7. Software code submission with documentation (65%, due end of Week 15, Dec 09 2021)

Each team must submit the software code produced for the project along with a written documentation. The documentation should consist of the following elements:

1) An overview of the function of the code (i.e., what it does and what it can be used for).

The function of this code is to create a Google Chrome extension that can be used as an enhanced version of the basic ctrl-f find/search feature. This code takes a user's search term and matches it with portions of the document that contain not only that term, but other synonymous terms. It then presents the user with a list of the results within the context of the full sentence and highlights the portion of the page containing the found term. This enhances the basic search functionality by allowing users to

2) Documentation of how the software is implemented with sufficient detail so that others can have a basic understanding of your code for future extension or any further improvement.

The majority of our functionality is contained within the content.js and popup.js files. The popup.js file accesses the html elements defined in popup.html. This includes a text input for our search term, a button to initiate the search, and a text box to contain the results of our search. When the button is clicked, we send the information from the text input as a message to the content.js file.

In the content.js file, we listen for a message to be sent from popup.js with the information for the search. Then we send that search term into our findText() method as a parameter. From there we normalize our search term to lowercase for easier querying, and send it off to a thesaurus API. The results of the API can either contain an array of synonymous terms or it can return null. If the API does return data, we concatenate all the possible synonymous search terms into one large array named syns, and pass it to our helper() method. Note that regardless of synonyms being returned by the API, we still search for the original term provided by the user.

In helper(), we search through each HTML element of the current web page and match the terms in syns with the text contained in those HTML elements. We change the background color of the HTML element to red and store the sentence containing the search term in outputArray. Finally, now that the outputArray has been filled and the various HTML elements have been highlighted for the user, we create a string containing the number of search results and a list of all outputArray elements and store it in chrome.storage.local for use in popup.js.

After sending the message to content.js, popup.js enters a sleep() method that simulates a sleep statement in many other coding languages. This allows the chrome storage to successfully update the information within it before popup.js prints the results found within that storage. The code inserts the results string as HTML into our extension popup so users can view a bulleted list of the results and better understand how their term and its synonyms are used within the document.

3) Documentation of the usage of the software including either documentation of usages of APIs or detailed instructions on how to install and run a software, whichever is applicable.

https://www.dictionaryapi.com/products/api-collegiate-thesaurus

We use this API described in the link above to search for query synonyms. We just have to pass a query word in an api link with our api key in our link to gain access to an array of synonyms of our query. We also used the FETCH api to make the request towards the web server. The code of us using the api is in our content.js file in the findText function. We call our API every time we use our extensions button to activate it's functionality.

4) Brief description of contribution of each team member in case of a multi-person team.

Ben and Paul worked/met together for each stage of the project and did not divide the work systematically in a way to fully answer the question. However, Ben made contributions towards the implementation of the chrome extension while Paul helped make the api calls required for our synonym lexical analysis for our chrome extension. However both members were present when both major tasks were completed/worked on. Many hours were spent learning the basics of javascript and in particular the concepts required for programming a chrome extension. This process proved difficult and time consuming to implement basic functionality, so both Paul and Ben learned a lot and are very proud of the effort they put into this project.