Contents

Getting Started with Mozilla Webextensions

During 2015 Firefox team has made an exciting very big announcement to Firefox add-ons, they have brought a new extension API's called WebExtension [1]. This update has lot of exciting announcements for developers.

1. Review process will become faster
2. Development is easier, and the Addons will be compatible with Chrome and Opera.

You can find the whole information regarding this update in Add-ons Blog [2]

The next step is to download the Firefox Browser and install them. You can either go for developer Edition [3] or Nightly [4]

So as of now Firefox Will allow only the add-ons which are signed when you start the browser. So for development purpose we have to change the settings.

1. Open the browser
2. in the URL bar type about:config

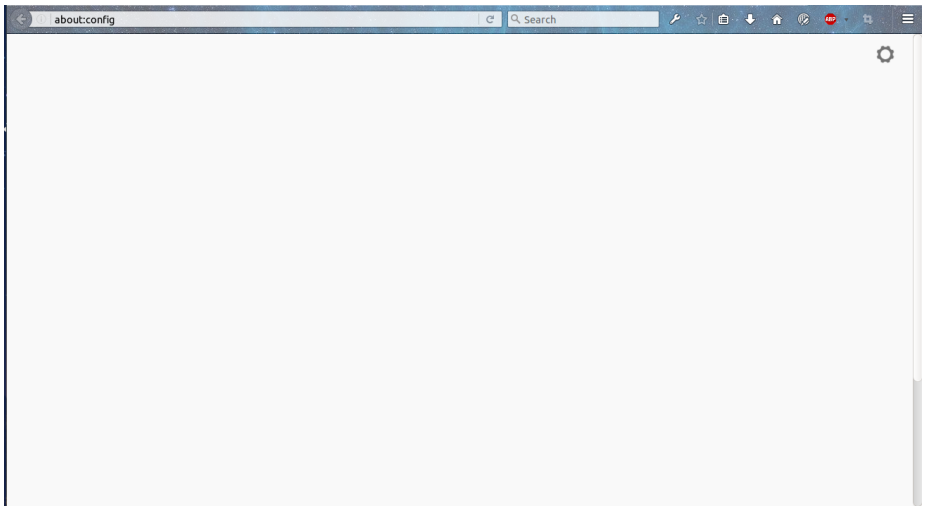

Figure 1: config in URL bar

3. In the search bar of that page type xpinstall.signatures.required , you will see the image as like below. By default it will be true
4. You can double click on it or else right click and select toggle. So it will be set to false

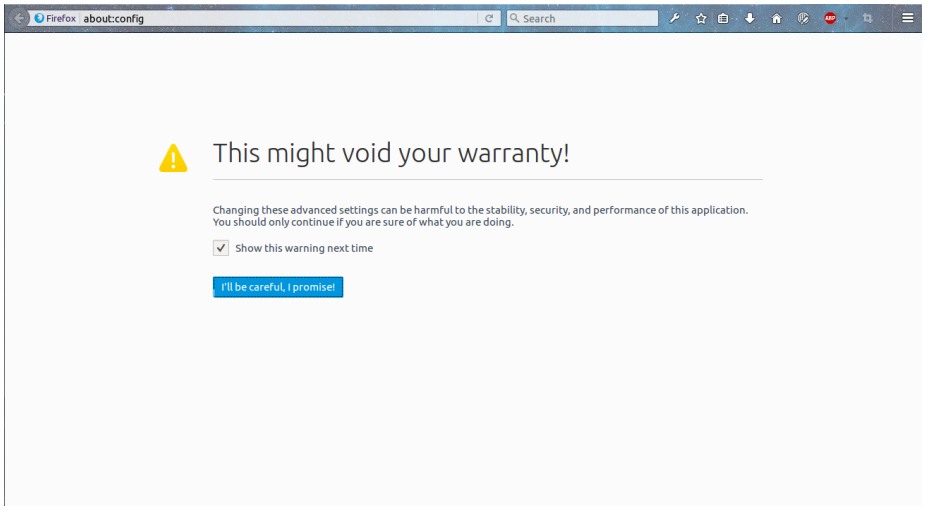

Figure 2: Warning

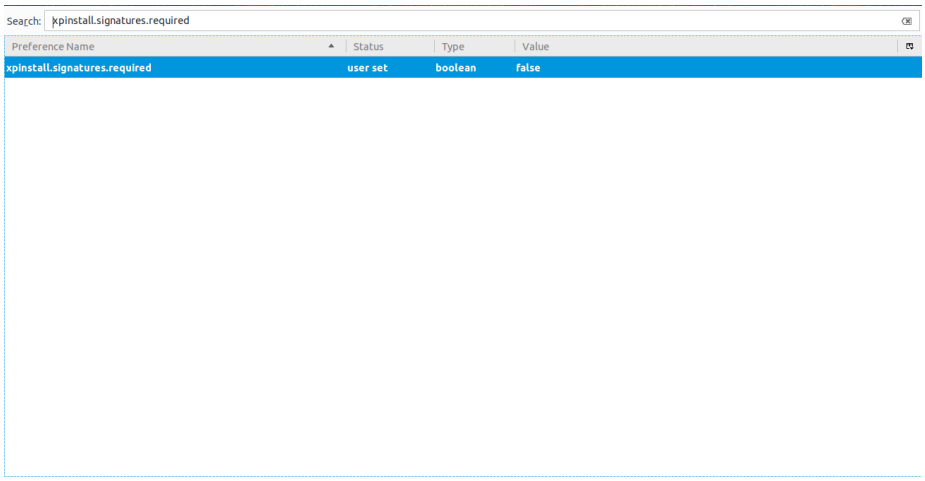

Figure 3: Signature

Why Web Extensions

One of the Main reason, to get started with Web Extensions is easier to develop. Previously I have tried developing some Chrome Extensions during my free time, so since the same thing is used here, it will be very easier for me to get started and develop well. So I write once for Firefox Addons and it will support multiple platforms.

Very excited for this new journey of learning.

References

- [1] https://wiki.mozilla.org/WebExtensions
- [2] https://blog.mozilla.org/addons/2015/08/21/the-future-of-developing-firefox-add-ons/
- [3] https://www.mozilla.org/en-US/firefox/developer/
- [4] https://nightly.mozilla.org/

Running WebExtension in Firefox

In this section we will be exploring more how to run an Firefox WebExtensions.

I will be showing this demo with the help of the small extension I have written before, feel free to download it from github [1], you can clone whole repo [2]. I personally use web-ext [3] tool for packing my extension, but for basic learning it is not mandatory.

