Assignment Three: My API

Assumptions

Below is a list of assumptions about the assignment gathered from the PDF:

- Calls to the web service will be from the host running “services.py”
- Responses from the Marvel API will be stored as a text document

Implementation

Access to the Canvas and Marvel API’s and HTTP Authentication were implemented using Flask. Additionally, the time, hashlib, and json libraries were used to format both the requests and responses.

```
2  from flask import Flask, request
3  from flask_httpauth import HTTPBasicAuth
4  from werkzeug.security import generate_password_hash, check_password_hash
5  from ServicesKeys import *
6  import requests
7  import sys
8  import hashlib
9  import time
10 import json
11
```

Figure 1: Libraries used for the assingment.

HTTP Authentication was derived from the basic example given in the Flask documentation.

```
15 # authorized users to access flask
16 users = {
17     "admin": generate_password_hash("secret"),
18     "james": generate_password_hash("pi")
19 }
20
21 # authorized function from flask example
22 @auth.verify_password
23 def verify_password(username, password):
24     if username in users:
25         return check_password_hash(users.get(username), password)
26     print('>Could not verify your access level for that URL. You have to login with proper credentials')
27     return False
28
```

Figure 2: Basic HTTP Authentication implementation

Requests to the Canvas and Marvel APIs were formatted in much the same way.

```python
# Canvas Route
@app.route('/Canvas')
@auth.login_required
def getCanvas():
    # building url to send to canvas
    filename = request.args.get('file')
    reqstring = 'https://vt.instructure.com/api/v1/courses/%s/files/?search_term=%s&access_token=%s' % (104692, filename, canvastoken)
    r = requests.get(reqstring)
    # downloading file from canvas
    canjson = json.loads(r.text[1:-1])
    url = canjson['url']
    canobj = requests.get(url, allow_redirects=True).content
    f = open(filename, 'wb').write(canobj)
    return (r.text, r.status_code, r.headers.items())

# Marvel Route
@app.route('/Marvel')
@auth.login_required
def getMarvel():
    # building url to send to marvel
    storynum = request.args.get('story')
    ts = str(time.time())
    hash = hashlib.md5((ts + marvelprivkey + marvelpubkey).encode()).hexdigest()
    reqstring = 'http://gateway.marvel.com/v1/public/stories/{story}?apikey={apikey}&hash={hash}&ts={ts}'.format(story=storynum, apikey=marvelpubkey, hash=hash, ts=ts)
    r = requests.get(reqstring)
    # writing file to directory
    f = open("Story" + str(storynum) + ".txt", 'w+')
    f.write(r.text)
    return (r.text, r.status_code, r.headers.items())
```

Figure 3: Requesting and Receiving from the APIs

However, receiving from the Canvas API was different as a second request was made to download the found file from Canvas. Responses from the Marvel API were stored as a text file.

Results

The project was completed in its entirety and works according to the project specifications and the above assumptions.

Evidence of the project working properly on the Raspberry Pi is displayed in **Figures 4-6**.

```
pi@raspberrypi:~/RPiWebService/HW3_Heiman $ python3 services.py -p 7808
 * Serving Flask app "services" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production environment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:7808/ (Press CTRL+C to quit)
```

