Tab Saver Implementation

For this project we used two API's to help with our main goal of saving lost tabs. The idea is that this chrome extension will be able to save all tabs when the browser closes and allow a user to easily view the lost data. We first used the *chrome.tabs.query* API call in order to get the tab data in the current browser. Later on we use *chrome.storage.sync.get* and *chrome.storage.sync.set* in order to save and retrieve the history. Furthermore, we used *chrome.windows.onRemoved* which created a listener that listened for the browser closing. When the browser is closed, data is then moved from temporary storage to permanent storage. A list of the functions and their purpose:

addToTabList() - This function takes a list that will be returned from the *chrome.tabs.query* call and displays the elements properly in the "tabs" div that is created in our HTML file.

temporarilySaveData() - This function gets the current date and time and stores it to a list called "data" in storage. Since data has to be temporarily stored because it needs to be appended to the permanent history, addToHistory is then called which will move temporary to permanent.

addToHistory() - This function first reads the current array from storage using *chrome.storage.sync.get* and then uses *unshift* in order to append the new data to the array list. This new data is coming from the temporary storage. Then, the new array is set as the "data list" using *chrome.storage.sync.set* and the temporary storage is erased and can be used again.

removeElement() - This function was the most complicated. This function will take an index sent from the 'x' button next to each link in the history list. For every date, it has its own index.

Example:

Index 0 contains :

10/12/2015 at 11:10:00 AM

- www.google.com
- www.cnn.com

Index 1 contains:

10/15/2015 at 12:30:00 PM

- www.abc.com
- www.facebook.com

Using the index from the button, we can also get the link from the list. We than go through the list and replace that index with an empty string so the element is erased. Similar to before we then have to add the array back to storage by first pulling the current list, and then setting it to the new one using *chrome.sync.set*.

Summary Component

Another component that this application contains is the ability to view a short one-sentence summary of each link. The way it is implemented is using an AJAX call to an API provided by textance. This is how we did it.

```
$.ajax({
	url: "http://textance.herokuapp.com/title/www.google.com”),
	complete: function(data) {
		console.log(data.responseText);
	}
}
```

This will return the title for each link we send a request for. Currently, the link summaries can only be viewed in the console. This is where we ran into many issues. We had a very hard time trying to get the response to show up in the popup itself. The reason that we found. was the fact that AJAX calls are asynchronous. This means that even though the piece of code that sends the list of the URLs to get called by AJAX was run, the rest of the code continued to run and didn’t wait for a return. We tried turning that async feature off as well as waiting for the AJAX call to finish using *success* instead of *complete.* However, when we did this, it caused the popup to run really slow and sometimes fail when trying to add items to the history div. For now, if you right click on the popup and press “Inspect Popup”, the summaries will show up in console in no particular order (because of the asynchronous property).