Step by Step procedure for running WebExtension

Step 1: Go to the directory which has contents of Extension

In my case it will look like the below. The content showed below is the contents of Extension, which the link is shared above.

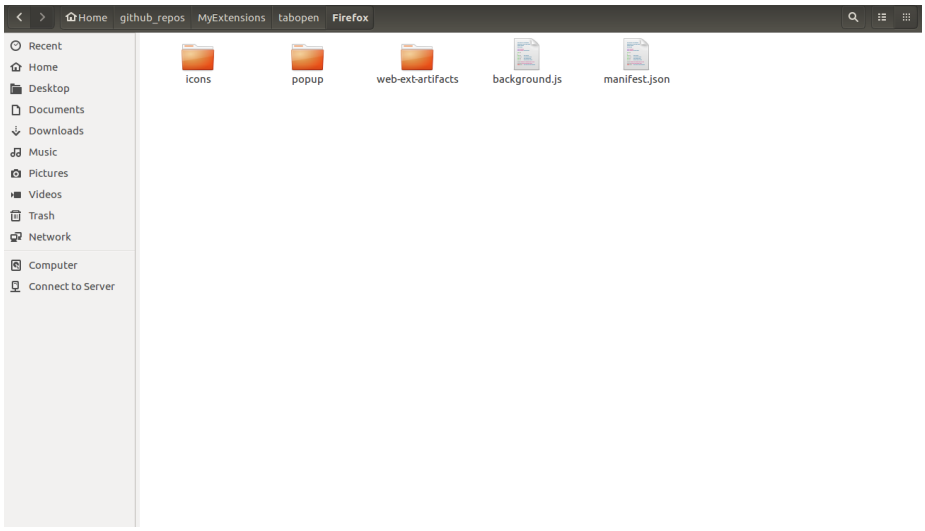

Figure 4: Directory Image

Step 2: We have to compress all the contents into Zip

- Select all the folders and files
- Make it to a zip, anyname.zip

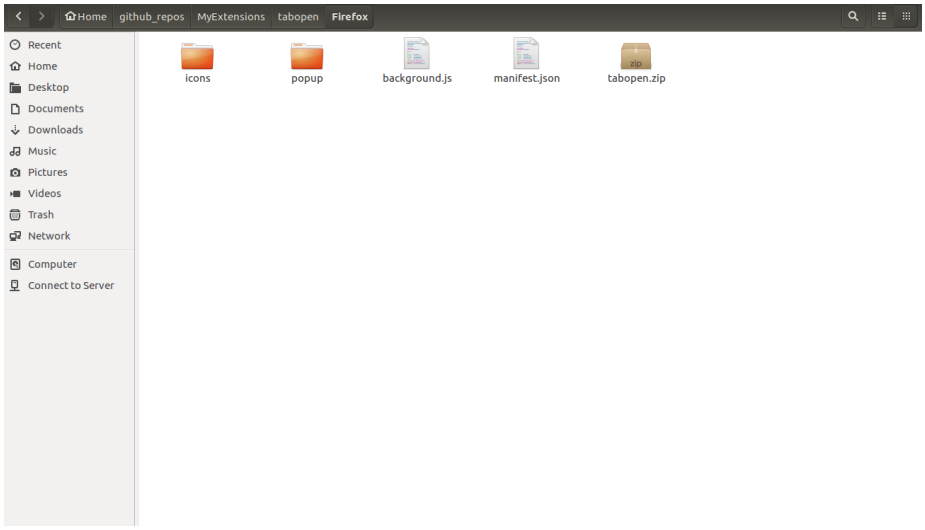

Figure 5: Directory Image

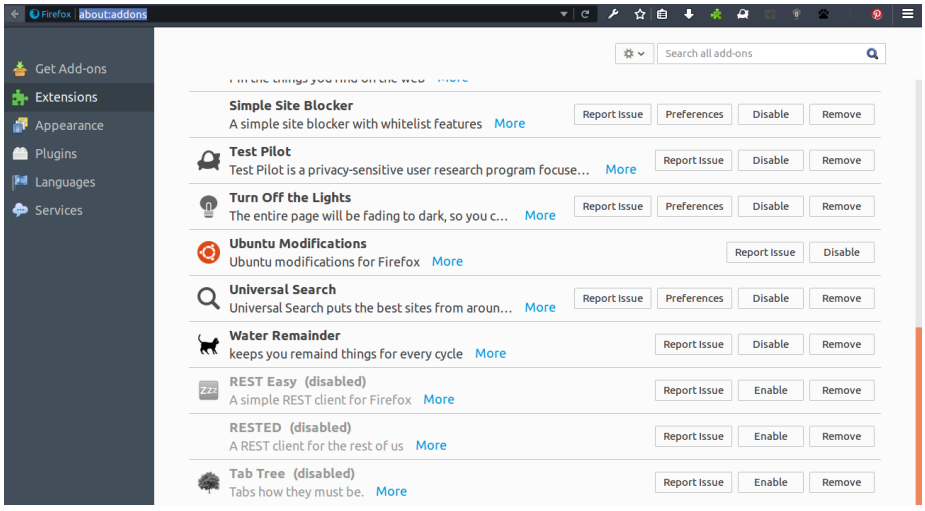

Figure 6: Directory Image

Step 3: Open about:addons or Ctrl+Shift+A

Step 4: Install Addons or Debug Addons

Install Addons

- Click on Tools for all add-ons (the icon - which is like Setting icon)
- Select Install Add-on From File ..
- Make sure you have set the file type to all files
- Choose the Zip which you created in step 2

