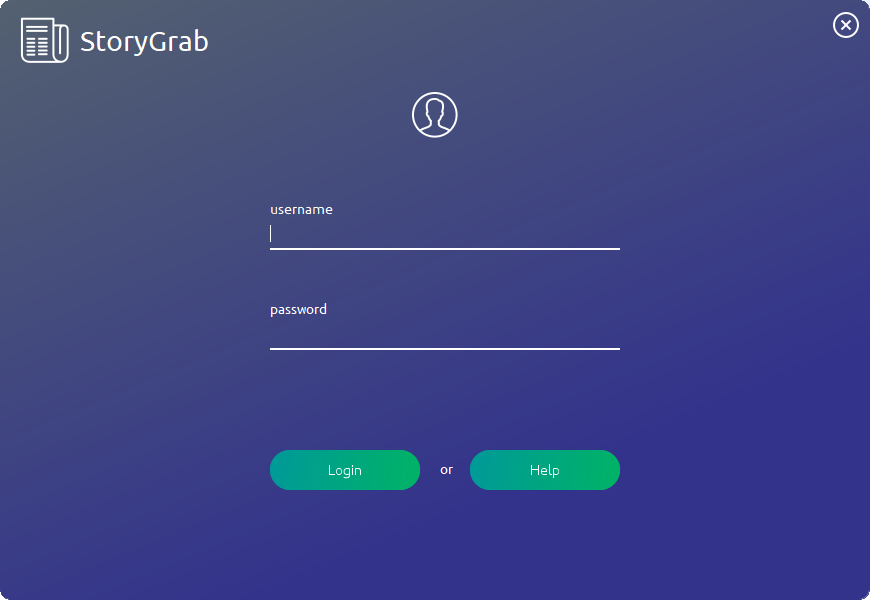
Criterion B: Solution Overview

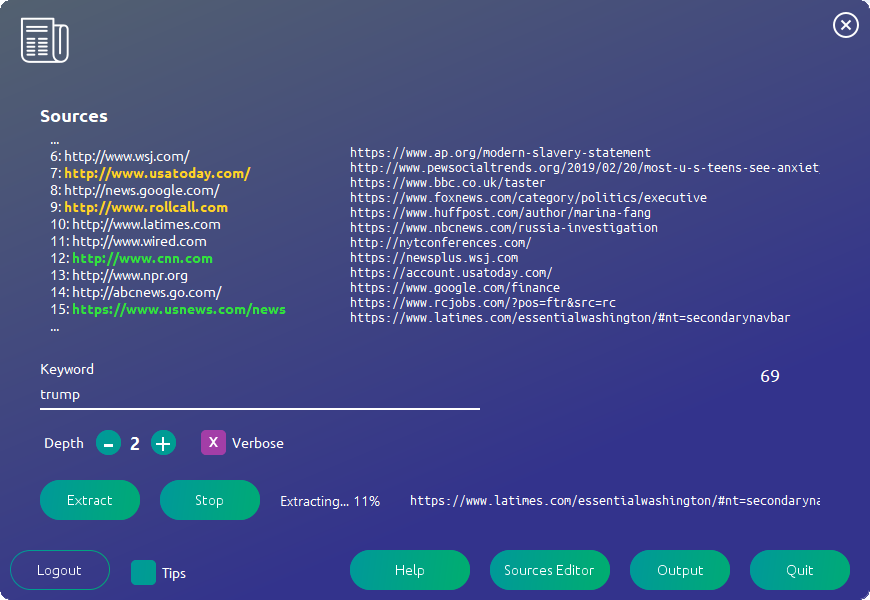
Designing a solution for a problem of this scope can be a very entertaining process. I decided to create it from the outside in – first starting with the GUI (Graphical User Interface), and then moving down into the “brain” of the program. I paid very close attention to detail and spent probably around 16 hours designing the GUI to look and feel very professional. After I had finished the bulk of the GUI, I transitioned to working on the search algorithm and interfacing that with each of the classes.

