Exercise 7: Reference Documentation

If this were a graded class, then this exercise would be your final exam. Fortunately, it's not! Documenting a POST and GET call will give you a chance to document pretty much all of the REST API documentation pieces that you'll need to know, so you'll be doing a POST first, followed by a GET.

Let's say that you were documenting an API for a new dating service for people who only care about how their dates sound, not how they look. (Yes, I'm making this up.) The company is called SoundDate, and here is the information you'll need to document the request for a user to upload a sound file to their own profile and to get information on sound files for another users's profile.

Uploading a Sound File to the current User's Profile

Fill out the template that's attached as a resource to this lecture. Here's the information you get from the developer team. Note that you don't need to specify the user's ID because it's for the current user, which had been determined through authorization.

1. The server address is https://api.sounddate.com
2. The resource is /profile/sound
3. The method is POST
4. There are three headers:
    a. Bearer has the access token. Required.
    b. Content-Type has the sound file format, which can be either audio/mpeg for mp3 files or audio/x-wav for wav files. Default is audio/mpeg.
    c. Accept has the response format, which can be application/xml or application/json. Default is JSON.
5. The POST body is the sound file
6. The sound file must be 5 minutes or shorter.
7. The response has only two elements:
    a. id: An integer, which is the ID of the new sound file
    b. length: A float, which is the length of the sound file, in seconds
8. Don't put in a Status and Errors table. We'll have one for the entire documentation set.

Hints:

- A "float" is short for "floating point", which means the decimal can "float" around the digits. In other words, it's a number that can have digits after the decimal point, unlike an integer.
- For the POST body, just say "The sound file".
- For the sample request, use "{sound file}" for the POST body.

Retrieving Sound File Information for a User

Fill out the template that's attached as a resource to this lecture. Note that this is information for another user, so you do need to specify the user's ID. Here's the information you get from the developer team:

1. The server address is https://api.sounddate.com
2. The resource is /user/{user id}/profile/sound.
3. The method is GET
4. There are two headers:
    a. Bearer has the access token. Required.
    b. Accept has the response format, which can be application/xml or application/json. Default is JSON.
5. There is one query parameter called sortOrder. This determines the order that the sound files are returned. It has four possible values:
    a. mostRecent, sorts most recent to earliest
    b. earliest, sorts earliest to most recent
    c. shortest, sorts shortest to longest
    d. longest, sorts longest to shortest
6. The sortOrder parameter is optional. The default is mostRecent.
7. Here's a sample response:

```
{
  "soundFiles": [
    {
       "id": 23456,
       "url": "https://api.sounddate.com/profile/sound/23456.mp3",
       "length": 11.2
    },
    {
       "id": 24559,
       "url": "https://api.sounddate.com/profile/sound/24559.mp3",
       "length": 19.8
    }
  ]
}
```

Status Codes and Errors

Add a status codes and errors table at the very end. Here are the status codes to document.

1. 200, success for both POST and GET
2. 401, access token is no good
3. 413, sound file that was POSTed was too long

Answers

You can see how I documented these requests at:
http://sdkbridge.com/ud/Exercise7Answers.pdf.