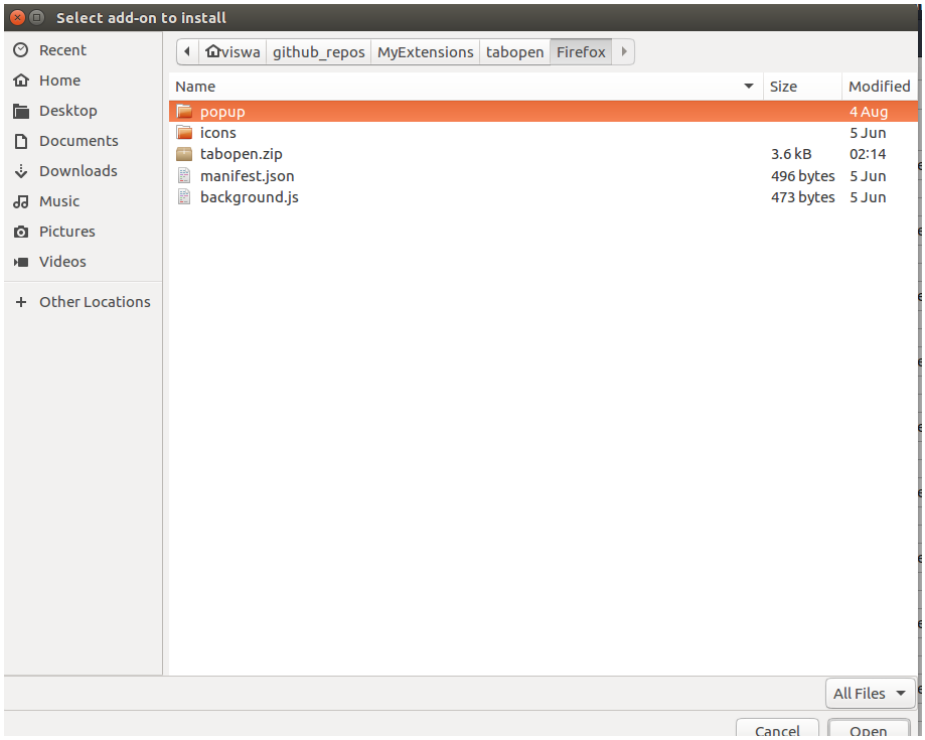

Figure 7: Choose the Zip file

- You will get a pop-up notification after selecting at left croner
- In that pop-up click Install button
- Now your Extension is installed.

Debug Addons

During testing it is enough to go with Debug Add-ons option. Only drawback will be, if we close the Firefox then the Add-on which we installed during our development phase will be uninstalled. Follow steps below.

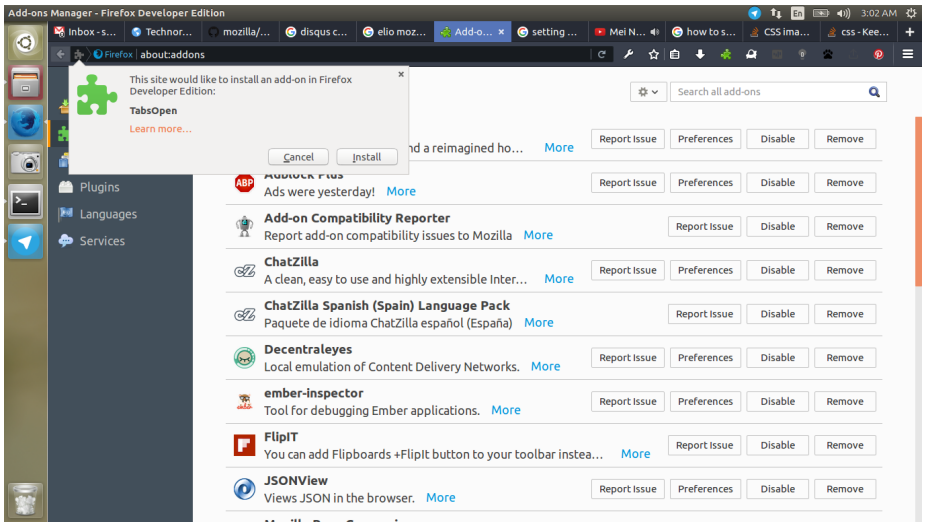

Figure 8: Choose the Zip file

- Click on Tools for all add-ons (the icon - which is like Setting icon)
- Debug Add-on.
- Goto the folder where you source code is present.
- Select the manifest.json file, so the Add-on is installed temporarily.

Step 5: Run the Extension

For most of Extensions you will get the icon in the tool bar itself. So for this a page like icon will be coming at toolbar, if every code is run properly. Click on that icon.

So on Selecting any one of the three websites, it will open a new tab and open the site. As of now it is in static mode.

In future, based on user usage we can show top 3 websites there.

Follow for amazing journey of Building WebExtensions.

References

- [1] https://github.com/iamVP7/MyExtensions/tree/master/tabopen/Firefox
- [2] https://github.com/iamVP7/MyExtensions/
- [3] https://github.com/mozilla/web-ext

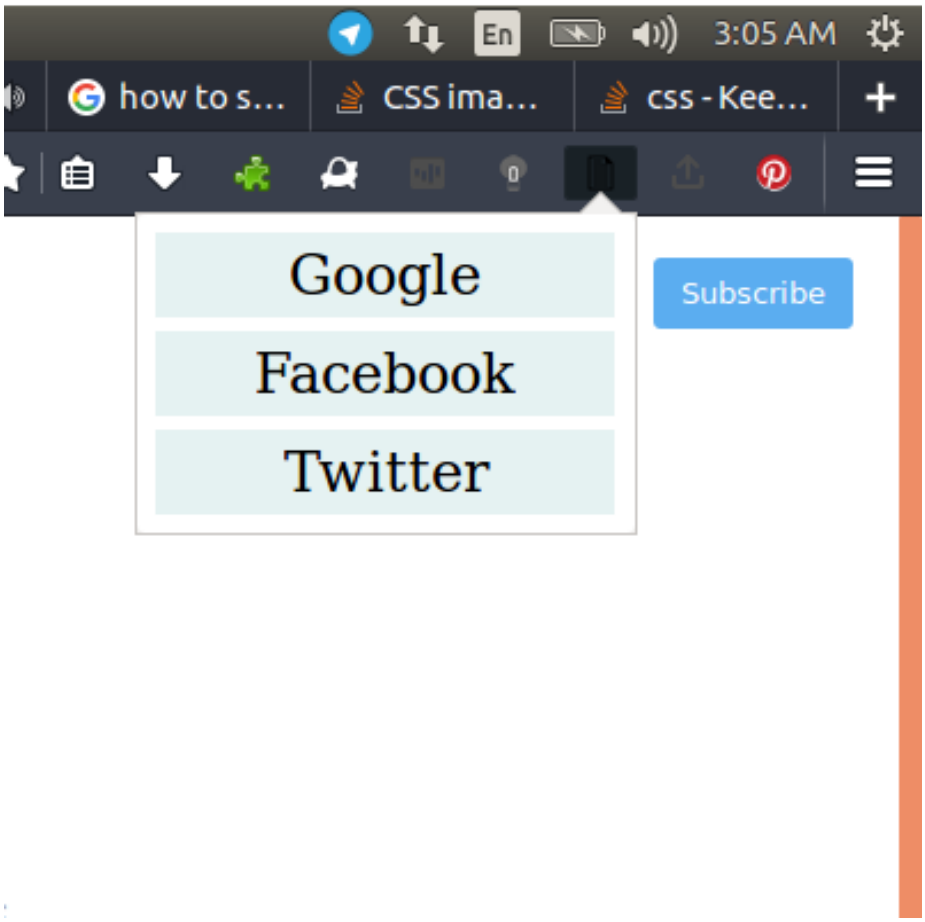

Figure 9: Choose the Zip file

Building our Motivational Tab

We got to know what is meant by webextension and how to install WebExtension from the source code available in our system. Those leasons are very important to us, as we move forward with our development, we will be always using them.