Object Design

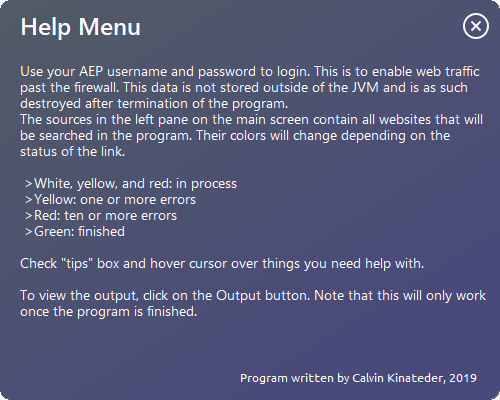
The program will have three forms: a login form (to use with the firewall), a main screen (to operate the extractors from), and a help menu. The color scheme for all three contains blue, grey, and green as the primary colors, and the font family is Ubuntu. The login screen is as follows:



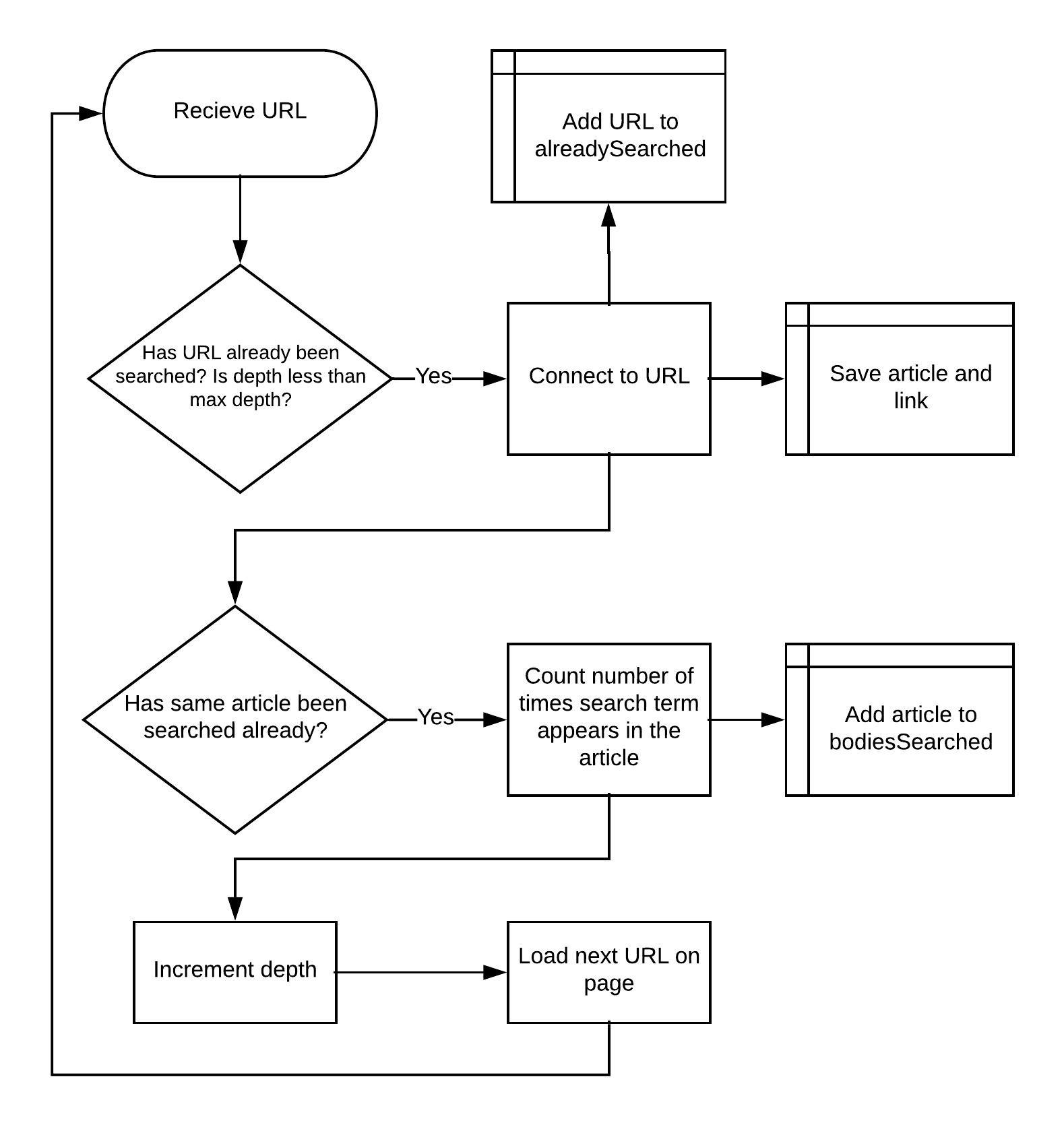
The interface for the main screen is designed for maximum information. The more the user knows about what is going on and how far the program has progressed, the better.