Figure 4: Initialization of services.py

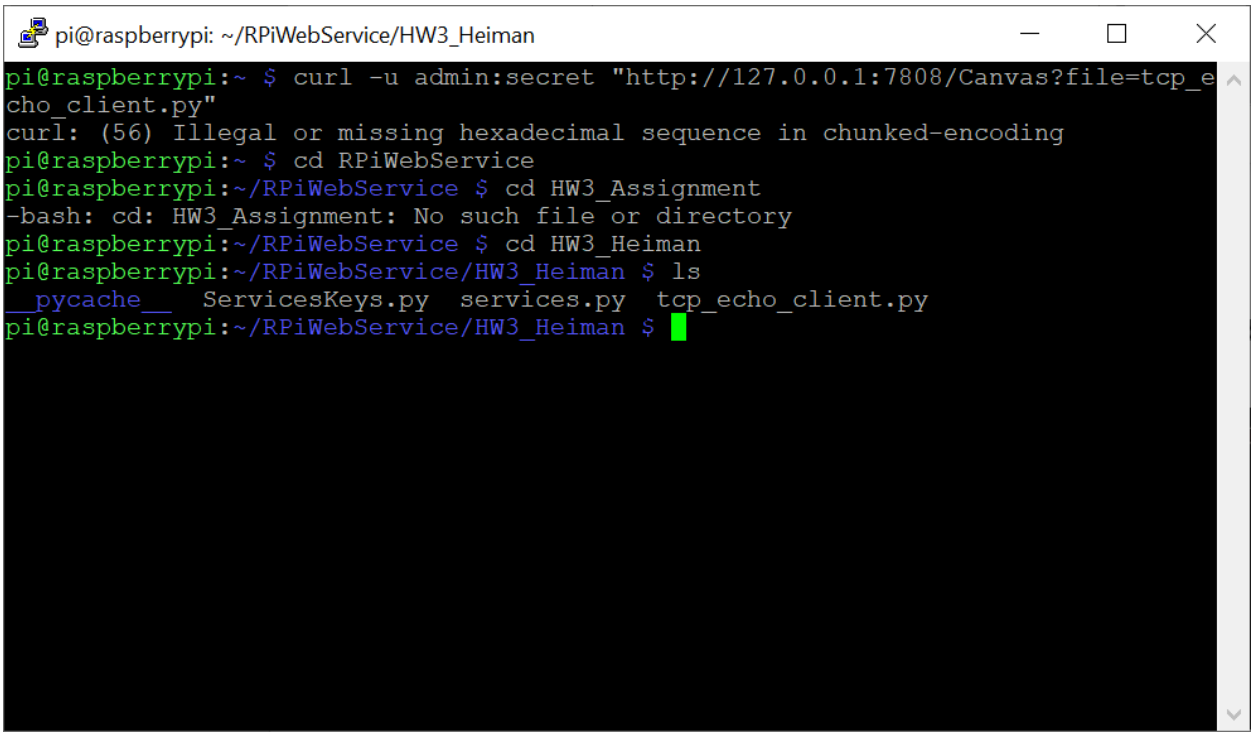

Figure 5:Using curl to connect to Canvas API and result

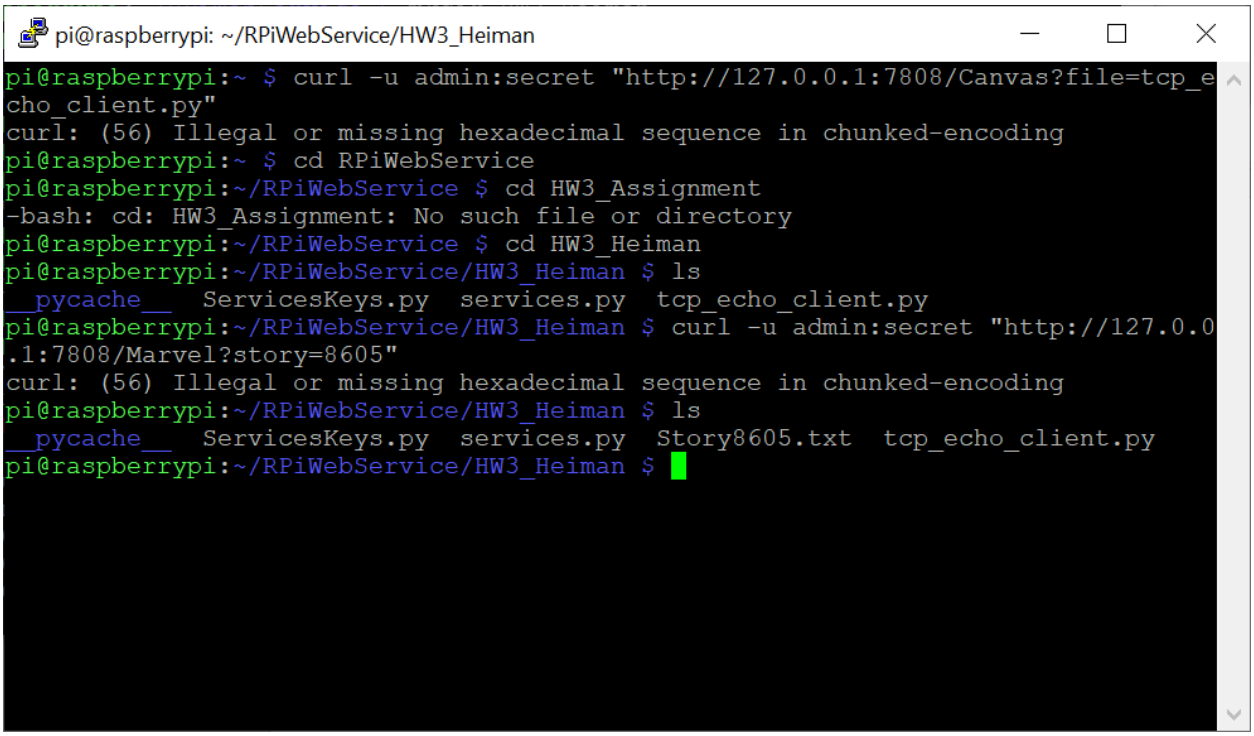