Our Approach is learn the basic code, try them out with little modification and run them in our system, try out the execrise and share about this learning in social medium of community benefit.

Problem statement and Solution

Problem Statement

We have to open different web pages each and every day. Sometime we will be bored to do them. At that time, we may feel to get a cup of coffee. For sure its not possible to count the number of copies we had. Having too much of coffee or other caffeine products is injurious to health.

Solution

So the very first solution is to make sure we are not bored, and we are motivated when we open a new tab. We can be motivated by seeing strong quotes by our favorite personality or a our family pictures whenever we open a new tab.

What is relationship with WebExtension???

Whenever you are opening a new tab, it means we are calling some internal API to open the tab. Each and every action we do is an API, some of them are exposed via WebExtension API. So by default if you check in your browser you will understand you have options to change what should be shown when a new tab is opened. A screenshot from Firefox is shown below, check under the heading *New Windows and Tabs*

How we can capture this

We need not worry much. It is easy to capture when a new tab is opened. And also it is easy to open our preferred page whenever new tab or window is opened. In this section we will be seeing how to open our own page with some random pre defined quotes using *chrome_url_overrides* WebExtension API.

Building Blocks of Our Motivation Tab WebExtension

Writing manifest.json

For every program we will start by writing the manifest.json, as it will be giving us the clear view what we are going to do and what are all the components like permission, background script, resources and so on, we are going to have.

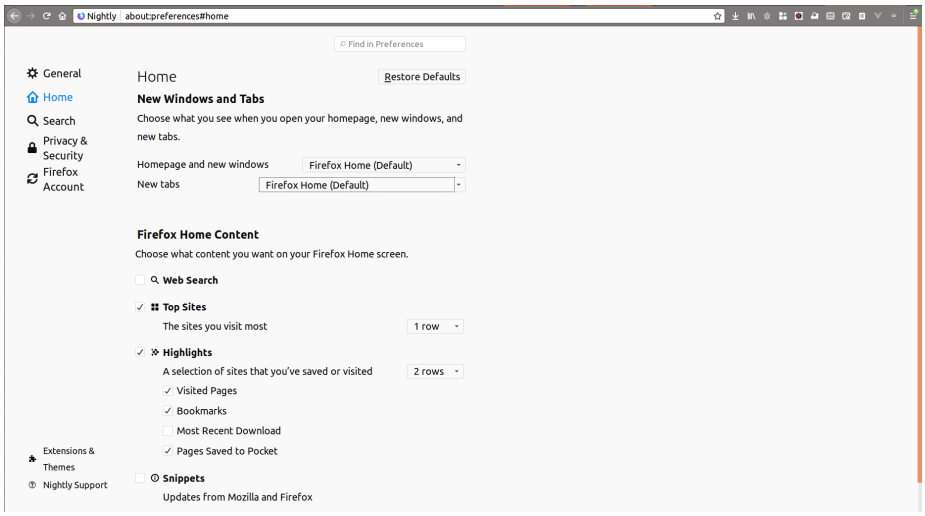

Figure 10: New tab Configuration

```
{
"manifest_version": 2,
"name" : "Motivation Tab",
"description" : "Shows Random Quotes ",
  "version": "1.0",
  "chrome_url_overrides" : {
    "newtab": "my-new-tab.html"
  }
}
```

The above is our manifest.json required for this program. The very important keys which we have to define for the manifest json are as follows

- name
- description
- manifest_version is always 2 (till Mozilla changes)
- version is the version of this addon, and it should be incremental.

Additionally for this particular Add-on (WebExtension) we are going to use *chrome_url_overrides*

What is chrome_url_overrides ??

We can remember it as URL Overrider, means it will be over riding the default with the value we give in the extension. In our example we are saying to over ride the **newtab** , so whenever there is a new tab created if our extension is installed then, our extension will be over riding the other pages and will show

the page which we have defined.

So our definition will be like below,

```
"chrome_url_overrides" : {
    "newtab": "my-new-tab.html"
  }
```

In the chrome_url_overrides defintion, the value of *newtab* is "my-new-tab.html", which is defined by us. We can define what pages we want.

Our Basic HTML Page

Our next step is to write the HTML page.

```
<!DOCTYPE html>
  <html>
    <head>
      <meta charset="utf-8"/>
      <title>The best Motivation Tab</title>
    </head>
    <body>
      <h1 id="quote" style="text-align: center"></h1>
      <script type="text/javascript" src="script.js"></script>
    </body>
  </html>
```

Our HTML page is very minimal, we will be having only **one heading element** with unique id for that, and we will be including the javascript file.

Our Javascript file,

Our JS file will be having random quotes and the colors for the background.

```
let random_quote = [
 "quote1", "quote2", "quote3"
];

let color_hex = ['#778899','#B0C4DE','#E6E6FA','#90EE90','#00FF7F'];
function getRandomQuote() {
  // Generate Random Number for Quote
  var quote_random_num = Math.floor((Math.random() * random_quote.length));
  // Set the content in the div
  document.getElementById("quote").innerText  = random_quote[quote_random_num];
  // Get Random Number for color
  var random_col_num = Math.floor((Math.random() * color_hex.length));
  // Update the background with color
```

```
  document.body.style.backgroundColor = color_hex[random_col_num];

}

// Call the method getRandomQuote when this js loads
getRandomQuote();
```

Thats it, our extension (WebExtension or Add-on) is ready, once we do temporary install we can experience it. When we open a new tab our page by default will look like below.

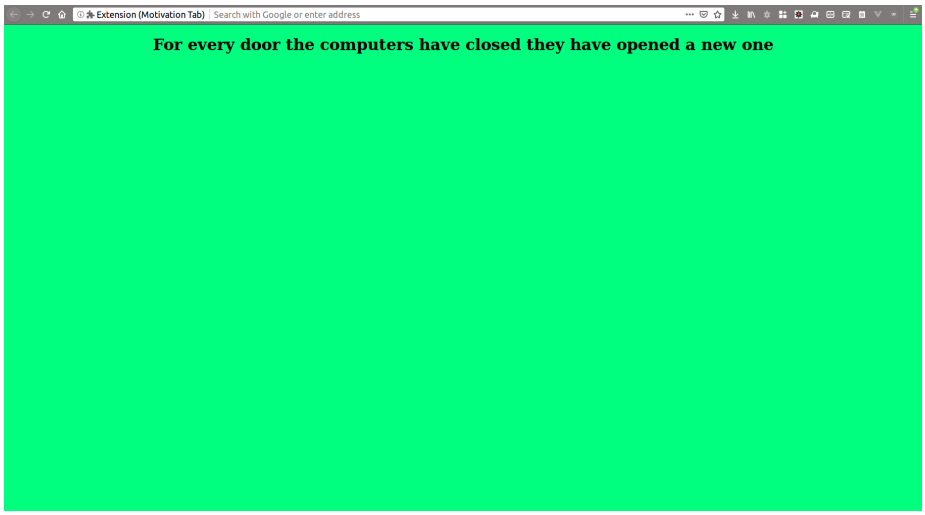

Figure 11: New tab Addon

Exercise

Make sure to visit Mozilla Developer Network (MDN) [1] to learn more about the above API.