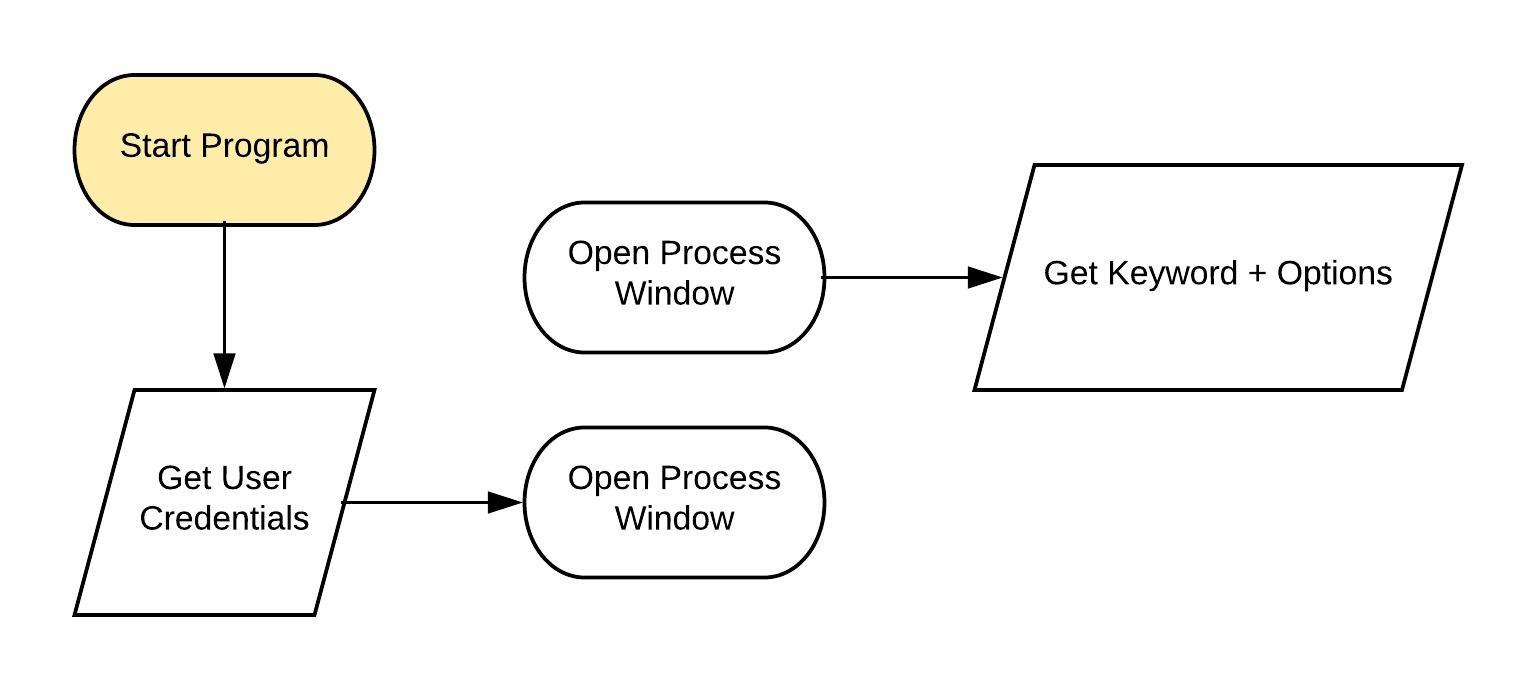
