Dhairya Surana
4/19/2020

ECE 4564 Net Apps

Assignment 3

This project involved developing an application that would allow users to interact with the Canvas and Marvel websites. The project was implemented using the flask microframework and Python requests library. Since these technologies were new for me, I followed the tutorial on this website to setup the starter code: https://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask.

Authentication

One of the easiest parts of this project was the authentication.

```
################################### AUTHENTICATION ##########################################
@auth.get_password
def get_password(username):
    if username == 'admin':
        return 'secret'
    return None

@auth.error_handler
def unauthorized():
    return make_response("  Could not verify your access level for that URL. \n  You have to log

#############################################################################################
```

The code structure for the authentication was given in the tutorial. All I did was change the username and password to ‘admin’ and ‘secret’ respectively. Also, the unauthorized error message was modified to match the example given in the project spec:

```
curl -u bad_user:bad_pass http://0.0.0.0:8080/goodbye

>Could not verify your access level for that URL.  You have to login with
proper credentials

0.0.0.0 - - [08/Nov/2017 14:43:59] "GET /goodbye HTTP/1.1" 401 -
```

Dhairya Surana
4/19/2020

Canvas Interaction

The Canvas interaction was implemented by the following function:

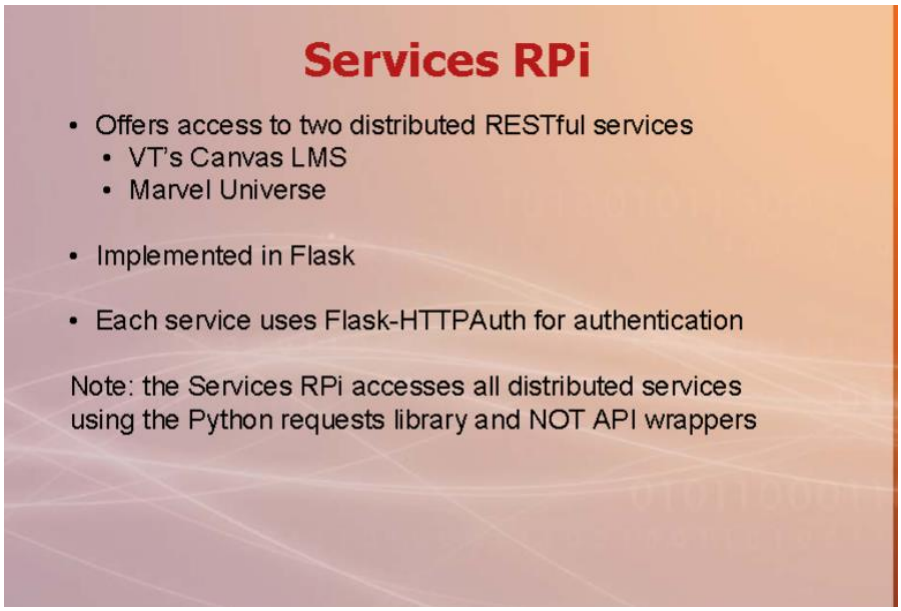
```
@app.route('/Canvas', methods=['GET'])
@auth.login_required
def getCanvasInfo():

    # Get file URL
    file_name = request.args.get("file")
    url = "https://vt.instructure.com/api/v1/courses/%s/files/?search_term=%s&access_token=%s" % (
    r = requests.get(url)

    # Download file
    file_info = r.json()[0]
    download = requests.get(file_info["url"])
    open(file_info["filename"], 'wb').write(download.content)

    return r.text
```

This function was implemented in 2 steps: getting the file url and downloading the file. Obtaining the file url was relatively easy compared to the other step. The example code was given in the project specification and all I had to do was copy and paste.

Figuring out how to download files, on the other hand, took a while. For a long time, I assumed that Canvas had its own API functions for downloading files. This assumption was caused by the following:

- 70 pts – Services RPi operates as specified in this document.
  - 15 pts - Use HTTP Basic Authentication.
  - 15 pts - Accepts REST requests from cURL or HTTP web browser.
  - 20 pts - Supports file download from Canvas using Canvas API.
  - 20 pts - Supports story download from the Marvel website using Marvel API.

I spent approximately an hour and a half trying to find ways to download files using the Canvas API. Later, I realized that the Canvas API consisted of just a bunch of get and post commands to the Canvas website. Also, when I read the specification again, I noticed the following:



The note at the bottom of the slide indicated that the requests library must be used to access the APIs. This was done by converting the requests object (r) to a dictionary via json(). Once I got the dictionary, I obtained the download url via the "url" field and recalled requests.get on the download url. Finally, I created a new file via the open function and wrote the content of the requests object to that file. Thus, I was able to download files by using the requests library to access the Canvas API.

Marvel Interaction

Once I figured out how to implement the Canvas interaction, doing the same for the Marvel interaction became a breeze. At first, I copied the function for the Canvas interaction and changed the app route to '/Marvel'.

```
@app.route('/Marvel', methods=['GET'])
@auth.login_required
def getMarvelInfo():

    # Get file URL
    story_name = request.args.get("story")
    url = "http://gateway.marvel.com/v1/public/stories/%s?apikey=%s&hash=%s&ts=%s" % (story_name,
    r = requests.get(url)

    open("Marvel.txt", 'wb').write(r.content) # Download file

    return r.text
```

Once that was done, I changed the code used for downloading files so that any Marvel story could be saved in a text file.

Goodbye

Like the authentication, implementing the goodbye functionality was one of the easiest parts of this project.

```
@app.route('/goodbye', methods=['GET'])
@auth.login_required
def goodbye():
    return "Goodbye World"
```

I'm not sure what the purpose of this function was. The only reason I implemented it was because the project spec had the following example:

```
curl -u admin:secret http://0.0.0.0:8080/goodbye

>Goodbye World

0.0.0.0 - - [08/Nov/2017 14:38:15] "GET /goodbye HTTP/1.1" 200 -
```

Conclusion

Overall, this project was very useful. As the internet has grown, the demand for web developers and engineers has increased. Learning about frameworks such as flask is necessary for future developers.