Figure 6: Using curl to connect to Marvel API and result

The file stored from the Marvel API can be seen below in **Figure 7.**

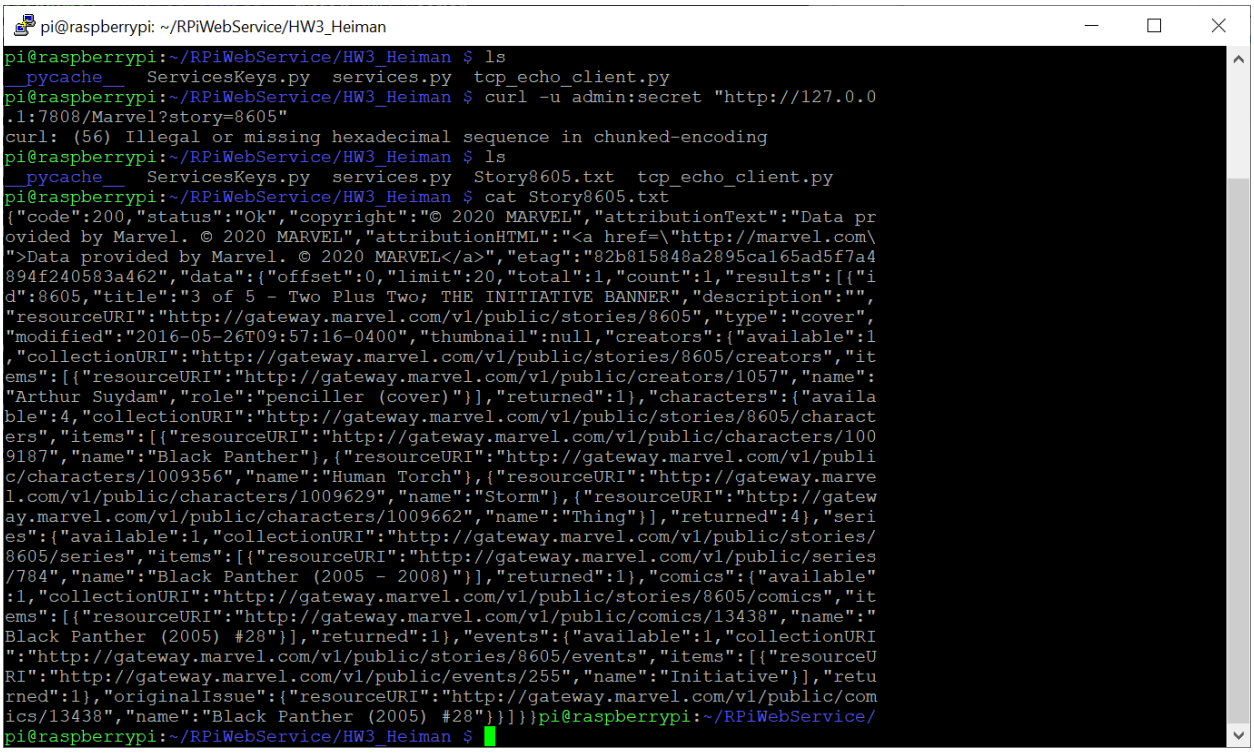