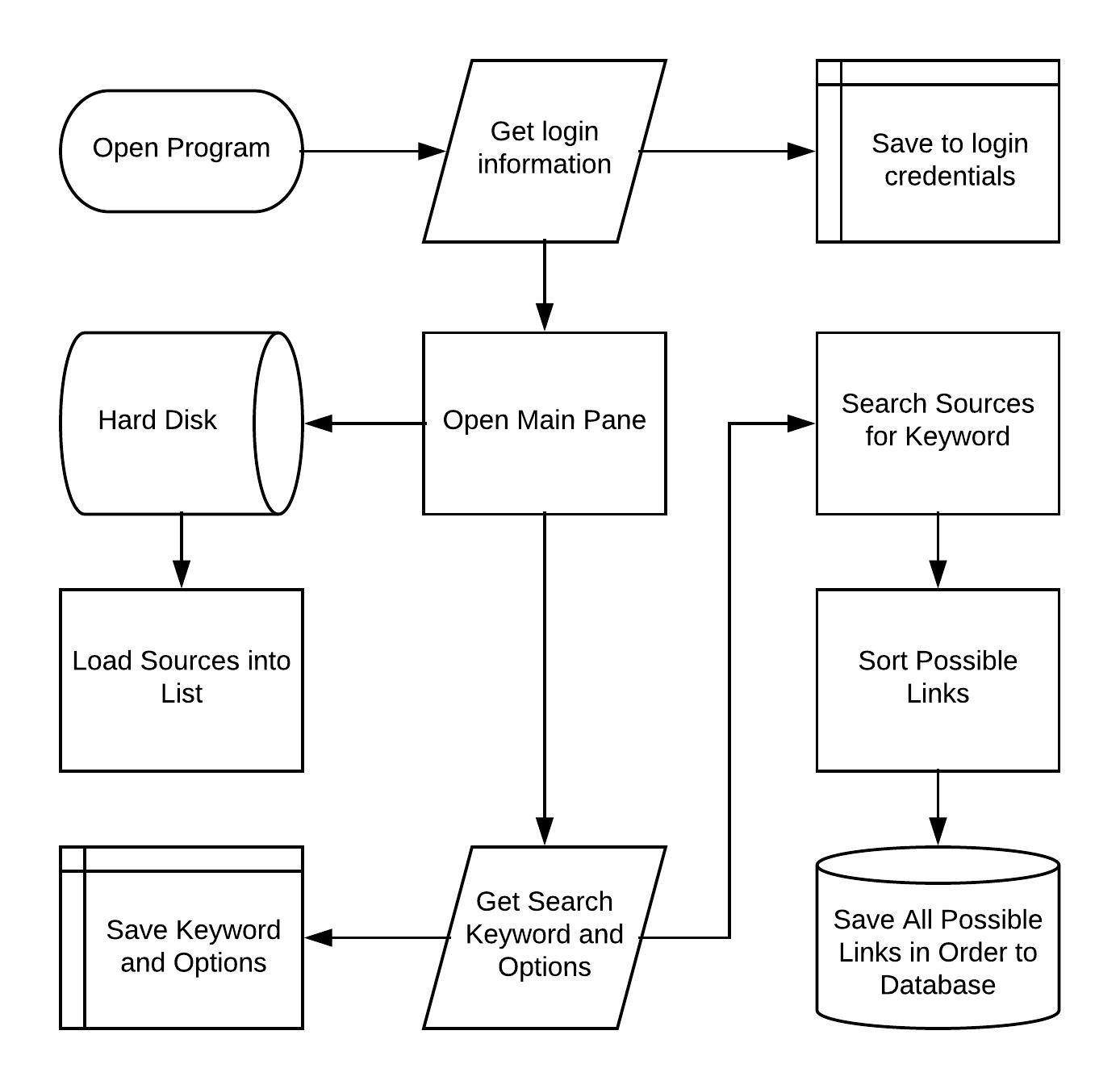