So try out this API in your get started program. Your task is simple you have to load beautiful images whenever a user opens a new tab. We have already done opening Beautiful Quotes, instead of words you make it as image. Remembers visuals have more power compared to basic text words.

Optional: It will be great if you can share your code or blog about this learning in twitter. Our Mozilla Webextension twitter handle is (@MozWebExt) and make sure you use hashtag #WebExtLearn when you are tweeting about this learning.

- [1] https://mzl.la/2PBUToi
- [2] https://twitter.com/MozWebExt

Building our Power Search Add-on

We got to many browser actions can be used by developers via WebExtension API(Application Program Interface), we also built our first WebExtension with which we can get motivational images whenever we open a new tab.

Problem statement and Solution

Problem Statement

When we are surfing a lot there are lot of cases where we need to search on different search engines. We always don't need to use Google or Bing alone. Sometimes we will directly search in Wikipedia or want to do video search in Youtube. We are not always active to make everything using Keyboard(KB) because keyboard follow is something like

- select the word using mouse, right click and copy it or Ctrl+C
- goto the website (if we know the address correctly we will go directly or type in google and go there)
- paste the word which we copied and hit enter.

I understand you might have sipped your coffee more than twice here.

Solution

So our good solution will be to use the mouse. The follow with mouse as follows;

- select the word.
- right click.
- search in our preferred search engine by selecting from list.

When you are selecting a word and doing the right click, you will be poped up with list of options. This small window which is coming near your mouse cursor is called as Context Menu.

How we can capture this

Our proposed solution has totally 3 steps. First is selecting the word, once right click is done we have to populate our options (list of search engines available in browser), then once the preferred search engine is selected then we have to create the new tab. As we have 3 steps here we are going to use 3 simple WebExtension API for this.

- Context Menu API
- Search API
- Tabs.create API (Tabs API has so many sub-api inside them, we will go through them in future)

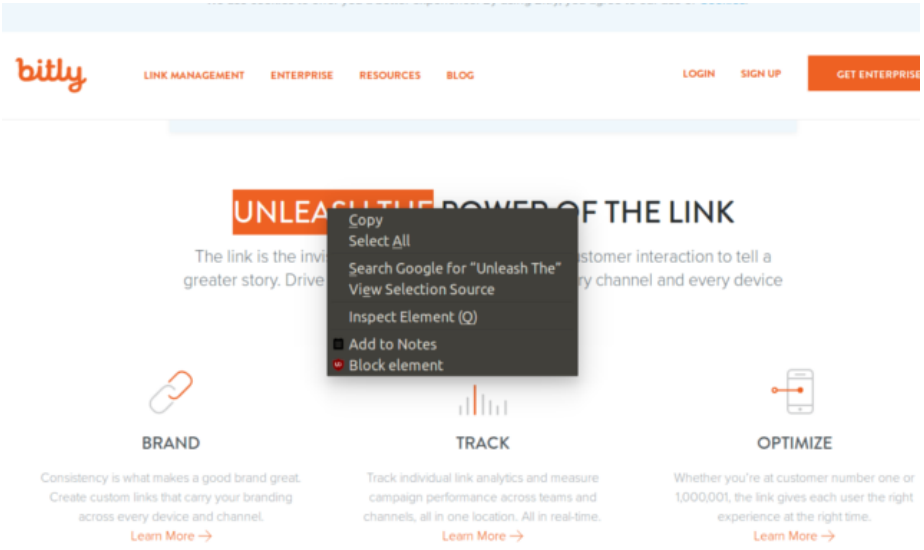

Figure 12: Context Menu

Since our problem statement is clearly defined and we are going to use this three API only, our WebExtension development will be easier and can be finished quickly.

Building Blocks of Our Power Search Add-on

For this extension we will learn the API step by step. So at the end joining them will be very easier.

Create the New Tab

To create the tab, we just need the fully qualified URL which we need to open. We need to give url like "https://www.google.com" instead of "www.google.com"

A sample script is below.

```
browser.tabs.create({
    url:"https://twitter.com/"
  });
```

In the above example, the value of **url** is "https://twitter.com/", we have to change this if we need to open the different tab.

Context Menu API

Context Menu is nothing but when we click the right click a new pop-up comes near our mouse pointer. We can add our own options in the context menu using Browser Action API. It comes under privileged API, so we need to get **"contextMenus"** permission for using context menu. In our program we will be adding the list of search engines in the context.

To add Options in Context Menu

It is very simple to add options in the Context menu.

```
browser.contextMenus.create({
  id: "id1",
  title: "Display Word",
  contexts: ["selection"]
});
```

For each and every context menu item we need to give unique **id** so based on it we can do different actions. In this the value of **title** is what general users see. In Previous image **Add to Notes** is title. Then we need to mention, when we need to show the particular contextMenu item, whether we need to show always or when we select a word (selection), when we select image alone; this is defined in **contexts**

Note: If we have only one contextMenu for our WebExtension then it will be shown directly, if we have more than 1 then under our Extension Name the list will be available.

Getting the selected word

Whenever we click on the contextMenu item we will be getting two things to the listener method we have defined. First one is **info Object** which will have variety of options based ont he context and Second one is **tab Object** which will have information like tabid, url and so on. When we want to start the search in our desired website, we need to have the word which is selected, so from contextMenu item we will be getting the selected word.

The word which we have click will be available in **info Object** with the key **selectionText**

Search API

Search API will be providing us the details like what are all the search engines installed in the browser, which one is default, what is their fav icon. Search API is also privileged one, we need to get **"search" permission** from user.

Get list of Search Engine

Search API will be giving the list of search engine installed in our browser. As discussed before we may be having more than 1 search engine in our browser. If its not available we can install from addons.mozilla.org it will be easier and helpful.

```
var list_of_searchengine = browser.search.get()
```

The above simple snippet will be fetching and giving us the list of search engine available in our machine. Each search engine will be having the values like name i.e., search engine name , may have favIconUrl i.e., search engine fav icon URL. We may be using this two things.

