AN EFFICIENT CHROME EXTENSION FOR SIMPLIFIED TAB MANAGEMENT BY DOMAIN

Fan Lin[1], Garret Washburn[2]

[1]Lower Merion High School, 322 Parsons Avenue, Bala Cynwyd, PA 19004
[2]Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

The problem of managing multiple open tabs in browsers like Google Chrome is common, often leading to decreased productivity due to the difficulty in navigating and locating specific tabs [1]. This paper proposes a solution through the development of a Chrome extension called Tab Sorter, which organizes open tabs by domain, providing a simple and effective method for users to quickly find and manage their tabs. The extension avoids the complexity of AI-based solutions and focuses on user-friendliness and efficiency. Key components include a background service that listens for tab changes, a popup script that updates the user interface, and a clean, organized frontend. Experiments demonstrated the extension's efficiency in reducing the time needed to locate tabs and its minimal impact on system resources. Compared to other existing solutions, Tab Sorter offers a streamlined, no-cost alternative that significantly enhances tab management without overwhelming users with unnecessary features [2].

KEYWORDS

Chrome Extension, Tab Management, Browser Productivity, Domain Sorting, Tab Organization

1. INTRODUCTION

A very common problem that most people encounter with browsers like Google's Chrome, is that when one has quite a number of tabs open, it becomes increasingly difficult to navigate and find a desired tab. This issue can prove to be significant, as finding a specific tab in a mess of open tabs can take quite some time and counters productivity. This issue can negatively impact the people with a habit of having a lot of tabs open and that continuously forget where their tabs are. Most individuals, as the number of tabs increases, would confirm that it becomes increasingly harder to not only find specific tabs, but keep track of all the open tabs if it is necessary to be constantly switching back and forth between them. Most students would likely agree that in a class setting it is important to be able to quickly switch tabs when necessary, as it is important to be able to quickly navigate tabs in order to keep up with the instructor. Additionally, this extension would likely prove to be convenient in a meeting setting as it would enable any presenter or speaker to quickly sort through their tabs in an effective manner while maintaining the attention of the audience.

The Simple Tab Sorter organizes browser tabs manually, requiring user effort to group and title tabs. While it offers a degree of organization, it is time-consuming and relies heavily on user

David C. Wyld et al. (Eds): CCSIT, NLPCL, AISC, ITE, NCWMC, DaKM, BIGML, SIPP, SOEN, PDCTA – 2024
pp. 315-323, 2024. - CS & IT - CSCP 2024 DOI: 10.5121/csit.2024.141727

input. Our Tab Sorter improves on this by automating the process, reducing user effort, and enhancing productivity.

Tabber offers cross-browser synchronization and a feature-rich interface, but its free version limits users to 10 tab groups, and the full features require a paid subscription [3][4]. In contrast, our Tab Sorter provides unlimited tab sorting for free, making it accessible to all users without any financial commitment.

The Tab Sorter 2 extension offers multiple sorting options, but its overly complex interface can overwhelm users. Our Tab Sorter prioritizes simplicity and ease of use, presenting a clear and straightforward way to manage tabs, which helps users quickly find what they need without unnecessary confusion.

The solution proposed within this paper is a Chrome extension, the Tab Sorter, that creates and displays an organized list of the open tabs sorted by domain [5]. This Chrome extension serves as an efficient and convenient organizing tool, as it cleanly displays the sorted tabs and is right there in the browser. The extension organizes tabs by domain, which is an effective way of sorting as it enables the user to look through and find the specific domain they are trying to find. This extension, unlike most other sorter extension, remains simple yet effective as it is not necessary to overcomplicate the process of sorting tabs. The research conducted in this paper finds that overcomplicated tab sorters that utilize AI tend to be diminish productivity rates as they make the process of finding tabs more complicated than they need to be. Keeping the process simple and the display clear and organized enables the user to find exactly what they need consistently. Additionally, the extensions that attempt to utilize AI can be incorrect, which our web extension eliminates the possibility of as it only sorts the tabs by the domain they are on [10]. Another way in which the extension seeks to be more productivity enabling compared to other extensions is to be designed to be easy to use, including customized css that allows ease of use from all users.