The help menu aims to provide clarity to any possibly vague instructions on the main program.

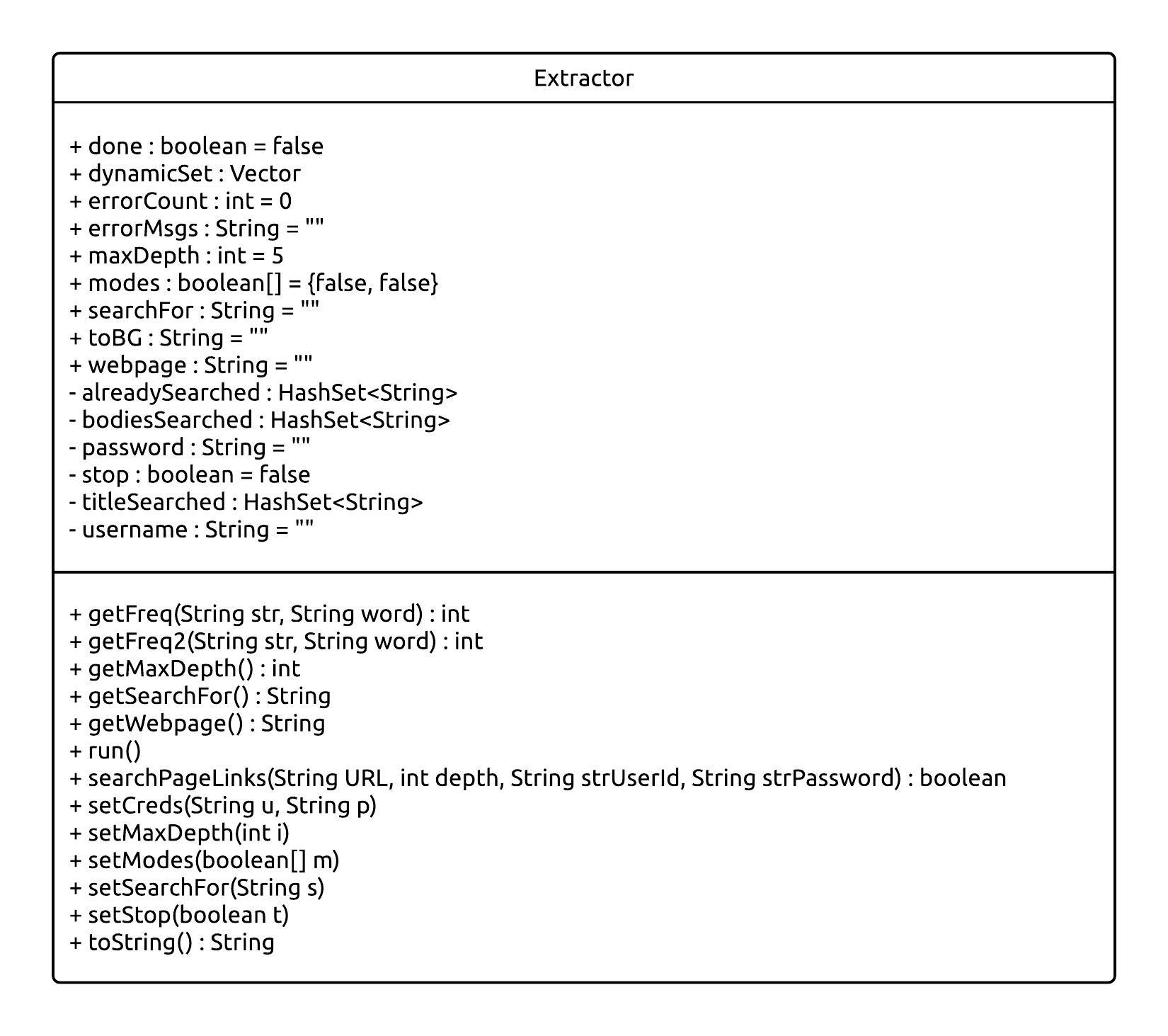


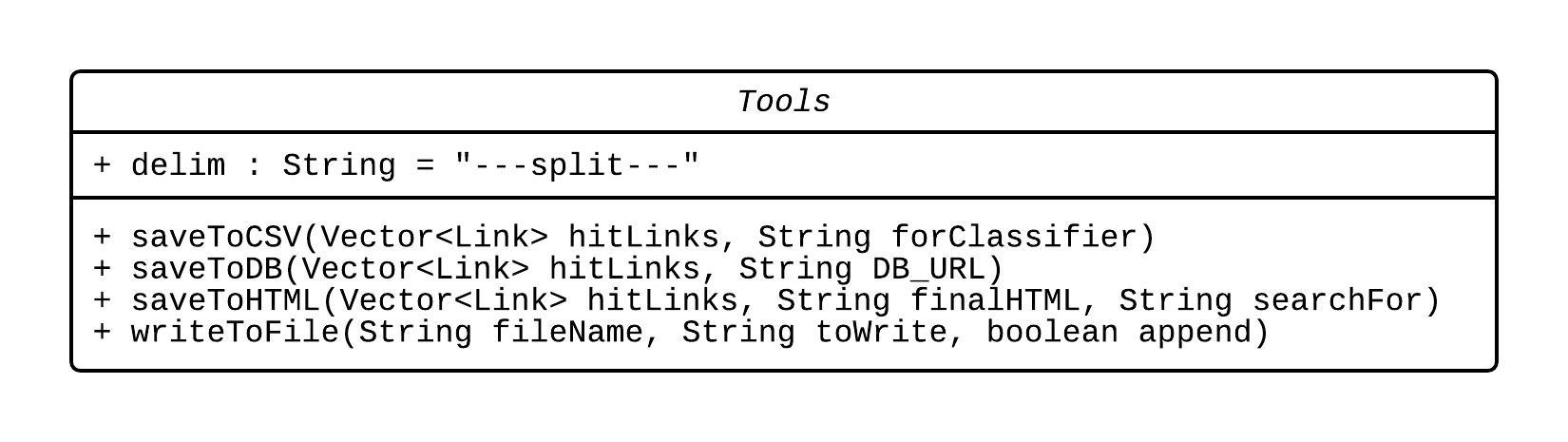
