Criterion C: Development

The program's main purpose is to fetch data from other services and display them in a centralized application. In this case I used the APIs from the other programs to get that data. I used a external python library called "requests" to handle the API calls. Each service had a different method of getting the information I wanted. In my product, I had two services that I had programmed, Steam and League of Legends. For the Steam API, I make two calls to the steam servers. The first call is to get the user's friends list, then a second call to get the online status of the user's friends. League of legends has no way to get the friends list or the status of their friends, so I had to find a workaround. The workaround was to scrape the users game history to look for other players that the user plays with often, it then assumes them to be the user's friend. It also grabs a list from a file that the user can set. To then determine if they are online, the program checks if they are currently playing a game, or if their game was within the last few minutes. I designed the services to be easy to integrate into the program. Each service is defined in a python module, and the module just needs to have a function called get_all_friend_statuses(api_key, user_id), and have that function return a formatted list. The program reads these dynamically, so that the programmer just needs to create these plugin files and place them into the correct folder. This also allows users to customize what services they need, or expand upon the product as they see fit.

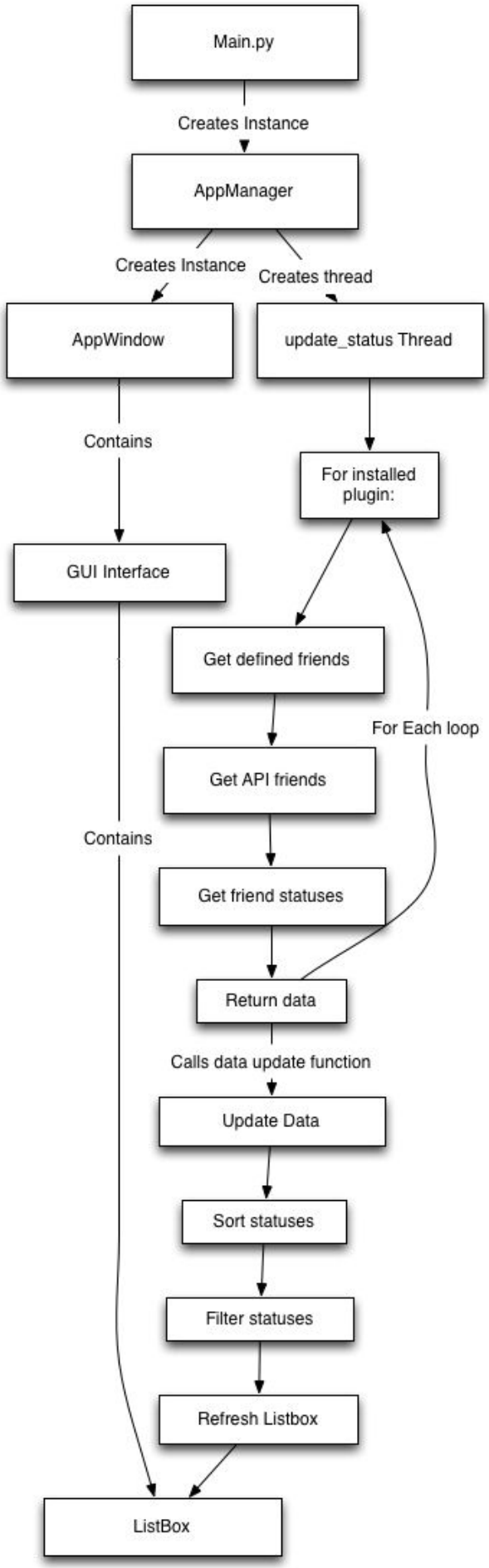
The API calls themselves caused trouble while I was programming. The initial design of my program had the APIs as blocking calls, the program could not continue while they were being made. These long API calls froze the GUI and led to a bad user experience. To resolve this I multithreaded the program. The GUI was in one computer thread, and the API calls would be made in another thread, so the blocking calls would not affect overall program responsiveness. This thread was created in a very simple way, I called a function into another thread, and the function contained an infinite while loop that made the API calls.

All these API calls requires a lot of prior data, such as the username of the user, and other predefined friends lists or data. In order to make the application as user friendly as possible, this data can be edited in three ways. The first is a small setup script that prompts the user for data then inputs the data into the appropriate files. This setup script is run the first time the program is run. The second way to edit the data is by directly modifying the files themselves. Most data is stored as plain text, and thus can be modified with simple and widespread programs. The third and final method to edit the data is with the options window in the program. This method will be what the user will use the most, and it fits with the design of the rest of the program.

The user statuses are output into an listbox format. From there the user can modify the sorting and filtering. The first sorting method is by service. From each service, it creates a section that shows the username and status. The user can sort it that way, or sort it by person. To sort by person, the user must first define a list of people, and the usernames they go by. This is stored in a file, and is parsed when the application goes to sort the data. Additionally, the user can filter out results by clicking on the logo buttons in the left column. This is done through an if statement in the sorting method. The sorting methods are stored in the tools module despite only being used by the graphics module. I sort them by category, then alphanumerically using python's built in function sorted().

The program was designed to be as modular as possible. There are 3 main parts to the source program. The "main.py" file, the library directory, and the plugins directory. The library directory contains two files, "graphics.py" and "tools.py". The plugins directory contains all the ".py" files that define the modules for all the service APIs. The program works when you call the "main.py" file. It sets up an instance of the AppManager, which in turn sets up an instance of the AppWindow (The GUI). The AppManager then starts up the thread that makes the API calls. The program determines what services to use by walking through the plugins directory, and it attempts to use each file in the folder as an API module. The plugins folder allows the user to customize what items they want to use. All other files, such as data files, images etc. are stored in another directory called "files". A flowchart of the logic order of the program is on the next page.

To create the program I used Sublime Text as my text editor, and I ran and tested the program with the OSX app terminal. I used a program called PyLint to lint my code before running and check for obvious errors. I used several of the built in Python Libraries, including an external one called "requests."


Main.py
Creates Instance
AppManager
Creates Instance
Creates thread
AppWindow
update_status Thread
Contains
For installed plugin:
GUI Interface
Get defined friends
For Each loop
Get API friends
Contains
Get friend statuses
Return data
Calls data update function
Update Data
Sort statuses
Filter statuses
Refresh Listbox
ListBox

Word count: 913 words

Bibliography

(n.d.). Retrieved February 14, 2016, from https://upload.wikimedia.org/wikipedia/commons/7/77/Gear_icon.svg

(n.d.). Retrieved February 17, 2016, from http://vignette4.wikia.nocookie.net/fallout/images/3/3b/Steam-Logo.png/revision/latest?cb=20130607061922

(n.d.). Retrieved February 19, 2016, from https://yt3.ggpht.com/-UHcRyzejL1U/AAAAAAAAAAI/AAAAAAAAAAA/ucobkN7xTco/s900-c-k-no/photo.jpg

Lundh, F. (n.d.). Tkinter Classes. Retrieved January 29, 2016, from http://effbot.org/tkinterbook/tkinter-classes.htm

Python 2.7.11 documentation. (n.d.). Retrieved January 28, 2016, from https://docs.python.org/2/

Reitz, K. (n.d.). Requests: HTTP for Humans. Retrieved February 12, 2016, from http://docs.python-requests.org/en/master/

Riot Games API. (n.d.). Retrieved February 10, 2016, from https://developer.riotgames.com/

Shipman, J. W. (n.d.). Tkinter 8.5 reference: A GUI for Python. Retrieved January 28, 2016, from http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/index.html

Steam Web API Documentation. (n.d.). Retrieved January 30, 2016, from https://steamcommunity.com/dev