Searching in Preferred Engine

Once we got the selectedText from contextMenu our next step would be to search in the preferred search engine. To search in our required engine below code snippet will do.

```
browser.search.search({
    query: "word to search",
    engine: "Wikipedia (en)"
  });
```

In this case **query** which will have the word to be searched in mandatory, **engine** is not mandatory, if we are not mentioning it, then default one will be taken. In case we mentioned it wrongly exception is thrown.

Assembling our Power Search Extension Parts

Previously we learned about different parts for building our Power Search Extension. We have to create context menu using the search engines avaialble, get selectionText value and search it in the search engine we have.

Writing manifest.json

As usual lets start with writing our manifest file. Here in this time we are additionally introducing three new keys for manifest.json

```
{
"manifest_version": 2,
"name" : "Power Search Engine",
"description" : "Search the selected text in other search engines in your machine",
"version": "1.0",
"icons": {
    "96": "icons/icon-96.png"
```

```
  },
  "permissions" : [
    "search","contextMenus"
    ],
"background": {
    "scripts": ["background.js"]
  }
}
```

icons

icons is the JSONObject, which can be used to show the icon for particular extension. It is good pratice to have beautiful icons for our extension. The normal size we should include is 96px X 96px, 60px X 60px.

permissions

Permission are mandatory if we are going to use API with specific data. In this experiment we are going to use search API and contextMenu API. We have to request permission from user. Only if user grants permission we can run this extension.

background

As the name suggest we will have something running in the backgroud. Here we can define array of scripts so they will be useful and will be listening to everything happens around our extension.

Note: In this program we wont be using any HTML pages, without even HTML page we can run the WebExtension

Our Background script file

In this experiment we are using only one background script namely background.js; we should have included this in manifest.json else we cant use it.

```
createMenu();

function createMenu() {
// Get the list of Seach Engine
browser.search.get().then(buildContextMenu);
}
```

So our First step we are going to do in the background is to fetch the serach engine and once we are done with fetching we are going to build the context menu.

Whenever the script is loaded, the very first method **createMenu()** is called.

In this method we have done steps to get the search engine **browser.search.get()**.

.then(some_function_name) represents once the previous process is done successfully jump to the method *some_function_name*

So in our case once we have got the search engine list using WebExtension API we will be going to the function names **buildContextMenu**

```
function buildContextMenu(searchEngineGot) {

// Iterating Search Engine One by One
for (let searchEngine of searchEngineGot) {

// Create ContextMenu Item
  browser.contextMenus.create({
    id: searchEngine.name, // searchEngine name is set as id.
    title: searchEngine.name,  // searchEngine name is set as title.
    icons: {
      "32": searchEngine.favIconUrl  // searchEngine favicon url is set as icon
    },
    contexts: ["selection"] // context is when we selected a word
  }); // End of contextMenu creation

} // End of iteartion of SearchEngine List

} // End of method
```

Once we got the list of searchEngine successfully we are going to method named **buildContextMenu**. The parameter which we will get here is list of SearchEngine Object. This is defined by WebExtension API and we should follow the params as it is.

In this method we are having so many searchEngine's in **searchEngineGot**, so we are iterating one by one.

In one searchEngine we are having **name** and **favIconUrl** value. Additional values are also present but we are not using it. We are then creating the contextMenu with **name** as the *id* and *title*. Then we have to **favIconUrl** which we are using as icon to dispaly the menuitem.

```
//When Menu Item is clicked we will goto function menuItemClicked
browser.contextMenus.onClicked.addListener(menuItemClicked);
```

After the menu items are created our next step is to listen whenever an menu item is clicked. This is also provided in WebExtension API. Whenever a menu item is clicked we can have a listener and pass the values to listener.

```
function menuItemClicked(menu_info_object, tab_object){
// First we have to check whether there is selectionText and also the menuItemId, both shou
```

```
  if(menu_info_object != null && menu_info_object.hasOwnProperty('menuItemId') && menu_info_
    browser.search.search({
      query: menu_info_object.selectionText, // Selection text is the text which we selecte
      engine: menu_info_object.menuItemId // is the searchEngine.name which we defined prev
    });
  }
}
```

We have created a Listener named menuItemClicked; so whenever an menu item is clicked our code flow control will be passed here. We will be getting two objects, one is menu item information object and another one is tab object.

First we will be checking whether we have the menuItemId and the selected text. Both of them should be not null. And we can also have a additional check whether menuItemId we got matches with the list of search engine we have in our machine.

Once all the preliminary checks are done we will be at our final stage. Now our only step pending is to use the query text and menuitemId(search engine) and search in the respective Website. *browser.search.search* will accept the JSONObject, where *query* key is mandatory here we will be passing the selectedText and for search engine we have *engine* key.

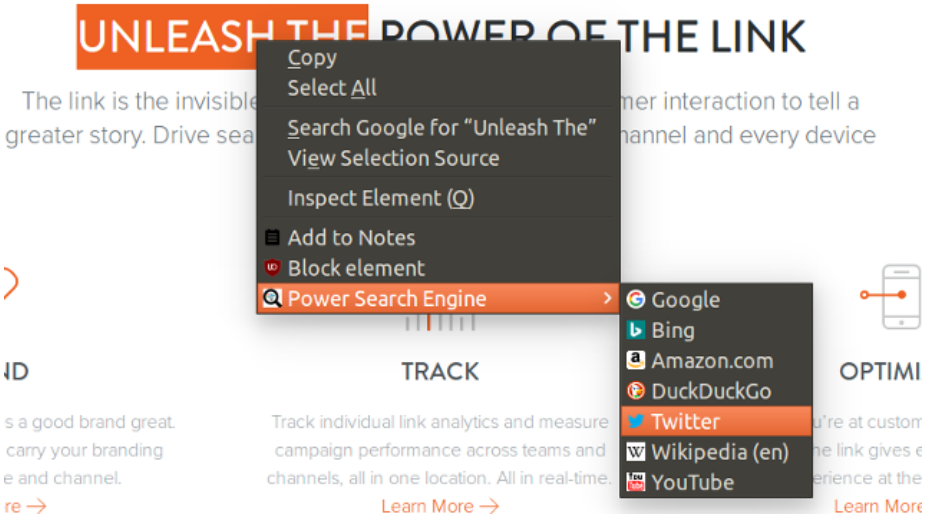

Figure 13: Power Search Addon Demo

Just in case, if you want to search in the current tab itself instead of opening in new tab, you have to make slight changes.