Flowcharts

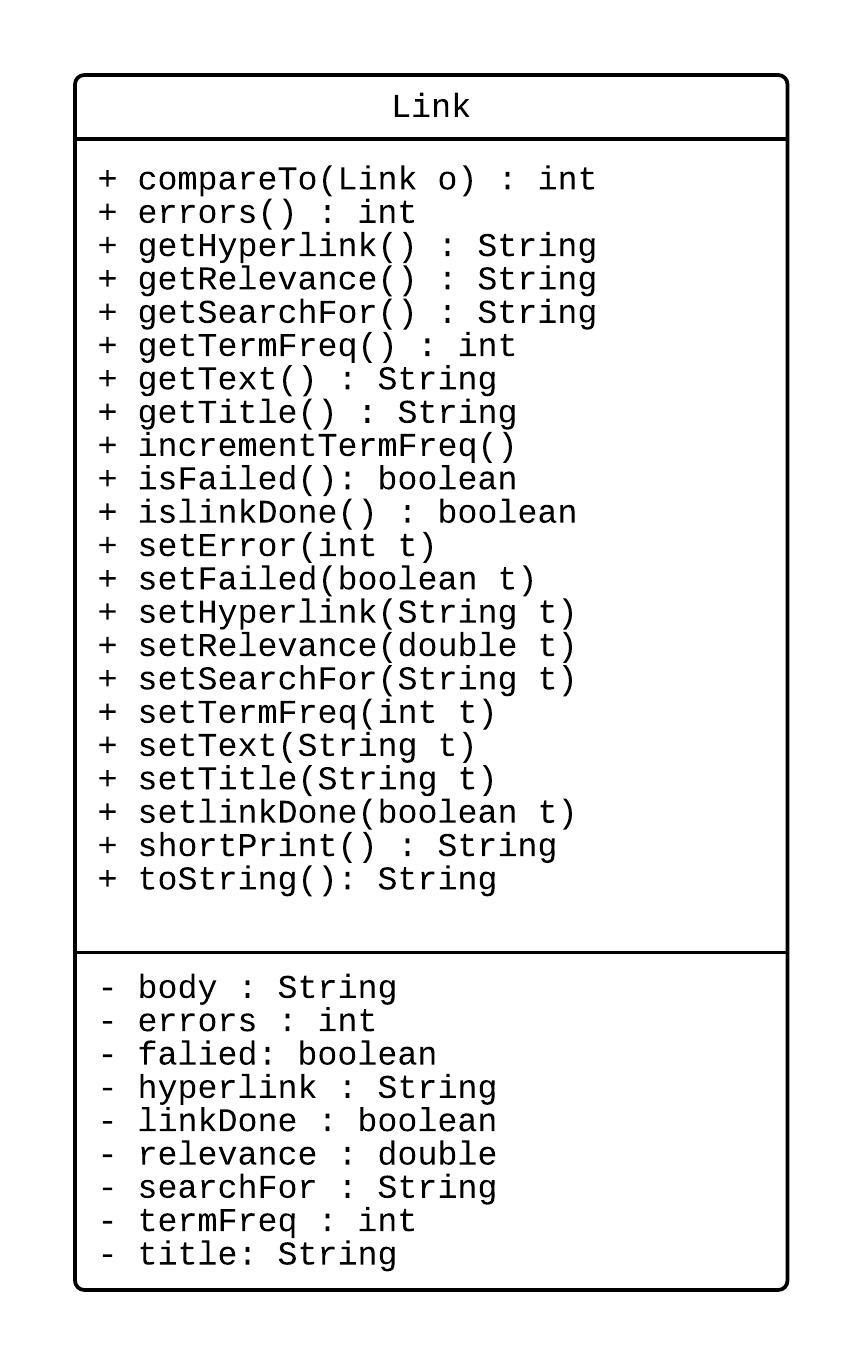
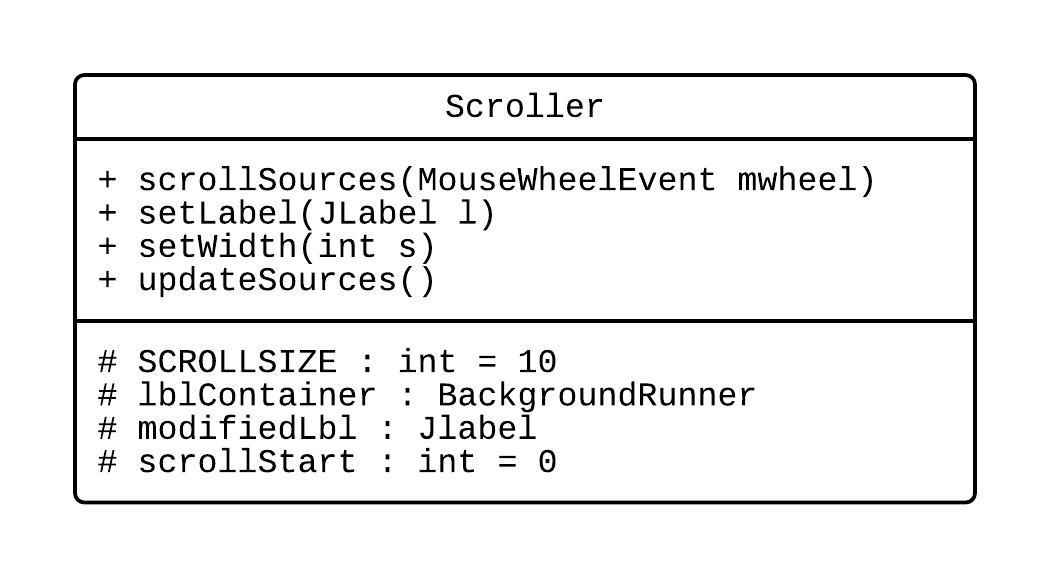
****



Class Diagrams

Using object-oriented programming, I will design this project with 9 classes. They will all interact with each other and make use of polymorphism, encapsulation, and inheritance. 





