Project Report for GeneralAPIWrapper

The purpose of this project was to decrease the amount of time I spend making, executing, and unpackaging API calls. The last few personal projects of mine have been centered around APIs, and I couldn't shake the feeling that I had written very similar code before. I almost always needed an API key, and then would structure requests in a similar manner across APIs. I also have been working with the Steamworks API recently, so I wanted to implement a specific solution for that.

With those goals in mind, I made two use cases. The actor for both of them are developers who are using python to make API calls. In the first use case the developers are accessing one or more APIs. The scenario is that the developer makes a dictionary of get requests. They instantiate the APIWrapper class, and then store the dictionary in it. They are then able to execute a specific get request by giving the corresponding key. They are then returned the response body of the request.

In the second use case the developers are accessing the Steamworks API. The scenario is that the developer wants to access the achievements information about a specific game. They instantiate the SteamworksWrapper class, and then call a method passing in the game ID. They are then given the response object.

To meet the use cases I made two classes, the APIWrapper class and the SteamworksWrapper class. Both of these classes use the python requests package to execute API calls. The SteamworksWrapper class is a child of the APIWrapper class. The APIWrapper class has three dictionaries, one for get requests, one for post requests, and one for keys. The keys dictionary is for storing api keys, passwords, or values like profile IDs. There are then getters and setters for each of these dictionaries. There are two methods to execute API calls. The first is executeGet() and the second is executePost(). Both of these take in a key, pull the request from the corresponding dictionary, and then execute the request using the requests package. They then return the response object. The SteamworksWrapper class is a child class of the APIWrapper class. It has one filled-in method getGameAchievements() which takes in a game ID for a steam game, and returns a json object with the information.

One design decision I made was to have a generic wrapper class and a specific wrapper class for Steamworks. The reason for this decision was for the generic class to provide functionality for APIs I hadn't made specific functionality for. This class allows for someone to create an API request for any API, or multiple APIs if they choose, and store them in one or more instantiated APIWrapper classes. They are then able to execute these requests without having to recreate them each time or store them in a variable.

Another software design decision I made was to make the generic wrapper class a parent class to the Steamworks wrapper class. The reason I chose to do that was to both provide a base functionality for any specific API classes I made, and to allow

developers to store and execute requests that I hadn't implemented in the Steamworks class yet. The Steamworks API is extremely large and extensive, and I am extremely unlikely to ever implement every API call that can be made. By having the generic API class as a parent class, developers will be able to store and execute those requests.

The package I decided to compare my project to was the SteamworksPy project. This project is a full implementation of the steamworks API, and is an example of how I would probably structure my project if I intended to provide functionality for every API request. It doesn't have something like a parent generic class, but it structures the project around the different Steamworks interfaces. Under SteamworksPy/steamworks/interfaces/, there is a collection of python modules that each handle requests for a specific interface. For example the friends.py module handles all of the requests that deal with Steam friends. The reason why I like this project structure a lot more than the way I handled it gets into the extensibility of my software.

On a solo project level, it is easy for me to extend my code. When I have a request I want to execute but isn't currently in the Steamworks Wrapper class, I can just add another method pretty easily. As I add methods though the class will become increasingly unwieldy, and hard to read and bug fix. If I want to continue with this project, I think the ideal approach would be to split the Steamworks wrapper class into multiple classes where each of these subclasses deal with a specific Steamworks interface. This would make it easier to parse the project and make it much easier to update the project with new interfaces and requests.

There is a similar problem with the generic wrapper class. Since everything is handled in one class, as I added functionality for different types of requests, the class became much larger and harder to update. Currently it is fine since I am the only one using this project and I can extend it bit by bit, but I think it would be much more extensible if I split the generic API class into different sub classes based on the request type. Since different requests like get and post have different expected behaviors, it would be easier to deal with these differences if they were separate classes.