- First in permission you additionally have to get **"activeTab"** Permission

- Second you have to make small modification in the *browser.search.search* API.

```
browser.search.search({
      query: menu_info_object.selectionText, // Selection text is the text which we selecte
      engine: menu_info_object.menuItemId, // is the searchEngine.name which we defined pre
      tabId: tab_object.id // We will be passing the current tab id here.
    });
```

Exercise

To learn More about Search API [1] and ContextMenu API [2], visit MDN link given below.

Your simple task here to find out the various search engines (around 15-20). Make them listed in your ContextMenu. This task will involve you to understand the basic how the search Query params works.

For Example: In Wikipedia if you are search **hello world** you will be getting the URL like below

Wiki URL: https://en.wikipedia.org/w/index.php?search=**hello+world**&title=Special%3ASearch&go=Go

In this only the highlighted part will be changing everytime. Likewise find for other search engines also and create your extension. You may have to decide what title you have to give, id you have to give.

You may also have to do some little parsing.

Optional: It will be great if you can share your code or blog about this learning in twitter. Our Mozilla Webextension twitter handle is (@MozWebExt) and make sure you use hashtag #WebExtLearn when you are tweeting about this learning.

- [1] https://mzl.la/2q13lSO
- [2] https://mzl.la/2J3k7tc

Building our Tabs Closer

We have seen previosly how to search using different search engine available in our browsers. It should interesting and easy one right ? Hope the execrise you tried might have left to finding the URL of different search engines and what are all the extra information they are sending as request params.

Problem statement and Solution

Problem Statement

Now when we are star surfing internet, we end up opening atleast 10-20 tabs, because each and every page have some many hyperlinks which we love to learn and come back, but in reality hyperlinks in that pages leads to different page again there are so many hyper links. In such a case if you are in college or office closing the social media websites (some of us do, we dont love to show it to our boss) will be difficult have to close so many tabs after finding them.

Solution

The simple solution will be to have the list of social websites we visit and with one click, we can close all the tabs. The another solution will be to list all the tabs we have close from an pop-up one by one; this will work great when we are having more than 20 tabs.

How we can capture this

Breaking according to two solutions.

Showing the pre-defined Social Network sites

In this type, we will be pre-defining the one social media website which we visit often. We will be having the icon to close that particular social media website alone. On clicking the icon, all the tabs having that social media website URL should be closed.

Fetching all opened URL

Another approach will be fetching all the opened websites, listing them one by one and when we click on close button we can close them. Just incase we need to visit them we will have another button to visit the particular tab.

In this execrise we will be learning the following list of API

- Tabs.query (To fetch all tabs)
- Tabs.remove (Close)
- Tabs.move (To display the tab clicked)
- Browser Action Icon
- Browser Action pop-up

Building Blocks of Our Social Media Tab Closer

First we will take a look at the API we will be using in this Execrise.

Getting started with Basics of API

Starting the Closing Operation

First step is we need to click on the browserAction icon. As the name suggest browserAction icon will have to scope throughtout the browser and it is not specific to any one particular tab. So whenever we click on it irrespective of tabs we can do the procedure. Once we have click the icon we need to capture the click action.

We have browserAction API for capturing this. *browserAction.onClicked* will be used to listen to click action on that icon.

```
browser.browserAction.onClicked.addListener(action_jump_function_name);
```

Fetching the twitter tabs

In this case we will be knowing the website URL of Twitter, it is nothing but https://twitter.com/ . So our first step will be to get all the tabs where Twitter is loaded. Our WebExtension *tabs.query* have lot of combination to get(or fetch) the tabs loaded, in this case we are going to fetch based on **URL**

```
browser.tabs.query({url: "*://*.twitter.com/*"});
```

Some times we wont know whether its *http* or *https* so we are using the regex (Regular Expression) *://* before twitter.com and also we may be having anything loaded so we have /* at the end of .com

Close the tabs

Once we have got the tabs (or single tab) we need to close them. We have to get the tabid from the tab object we got, based on that only we will be closing the tabs.

```
var removing = browser.tabs.remove([tab_id1, tab_id10, tab_id15]);
```

Assembling parts of our Extension

So now our task is to just write all the above code in manifest.json and background script file. Mos to fhte things are coverd.

Writing manifest.json

```
{
  "manifest_version": 2,
  "name" : "Social Tab Closer",
  "description" : "Close the pre-defined Social Media Tabs",
    "version": "1.0",
  "icons": {
      "96": "icons/icon-96.png"
    },
    "permissions" : [
      "tabs"
    ],
  "background": {
      "scripts": ["background.js"]
    },
    "browser_action": {
      "default_icon": {
        "96": "icons/icon-96.png"
      }
    }
  }
```

Here changes we made will be getting the tabs permission. And also we have to define the brower_action icon alone. We wont make any other drastic changes compared to previous manifest.json.

Writing our background.js

We will be splitting the background javascript into parts.

- Listen to browser Action Icon
- Fetch the twitter tabs.
- close the tabs.

Listen to browser Action Icon

```
browser.browserAction.onClicked.addListener(icon_clicked);
```

Our first step will be to listen whenever our browserAction icon is clicked. Once it is clicked we have to pass the control to the method (icon_clicked) which we have defined.

Fetch the twitter tabs

```
function icon_clicked(tab){
  var querying_tabs = browser.tabs.query({url: "*://*.twitter.com/*"});
```

```
  querying_tabs.then(close_tabs);
}
```

Our next step will be to fetch(query) the tabs where twitter.com is loaded, so we will be using the regex which we discussed before and fetch the tabs with the url pattern.

Close the tabs

```
function close_tabs(twitter_tabs){
var array_of_twitter_tabid = [];
  for(let single_twitter_tab of twitter_tabs){
    array_of_twitter_tabid.push(single_twitter_tab.id);
  }

  browser.tabs.remove(array_of_twitter_tabid);
}
```

Our Next step is to get the tab id with the tabs we have fetched. We are first iterating each and every tab object which we are getting in tab array as a result of query using URL. Then while iterating we are getting the **id** and inserting in the array. After we finish the iteration we are just using the close (remove) API and closing the tabs.

Building Blocks of All Tabs Closer

We are going to almost use same set of API which we used for our social tab close. Here we will additionaly have a very small change in manifest.json.

Getting started with Basics of API

Adding popup html page

Previously we have mentioned the browserAction icon and just left. Now in manifest.json we are additionally going to add the html page which has to be shown when browserAction icon is clicked and also the title which has to be shown when we move our mouse above browserAction icon.