Within this research paper, multiple experiments were performed to examine the functionality of the Tab Sort Chrome extension [6]. The experiments performed targeted not only the ease of use of the extension, but also how resource intensive the extension is. The first experiment, designed to test how functionally easy it is to use the extension, found that the extension is quite quick to use, as the average time it took for a user to find a random tab was around 8.5 seconds. This may not seem terribly fast, however, it is important to note that these times were collected at 20+ open tabs. The seconds experiment, designed to see how many resources were being extended for the extension, found that the average amount of memory extended was around 1 MB.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Tag Data is Not Saved

One major component of the Tab Sorter Chrome extension is the popup.js functions. The purpose of the popup.js file is to serve as a set of functions to be used to update the popup.html with the tab items that are stored in the local storage of Chrome [7]. One problem during development that we did have to address was the case scenario in which there was no tab data previously saved in the chrome local storage, in which we handled by establishing the data in local storage beforehand consistently so that we wouldn't have to worry about it. Most of the other problems we faced during the development of the popup.js files were addressed in the other components, however, the popup.js set of functions is nonetheless an important piece of the project.

2.2. How to Parse and Organize Domains

The background.js file is the service worker running in the background of Chrome constantly, and listens for tab changes so that it might be able to update the list of necessary tab data in Chrome's local storage. A major issue that we had to address during development was definitely how to parse and organize the domains. What we settled upon doing to neatly organize these was to develop a function to parse and extract the domain, and then organize the tabs that at a url that matches the domain are stored in an array that is mapped to the domain name. All of the domain names are stored in one big map as keys, and are mapped to their respective arrays of tabs. This domain name map is stored in Chrome's local storage, and accessed when it comes time to update or display the tabs in the popup window [9].

2.3. Mistakes

The popup.html and styles.css files are the components in the Chrome extension that act as the frontend, in which they create and stylize the actual extension's popup in Chrome. The most prominent issues encountered during the frontend designing phase of the development were typically bugs in either the popup.html, or more usually, the styles.css. Formatting and designing the components to be just right in styles.css proved to be quite a task, but eventually we found a look that fit nicely. Additionally, next to fixing any bugs, was deciding on the actual frontend design. It was important to us, due to the nature of this app being productive, that the design promotes a smooth experience for the user. Thus, we avoided needlessly complex designs and went for a more streamlined feel.

3. SOLUTION

The three major components involved in making up this program are the the background.js service, popup.js functions, and the frontend popup.html and styles.css interface files. Upon opening up the extension by clicking it open in chrome, the user is greeted by the frontend designed as defined in the popup.html and styles.css files. The popup.html links the popup.js as script, implying that as the popup.html is opened and loaded it will run the popup.js. In the popup.js file, a few functions are defined. The two major functionalities defined in popup.js are to grab the tab data from Chrome's local storage and update the popup with that data. The grabbing and organization of the actual tab data from Chrome, however, is done entirely in background.js. In the background.js there are listeners constantly listening for tab change events, and when it hears a tab change event it will parse and organize all of the tabs by their current open domain. Once the tabs are organized, background.js stores this sorted list of tabs by domain in Chrome's local storage so that it may be accessed by popup.js and displayed. The most prominent technology used for this extension is the Chrome Extension suite, as Chrome itself is what the extension is built into and utilizes to run the program [8].