Class Overviews

There are 9 classes in StoryGrab, together totaling around 2700 lines of code.

**LoginWindow.java**

LoginWindow is the main class of the solution. It contains the code for the GUI as well as functions for the button actions. This class serves as the front-end and interface for the main algorithms.

**BackgroundRunner.java**

BackgroundRunner is the class that, using SwingWorker, allows the GUI to respond while the search algorithm is running. It is also responsible for controlling the multithreading and collecting all the data into a final list.

**Extractor.java**

The Extractor class contains the main searching algorithm. This method, which will be explained in more detail later, is a simple recursive bit of code.

**Scroller.java**

Scroller contains methods that work together to allow the user to control the lines that show up in a GUI panel with the mouse wheel. This is implemented in LoginWindow so that the user can see all the sources that are scanned and their corresponding statuses.

**AutomaticScrollUpdater.java**

To make the program more efficient, Scroller does not update every iteration of the main loop. Instead, it updates every 200 milliseconds. This class runs in the background and uses SwingWorker to be non-blocking.

**Tools.java**

Tools is the only abstract class in this project. It contains three functions: saveToCSV, saveToHTML, and saveToDB. They are used to save the output of the program to certain files to be viewed elsewhere.

**Link.java**

Link contains all the attributes of a Link object, such as title, hyperlink, and body. Implemented correctly, it is an easy way to collect all the data from one webpage and keep it all together.

**HelpMenu.java**

HelpMenu is a GUI form that gives the user help when they ask for it.

**User.java**

User collects the data of a user, just username and password, and stores it in an object.

Development Pseudocode

These give a general overview of what the most critical methods do.

***Login:***

*String username*

*String password*

*username and password =respective text fields on the login screen*

*switch to main screen*

***Start extraction (from LoginWindow):***

*if not running:*

*reset all variables*

*pass output labels, datasets to loader //loader is BackgroundRunner object*

*updateSources*

*start extraction in loader*

*endif*

***Start extraction (from BackgroundRunner):***

*while not done:*

*for every source:*

*if source isn’t searching yet:*

*start it searching*

*endif*

*publish output to screen*

*endfor*

*endwhile*

***Search webpage:***

*initialize authentication string to username+password*

*initialize error messages to a blank string*

*if link hasn’t already been searched and depth is less than max depth and no stop flags have been thrown*

*mark URL as already searched*

*connect to the URL and create a document*

*if search term appears more than once in the document*

*Count the number and add it to the list of matches*

*endif*

*for every link on page*

*re-run this method with depth incremented by one*

*endfor*

*endif*

Test Plan

These tests are here to ensure the solution meets the requirements

|  |  |
| --- | --- |
| Action to test | method of testing/expected result |
| Login screen is effective | Try to login to the firewall and access the internet from the program |
| Main screen is fluid and easy to use | Try to move the screen around and play with it to make things crash |
| All button clicks work | Test clicking each button |
| Extracting algorithm is accurate | Test searches with different depths and keywords and see if the output is reasonable |
| Scroll method works | Try scrolling in the source pane and make sure it works right |
| Help menu is helpful | Give the program to a new user and see if they can operate it without help |
| Source editor does not cause problems | Try to add a source from the source.txt file and make sure it does not fail to the backup |
| Output is readable and accurate | Run the program and make sure the output makes sense |
| Stop button saves correctly | Try terminating the program during execution and make sure the data is not corrupted in any way |
| Quit button does not crash but exits cleanly | Try terminating the program with the “quit” or “cancel” buttons |
| Colors in the scroll pane make sense | Run the program and watch the colors in the scroll pane and make sure nothing unexpected happens |