Note:We can either Click on the browser Action icon or we can show the HTML page

Fetching the all the tabs

Previously we have passed the url and fetched the tabs where twitter.com is loded. Now simply we dont need to pass anything so we will be getting all the tabs.

```
browser.tabs.query();
```

Close the single tab

We are going to close the tabs one by one, so instead of passing the array of tabIDs we can pass only one tabID also.

```
var removing = browser.tabs.remove(tab_id1);
```

Assembling parts of our All Closer Extension

Lets start with manifest.json, for now will share the whole content, in future we can slowly start sharing only the parts which are having difference.

manifest.json look

```
{
  "manifest_version": 2,
  "name" : "All Tab Closer",
  "description" : "Close All Tabs",
    "version": "1.0",
  "icons": {
      "96": "icons/icon-96.png"
    },
    "permissions" : [
    "tabs"
    ],
  "background": {
      "scripts": ["popup/script.js"]
    },
    "browser_action": {
      "default_icon": {
        "96": "icons/icon-96.png"
      },
      "default_title": "List all tabs",
    "default_popup": "popup/tabs_list.html"
    }
  }
```

So our change will be mostly in the **browser_action** part. We have additionally added two more keys for it. Most of the parts remains quite familier and its based on this Add-on.

Building our tiny HTML page.

We need a HTML page which will be acting as a popup when we are going to click on **browserAction** icon. So we need to design very small page.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <title>List of Tabs</title>
</head>
<style>
    .single_div {
        background-color: lightgrey;
        border: 5px solid rgb(11, 12, 11);
        padding: 25px;
        margin: 25px;
    }
    </style>
<body>
  <div id="tabs_list" style="width: 300px">
    </div>
  <script type="text/javascript" src="script.js"></script>
</body>
</html>
```

Our HTML page is having some css for design and most important part is the div with id *tabs_list*, we will be creating all our tab details and will show here.

And we have also included the Javascript file at the end. This same JS file will be acting as background script file.

Included JS and Background JS

Our first action once the javascript file is loaded is to collect all the tabs which are opened. We are processing this action in **startFetchingTabs** method

```
startFetchingTabs();

// Fetch all the tabs
function startFetchingTabs() {
  var querying = browser.tabs.query({});
  querying.then(create_list_tabs, onError);
}
```

After we have collected the tabs our action will be to list them in the popup browserAction page. So we are itearting tab object one by one. tab object will be having the details like tabID, tabTitle and tabURL. We as of now need tabID and tabTitle.

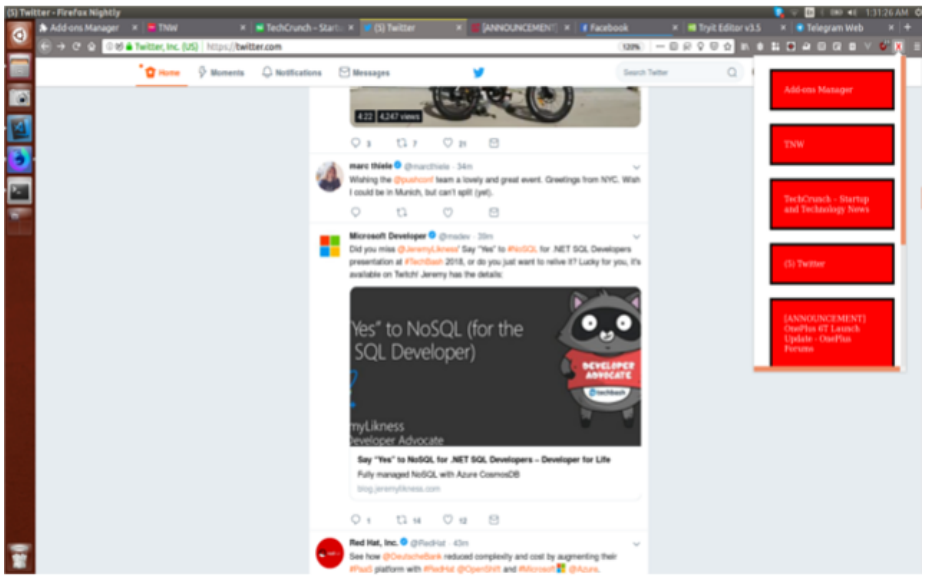

Figure 14: Popup Before Closing any tab

First we will be deleted all the already listed tabs in the pop-up HTML page, then after that we are creating the div one by one for all the tabs opened.

```
function create_list_tabs(fetched_tabs) {
  // delete already existing child nodes.
  if (document.getElementById("tabs_list").childNodes.length > 0) {
    var myNode = document.getElementById("tabs_list");
    while (myNode.firstChild) {
      myNode.removeChild(myNode.firstChild);
    }
  }

  // Create new child nodes.
  for (let single_tab of fetched_tabs) {

    var created_div = document.createElement("div");
    created_div.style.width = "200px";
    created_div.style.background = "red";
    created_div.style.color = "white";
    created_div.innerHTML = single_tab.title;
    created_div.id = single_tab.id;
    created_div.className = "single_div";
    created_div.addEventListener("click", function () {
      close_tab(single_tab.id);
```

```
    }, false);

    document.getElementById("tabs_list").appendChild(created_div);
  }
}
```

We have created the div in such a manner, the tab Title will be displayed to the user. And each and every title is associated with tabID. When we click on any part of the div, the tab ID is passed to **close_tab** method and there we will start the process of closing the tab.

```
// Close the tab and recreate the popup html
function close_tab(tab_id_to_close) {
  browser.tabs.remove(tab_id_to_close);
  startFetchingTabs();
}
```

As discussed before, once the title is clicked, the respective id is passed and we will be closing the tab with that particular ID after that we will again be creating the popup browserAction HTML page.

Exercise

Make sure to visit Mozilla Developer Network (MDN) to learn more about the above API.

- Close API [1]
- Query API [2]
- Capture Tab [3]
- Reload Tab [4]
- Browser Action [5]

There are lot of things related to tabs API to be explored. We have as of now have used the title and showed the list of tabs. But sometimes Title alone wont be enough, we have an API to capture the current instance of the tab as image, use this API instead of title make sure to show the image and on image have two options one for reload and another for close. So based on the button clicks use Reload API or Remove API.

Optional: It will be great if you can share your code or blog about this learning in twitter. Our Mozilla Webextension twitter handle is (@MozWebExt) and make sure you use hashtag #WebExtLearn when you are tweeting about this learning.

- [1] https://mzl.la/2QZ1PfB
- [2] https://mzl.la/2PfiZYW
- [3] https://mzl.la/2yLRH2g
- [4] https://mzl.la/2yrb5SZ

- [5] https://mzl.la/2OCdEf1