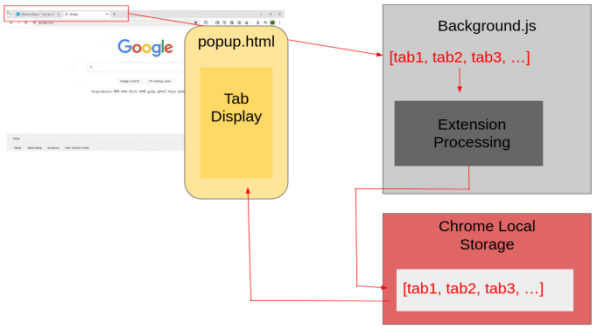

Figure 1. Overview of the solution

The first major component, the background,js service, is essentially the backbone of the extension in that it is constantly updating and organizing the local storage with the list of the tab's domains. During development, good use of the Chrome storage and tabs objects were employed to interact with Chrome's local storage and tab items for the app's functionality [14].

```
function extractDomain(url){
    let domain;

    if(url.indexOf("://") > -1){
        domain = url.split('/')[2];
    } else {
        domain = url.split('/')[0];
    }

    return domain;
}

// function to update local storage after a tab event
function handleTabChange(){
    let example_url = "youtube.com";
    // grab the list of tabs from the browser
    chrome.tabs.query({}, function(tabs) { // create a query and get the current tabs open (a query is basically a question)

        let tabData = {}; // {"youtube.com": [tab1, tab2], "slides.google.com": [tab3], "reddit.com": [tab4]}

        tabs.forEach(function(tab) {
            let domain = extractDomain(tab.url);

            if(tabData[domain]){
                tabData[domain].push(tab);
            }else{
                tabData[domain] = [tab];
            }
        });
        // store tabData inside of the local Chrome storage
        chrome.storage.local.set({"tabData": tabData});
    });

    // store the list into local storage
}

// create listeners for tab changes
chrome.tabs.onUpdated.addListener(handleTabChange); // listening for an update inside of the tabs inside of chrome
chrome.tabs.onRemoved.addListener(handleTabChange); // listening for a tab to be removed inside of the tabs inside of chrome
chrome.tabs.onCreated.addListener(handleTabChange); // listening for a tab to be created
```

Figure 2. Screenshot of code 1

The background.js background service serves the purpose of listening for tab changes, updates, new tab creations, or tab closures and updating the list in storage of open tabs and what domain they are currently on. The background.js file starts with the Chrome tab listeners, in which it is listening for tab changes or events. Once a tab change is heard, the handle Tab Change function is called and is intended to update the list in storage of the list of open tabs. In the handle Tab Change function, the very first action that is performed is the querying of the Chrome tabs in order to receive the list of the current open tabs. After the querying of the tabs, the tabs are

iterated through in an attempt to see what domain each tab currently is on. Each individual tab has its url passed to a predefined function as seen in the top of the image above, the 'extract Domain' function, in which the domain is parsed from the url and returned back to the 'for Each' loop. Once the domain is extracted, the tab Data map that is initialized prior to the 'for Each' loop is checked to see if the domain as a key is already present in the map. If the domain is present in tab Data, the tab is added to the array that the domain is mapped to; if not, a new key is created with the domain and the tab is stored in an array that is mapped to the new domain key. Finally, once the tab Data map is complete and all tabs have been iterated through, the complete tab Data map is stored in Chrome's local storage, so that it may be accessed by the popup.js and displayed in the popup.

The next major component, the popup.js, serves the purpose of acting as the service worker for the popup.html. The popup.js grabs the open tab data from Chrome's local storage and updates the list of open tabs in the popup by adding html to the list of tabs for the user to see.

