ClipItPack

An extension of SimplyClip 4.0

Functional Documentation

Description:
ClipItPack, which is an extension of SimplyClip 4.0, is a Google Chrome extension designed for research students and power users. It provides a unified shared clipboard, allowing users to easily collect and store text snippets they find valuable for their research. They can summarize the text and even get citations for the selected text. The extension efficiently organizes and documents these copied items, making them readily available for future research tasks.

Project Structure:
Our project mainly constitutes of the following two parts:
1. SimplyClip Chrome Extension
2. SimplyClip Backend

SimplyClip Chrome Extension consists of the below :-
SimplyClip/
├── content.js
├── popup.html
├── popup.css
├── popup.js
├── seleniumTest.js

Below are the JavaScript functions :-
Function 1:- callSummarizer(text) :- This function calls the backend Django rest service using sync ajax call to get the summarized text.
Input :- Accepts string as input (Selected text for summarizer)
Output :- Returns the success message with the output string.

Function 2:- summaryEventListener(buttonClick, text) :- This function sets the summarized text as a chrome session variable to be used while creating Doc/Csv file.
This function calls the above defined callSummarizer() function
Input :- Accepts string as input (Selected copied Text)
Output :- void

Function 3:- mergeEventListener(buttonClick, list) :- This function merges the contents which you want to be arranged together as one in the Document file.
Input :- Accepts the list of items as input (Selected copied items).
Output :- provides the output as chrome variable set with the merged content.

Function 4:- sortContentEventListener(buttonClick, list) This function sorts the items using the listeners on the up and down arrow keys to arrange it as per the user's choice.
Input :- Accepts the list of items to be sorted (all copied items)
Output :- provides the output on the UI with the sorted items list.

Function 5:- darkModeEventListener(buttonClick, currentSession) This function toggles the User Interface based on the switch button i.e., if it is dark visual, it will be switched off to white, or vice-versa.
Input :- Accepts the input as the current UI session
Output :- provides the output on the UI with the view mode changed (dark or white mode)

Function 6- callCitation(text) :- This function calls the backend Django rest service using sync ajax call to get the citations for text.
Input :- Accepts string as input (Selected text for citation generation)
Output :- Returns the success message with the output string containing citations.

Function 7:- citationEventListener(buttonClick, text) :- This function sets the citation text as a chrome session variable to be used while creating Doc/Csv file.
Input :- Accepts string as input (Selected copied Text)
Output :- void

SimplyClip Backend is a Django rest project which consists of below structure :-

simplyclip_backend/

```
├── textanalysis/
    ├── apps.py
    ├── urls.py
    ├── views.py
   ├── summarizer.py
   ├── citation.py
   ├── citationTest1.py
    ├── tests/
        ├── test_url.py
```

Backend Function :-

generateSummary(input text) :-

This function uses pipeline which processes the input text in below stages
It performs some pre-processing steps like converting text into count-vectors
These count-vectors are given as input to train.
Then we train the model in the summarization pipeline and it and then we test the input-text on the trained model to get the summarized text as output.

generateSummary(input test):

The `generate_citation` function processes a given input (usually a DOI or URL) and generates citations in multiple styles (APA, Bibtex, Chicago Author-Date, Modern Language Association, Vancouver). It first checks if the input is too long (over 300

characters), marking it as a URL if so. If it's a valid DOI, it fetches and formats the citations in different styles. If any error occurs during the process, it raises a `CitationError`. The function returns a list containing the generated citations or an error message. It's a versatile tool for automatically generating citations from DOIs or URLs in various styles.

Test_Generate_Citation(text):

1. It uses the @patch decorator to mock the habanero.cn.content_negotiation function, ensuring that the real function isn't called during testing.
2. It has two main test cases:
    - The first test case simulates a successful citation generation for a valid input (a URL) and checks if the function correctly returns the expected citation formats in APA, Bibtex, Chicago Author-Date, Modern Language Association, and Vancouver styles.
    - The second test case checks how the function handles exceptions by setting the mock function to raise an exception, and it verifies that the function correctly raises a CitationError when an exception occurs.

Following are the test functions present in SeleniumTest.js:

These are all automated tests which serve to test the correct working of our chromium extension by setting launching Chrome with a user profile that has the extension setup and then running different tests to check the validity of its various functions.

Verify setup with Google Search

This automated test functionality launches a Google chrome tab to search for https://www.google.com. After searching for a specific keyword it visits the first link and launches the extension.

Check browser copy functionality

This test functionality, works to test the copy functionality, by copying the text in the first div tag present on the webpage and checking if it was copied correctly.

Check simply clip functionality

After storing the text of the first div tag of the webpage, the cntrl+c (for windows) or command+c (for mac) command is run and then this clipboard text is compared with what was copied via SimplyClip to assert its correctness.

Check dark mode functionality

On the webpage, the dark-mode button is searched for via its XPath and then the button is clicked. If the action is successful, then the extension can successfully switch to dark mode.

Check sorting functionality

The priority arrow down/ arrow up key is located within the extension via its XPath and then it is pressed to assert if the text has been actually sorted after the change.

Check Doc export functionality

This automated test functionality locates the export as doc button on the extension via the element’s XPath and then it clicks it to check if the file was successfully downloaded.

Check edit text functionality

Again, the element is searched for via the XPath and if the automated test is able to

locate the element and is successfully able to click it to edit the text, the test proves that the functionality is working as expected.

Check the text color functionality

To test the color changing functionality, we locate the color selection dropdown through its XPath, after we have successfully opened the dropdown via clicking it, we send it Keys that correspond to the various colors in it. If it is successfully able to select those colors based on the Keys as input, then it proves the correct functioning of this color functionality.

Check citation functionality

We again locate the element via its XPath then confirm if it is able to launch the citation functionality by clicking on the button. Thereby, proving that it is working as expected.

Check Merge functionality

This functionality is tested by locating the checkboxes provided with the copied texts and then successfully clicking them. Once this is done, the merge functionality button is searched for and clicked to see if the texts have been successfully merged as expected.