Figure 7: File downloaded from Marvel API

The file downloaded from canvas can be seen below in **Figure 8.**

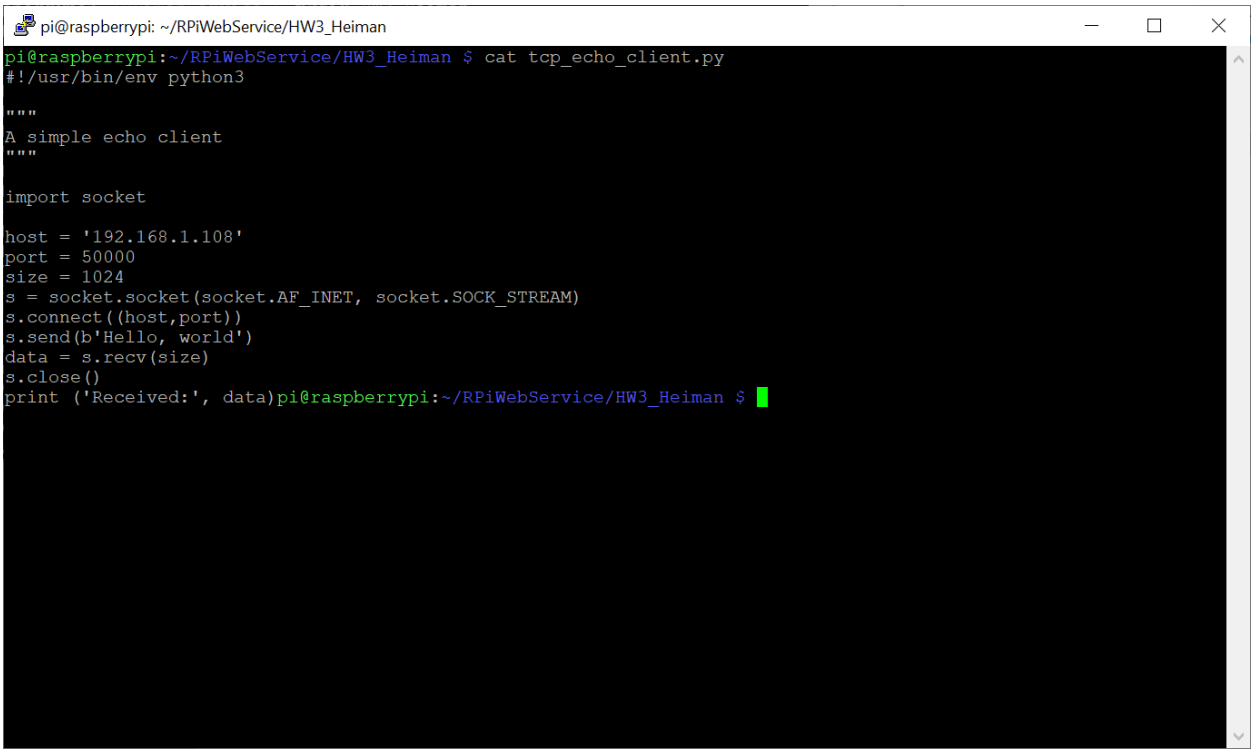

Figure 8: File downloaded from Canvas API