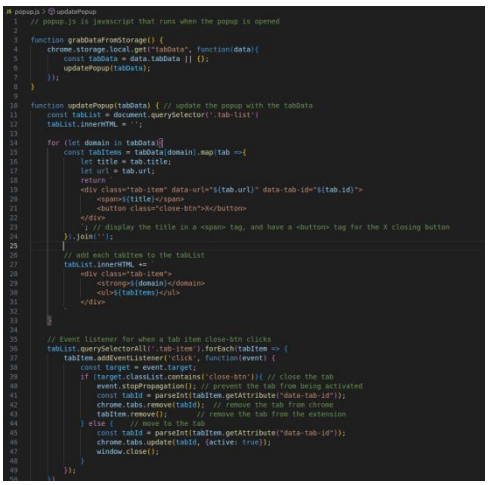

Figure 3. Screenshot of code 2

Depicted above is an image of the main functionalities in the popup.js file. The first function, the grabDataFromStorage function, serves the purpose of simply grabbing the locally stored tadData from Chrome's local storage [15]. This data is pulled out of storage as a json array, and eventually is parsed in the next function. The next function depicted is the updatePopup function. The updatePopup function performs the actual updating of the popup.html itself with the list of tabData, and dynamically builds the list of domains and the tabs underneath for each item in the tabData. Additionally, the updatePopup function also builds the eventListener for each tab item in the list which is listening for any click event within the tab item. If the click is inside of the close button marked with an 'X,' then the tab is closed and the local storage is updated. However, if the user is simply clicking on the tab item and not on the close button, the tab the user clicked on becomes the active tab and gets opened.

The third and final major components that comprise the TabSorter are the popup.html and styles.css files respectively. The popup.html and styles.css serve the purpose of essentially structuring what the frontend will look like for the user. The popup.html creates the structure, whereas the styles.css establishes the color and design of items.

```
# styles.css > .title
body {
    font-family: Arial, Helvetica, sans-serif;
    background-color: rgb(255, 255, 132);
}

.popup-container {
    width: 300px;
}

.title {
    font-size: 20px;
    color: rgb(0, 0, 0);
}

.tab-list {
    padding: 5px;
}

.tab-item {
    margin-bottom: 10px;
    padding: 10px;
    background-color: rgb(230, 230, 72);
    border-radius: 5px;
    cursor: pointer;
}

.close-btn {
    background-color: rgb(162, 138, 0);
    color: rgb(0, 0, 0);
    border: none;
    width: 20px;
    height: 20px;
    font-size: 14px;
    text-align: center;
    cursor: pointer;
}
```

```
popup.html > html > body > div.popup-container > div.tab-list
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="styles.css">
    <title>Really Cool Tab Sorter</title>
</head>
<body>
    <div class="popup-container">
        <h1 class="title">Cool Tab Sorter</h1>
        <div class="tab-list">

        </div>
    </div>
    <script src="popup.js" defer></script>
</body>
</html>
```

Figure 4. Screenshot of code 3

Within the popup.html and styles.css lays the structure of the popup that is visible when the user clicks open the extension within Chrome. The popup.html defines the structure of the popup, and inside you can see that it is actually rather simple. Within the body of the html all that is present is a h1 tag to create a header and a div to create the area in which the popup.js will establish the list of open tabs. Additionally, at the bottom of the popup.html is a script tag that references the popup.js, so that the tab list update code may be run and will be visible in the tab. Next, the styles.css defines the unique style that we planned for the popup. When designing the popup, the goal was to give the extension a notebook or sticky note feel to give the user a sense of productivity when using the extension. As such, the styles.css gives items, like the body and the tab-item classes the colors and fonts necessary to look similar to a yellow paper notebook.

4. EXPERIMENT

4.1. Experiment 1

One potential blind spot of the Tab Sorter Chrome extension is the repercussions of having a copious amount of tabs open. Specifically, we are concerned if the user has too many tabs open on the same or different domains that the extension will unintentionally make navigating the tabs more difficult for the user.

To test to see whether the tabs are sorted in an effective fashion, we will perform an experiment to see if the user is able to effectively navigate the extension to find a specific tab effectively. To perform this experiment, we will create a controlled browser environment, and introduce ten new tabs for each attempt. One team member will call out a random browser tab and start a timer, while the other team member looks for the specific tab. Once the second team member finds the tab and navigates to it, the timer will be stopped and the time recorded. This experiment will demonstrate in performance time how quickly a user can navigate through the extension and therefore prove it's efficiency.

