* The program is run by executing the \_\_main\_\_.py file. This naming convention is used because if you try to run the containing docket\_alarm\_api\_bulk\_download folder directly with python, it will automatically execute \_\_main\_\_.py if present.
* Main.py contains logic to launch in the GUI if the isGUI variable in the config.py folder is manually changed to true. It is false by default, so the program runs in the command line interface.
* When using the command line, the \_\_main\_\_.py launcher will execute the run() function which executes the welcome() function in menus.py
* Menus.py is where all of the logic for the menu system is located. It prompts the user for choices and routes the execution where it needs to go based on the choices the user makes. The welcome() function starts up the program and displays the welcome screen. When the welcome menus are done executing, it runs the handle\_input() function which displays all of the selectable menu options to the user. This is a separate function form welcome() so the user can be directed back here from elsewhere in the program and not have to view all the welcome dialogue all over again.
* The welcome() function also checks if the user is logged in by looking to see if a file called credentials.pickle exists in the sav folder. If it does not, it executes the login\_interface() function in the login.py file to walk the user through menus built for logging in to Docket Alarm.
* All of the user authentication takes place in the login.py file. Login\_interface() displays an interface for logging in, and also verifies if the login was successful or not. If the login is successful, it calls the store\_user\_info\_locally() function. This function stores the successful login credentials in a file called credentials.pickle in the sav folder. The program checks for this file every time it is run to see if the user needs to re-enter their login information.
* The login.py file is also where the Credentials class is defined. This class looks for the credentials.pickle file when a Credentials object is initialized, and the username and password can be accessed with the username and password attributes. Calling the authenticate() method on an instance of the credentials class returns a login token, used for API calls.
* Within menus.py in the handle\_input() function, The user is asked if they want to get all JSON and PDF files, Get JSON files only, get PDF files only, search for dockets, or see more options. When selecting any of the options for downloading PDFs and/or JSON, the program will run the select\_paths\_menu() and the specify\_client\_matter() functions, which bring up menus for choosing the output paths for the downloaded files, and selecting the client matter that will be passed along with each API call made with the script.
* If the user selects that they want to only download JSON files, the program will direct them to the function for downloading json, which is thread\_download\_json() in the get\_json.py file.
* If the user selects that they only want to download PDF files, they will be directed to the function for downloading PDFs, which is thread\_download\_pdfs() found in the get\_pdfs.py file.
* If the user selects that they want to download PDFs and JSON files, the program will execute the get\_json\_and\_pdfs() function, which is defined in the menus.py file. This function merely runs the thread\_download\_json() function from the get\_json.py file, followed by the thread\_download\_pdfs() function() from the get\_pdfs.py file when it is done.
* Getting a json file with the get\_json.py file uses threading to download multiple files at once and cut down on total download speed. Table\_to\_list\_of\_tuples() takes the csv that the user wants to use as input and creates a list of tuples as the output. Each tuple in the list represents a set of arguments for the download\_json\_from\_list\_of\_tuples() function. If you were to take all the values in one of the tuples and pass them, in order, to the download\_json\_from\_list\_of\_tuples() function, it would download the json file specified. Instead of doing that however, we are using multiple threads. The thread\_download\_json() function acts as a wrapper for these 2 functions and is ultimately what is called directly in the program for downloading json files. Inside this function, table\_to\_list\_of\_tuples() is called and its results are stored in a variable, then, the executor.map function is called with the function we want to apply threading functionality to, download\_json\_from\_list\_of\_tuples, with the arguments we want to pass to it during each iteration as the next parameter, which is the variable that we stored the results of table\_to\_list\_of\_tuples() in.
* The get\_pdfs.py file works exactly the same way.