Lab Assignment 2: Interacting with a REST API

In this assignment we will play the role of a web service client trying to obtain information from the corresponding web service server.

This assignment is modeled as a game. You will need to start at the top level of the API, and follow the clues along the way and make appropriate HTTP requests to find the answers you see.

Your code will make use of a local web service whos code is hidden from you. You can only get it to reveal information by making HTTP requests and looking at the answers.

This service is provided in the file rest-game-unix.pyc[1]. After you download the file, you should run it with: `python3 rest-game.pyc`, and *keep it running*. This creates a local server, and you should see back a response from it that looks something like this:

```
* Serving Flask app "rest-game.cpython-37" (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://127.0.0.1:5000/ (Press CTRL+C to quit)
```

The address at the bottom is what you can use to make your requests. You should create a new terminal window to start another interactive Python session. From this point forward, one terminal window acts as a server and the other window acts as a client.

On your client, you should look at the content of assignment2.py[2] and add your own code to achieve the objectives described below. A start has been made for you in this file.

- The `requests` module has methods like `requests.get`, `requests.post`, `requests.options` etc for each of the basic verbs. These methods take as a first argument the URI of the resource in question, including the server address and protocol. For example the base resource `"/"` is obtained via the string `"http://127.0.0.1:5000/"` where that final slash is the resource indicator.
- Not all URIs respond to all methods. Use a `requests.options` call, and then access the `Allow` header of the result `r` with `r.headers['Allow']` to see the available methods.
- In order to include some JSON content to your request (for example when trying to create a new user), you can do so by adding `json=aDictionary` to the request call. Here `aDictionary` is the dictionary with the information you want to send. The `requests` module will make sure to turn that into a proper string representing a json object with the information from the dictionary, and it will also make sure to set the `Content-Type` header to `application/json`, a requirement for proper processing of the body of the message as JSON.

[1] https://github.com/skiadas/DataWranglingCourse/blob/gh-pages/assignments/rest-game-unix.pyc

[2] https://github.com/skiadas/DataWranglingCourse/blob/gh-pages/assignments/assignment2.py

The objectives

For each objective, add the lines of code one would need in order to achieve it in your assignment file.

Important: Some of the values used through this problem are randomly generated from the server. You must make sure your code uses the returned values as they are obtained from the server, and you should not attempt to copy-paste these values instead.

1. Your first objective will be to find out the one user that is already in the system. Examine the result you get from a basic GET request on the base resource, and it should suggest to you how to request a list of all the users. The response you get from requesting that list should include links for requesting the information of a specific user. At the end of this objective, you should have the first and last name of the one user that is already in the system. It is a well-known scientist.
2. Your second objective is to create a new user, by providing a first and last name. The names must be the first and last name of the actor who played the user you discovered in the first objective in a recent movie. You will need to be careful with the spelling of the name. The request you make should not be specifying a username for the new user you are creating, just a first and last name. It will be the server's responsibility to provide a username for you. You must provide the first and last name as a JSON object in the body of your request. If you have done things correctly, the response code for your request should be a "201". You should look up on the internet and book references to find out why a 201 and not a 200. Make sure to take a not of the value of the Location header, as it contains the access link for your newly created resource.
3. Your third objective is to send a "message" to the "scientist user" using your newly created "actor user". The information about your actor user's username should have been communicated in the Location header of the 201 request you obtained when you created this user in the previous objective. You should get this user's information, which would include a link to the user's messages. You should now be able to send a request to create a new message to the aforementioned recipient. The text of the message must be the name of the aforementioned movie. You may need some trial and error with the system until it gives you back the correct response. Looking at the feedback you get back from the service should hopefully direct you to the correct request. If you have done things correctly, you should see the message "Congratulations, you have won!" in the corresponding response's contents.

You should submit your completed Python file as an email attachment to me. The name of the file should include your first and last name, in addition to the assignment's number. It should contain no whitespaces.