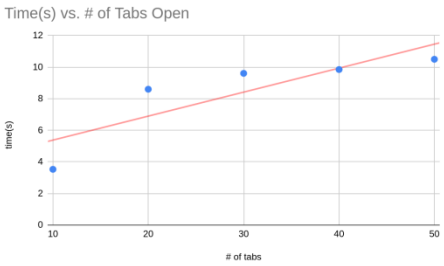

Figure 5. Figure of experiment 1

After performing the experiment, we found that depending on the number of tabs open it takes, on average, around 8.418 seconds for a user to find and open a tab if the number of tabs open ranges anywhere from 10 to 50 tabs. The quickest time it took to find a specific tab, as expected, was the attempt with the lowest amount of tabs opened, 10, and took around 3.53 seconds. Conversely, the longest attempt was the 50 tab attempt and took around 10.5 seconds. Surprisingly, after around 10 tabs, the time it took for the team member hunting for specific tabs was generally quite static and almost seemed to plateau at around 10 seconds. To follow up on why the research team believes this plateau occurred, it is likely due to the fact that in the range of 20-50 tabs, the amount of differing domains seemed to be the same. Therefore, we believe that the average time it took to find a specific tab in that range is because it took the same amount of time to find the grouped tabs by domain.

4.2. Experiment 2

Another potential point of concern within the Chrome extension is how much memory the Tab Sorter may take up from the Chrome browser and further the operating system when an extensive number of tabs are opened.

To test to see whether the extension is quite resource intensive, we will design an experiment that is quite similar to the previously designed experiment. To conduct this experiment, we will open a set number of tabs, starting from ten and moving our way up to fifty by tens, and will check the current memory usage of the extension through the chrome extensions settings window. By checking to see how much memory the extension takes up around every 10 tabs, we will gain a better understanding of how much memory the extension takes up and predict how much it may take during usage by users.

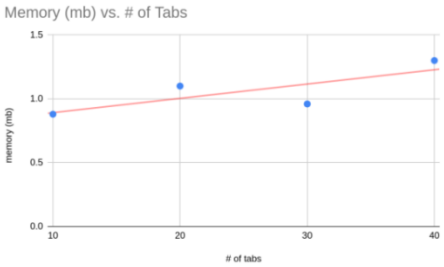

Figure 6. Figure of experiment 2

After performing the experiment, we found that the average amount of memory that the extension took was around 1.06 MB. This amount of memory is not extensive, and actually is quite practical when compared to the productivity boost that the extension offers. The most surprising analysis that we concluded during the experimentation process was that there was not nearly as much of a memory increase as expected from the extension for the amount of tabs we had open. As a matter of fact, we at least predicted that the extension would at least surpass around 2 MB at some point during the experimentation process, however, it did not even come close.

5. RELATED WORK

The Simple Tab Sorter is a Chrome extension on the Chrome Web Store that enables the user to create and title groups of tabs so that the user may sort and organize their own tabs [11]. This extension is useful as it does provide a level or organization, however, it does require the user to manually organize the tabs themselves. The Tab Sorter Chrome extension proposed within this paper sorts and organizes the tabs for the users, so that the user does not have to extend their time organizing them. This feature saves time for the user, and therefore enables the user to be more productive with their time.

Tabber is another extension for sorting and organizing tabs in the browser, however, Tabber is available in most available browsers such as Chrome, Firefox, and Microsoft Edge [12]. While Tabber does have a sync feature, enabling custom tab groups to be shared across devices and browsers, the free version of Tabber sets a maximum of 10 tab groups. In order to use more than 10 tabs and have access to automatic sorting features, the user must subscribe to the monthly subscription plan of $2.99. The Tab Sorter extension is entirely free, and able to sort however many tabs the user can open.

The Tab Sorter 2 Chrome extension, not to be confused with the Tab Sorter extension proposed in this paper as it is a completely different project from a different developer, is designed to give the user a variety of options for how to sort their currently open tabs [13]. Upon investigating the Tab Sorter 2 extension, it is quite obvious that the extension has a variety of options, however, the amount of options and way they are displayed, a single page filled from top to bottom with buttons, is definitely liable to overwhelm and confuse the user. The Tab Sorter extension proposed in this paper remains simple, yet effective, and organizes tabs with no effort from the user.

