refresh the terminal before each output
- platform - To check the user's system so that the file can be stored properly.
- sys - To get command line parameters
- getopt - To parse the command line parameters