6. CONCLUSIONS

After reaching a stopping point in the development process and reflecting on the project, it is apparent that the primary issue with the current state of the extension is how the tabs are sorted in a list. The list, while remaining effective and allowing the user to look through it in an easy

fashion, may not be the most effective manner of sorting the tabs. To remedy this issue, we have proposed the idea of adding a search bar feature inside the extension popup window that enables the user to search to find specific tabs by title. This would enable the user to find exactly what they are looking for, without having to sort through the list of open tabs. Additionally, to continue to improve upon the extensions ease of use, adding a bookmarks section at the top of the extension underneath the search bar where the user can bookmark the most important tabs that they have open will enable the user to keep better track of their important tabs while not extending too much effort.

Throughout the development process, the total knowledge and experience gained was of a high caliber, and the Chrome extension of a great quality. The Tab Sorter Chrome extension is a useful tool for anyone looking for an effective and productive way to organize and sort tabs, and will hopefully go on to help many users keep track of their tabs.

REFERENCES

[1] Rathod, Digvijaysinh. "Web browser forensics: google chrome." International Journal of Advanced Research in Computer Science 8.7 (2017): 896-899.
[2] Oeldorf-Hirsch, Anne, and Jonathan A. Obar. "Overwhelming, important, irrelevant: Terms of service and privacy policy reading among older adults." Proceedings of the 10th International Conference on Social Media and Society. 2019.
[3] Collins, Michael G., and John J. Barton. "Crossfire: multiprocess, cross-browser, open-web debugging protocol." Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. 2011.
[4] Lafreniere, Benjamin, Andrea Bunt, and Michael Terry. "Task-centric interfaces for feature-rich software." Proceedings of the 26th Australian Computer-Human Interaction Conference on Designing Futures: The Future of Design. 2014.
[5] Carlini, Nicholas, Adrienne Porter Felt, and David Wagner. "An evaluation of the google chrome extension security architecture." 21st USENIX Security Symposium (USENIX Security 12). 2012.
[6] Hu, Brian, Evan Gunnell, and Marisabel Chang2and Yu Sun. "Smart Tab Predictor: A Chrome Extension to Assist Browser Task Management using Machine Learning and Data Analysis." International Journal of Information Technology (IJIT) 1.6 (2022).
[7] Horsman, Graeme. "A process-level analysis of private browsing behavior: A focus on Google Chromes Incognito mode." 2017 5th International Symposium on Digital Forensic and Security (ISDFS). IEEE, 2017.
[8] Carlini, Nicholas, Adrienne Porter Felt, and David Wagner. "An evaluation of the google chrome extension security architecture." 21st USENIX Security Symposium (USENIX Security 12). 2012.
[9] Smith, Adrian. "Windows and pop-up menus in application design." Proceedings of the international conference on APL. 1987.
[10] Namikawa, Ken, et al. "Utilizing artificial intelligence in endoscopy: a clinician's guide." Expert review of gastroenterology & hepatology 14.8 (2020): 689-706.
[11] Bauer, Lujo, et al. "Analyzing the dangers posed by Chrome extensions." 2014 IEEE Conference on Communications and Network Security. IEEE, 2014.
[12] Oshri, Ilan, Henk J. de Vries, and Huibert de Vries. "The rise of Firefox in the web browser industry: The role of open source in setting standards." Business History 52.5 (2010): 834-856.
[13] Jin, Bihui, Heng Li, and Ying Zou. "Impact of Extensions on Browser Performance: An Empirical Study on Google Chrome." arXiv preprint arXiv:2404.06827 (2024).
[14] Darjazi, Hamideh, et al. "Evaluation of charge storage ability of chrome doped Mn2O3 nanostructures derived by cathodic electrodeposition." Progress in Natural Science: Materials International 26.6 (2016): 523-527.
[15] Johnson, Matthew P., et al. "Energy peak shaving with local storage." Sustainable Computing: Informatics and Systems 1.3 (2011): 177-188.