Magic Markup

A Google Chrome Extension

By: Alverto Ortega-Garcia

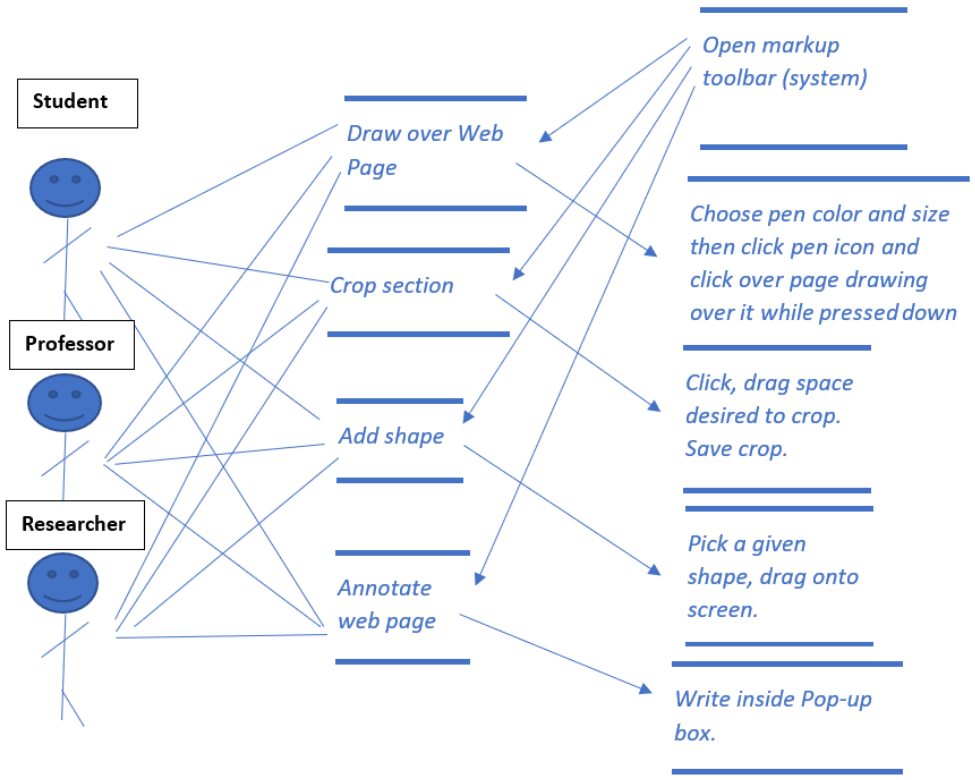
8/18/2020

I. Introduction:

The project I created is a Google Chrome Browser extension called Magic Markup. The main objective of this google chrome extension is to provide the client/user all the basic fully functioning markup and note taking tools on the fly with additional customization tools that will increase productivity and a more personalized experience on any Chrome web page. This application will follow minimalism guidelines for user-interface design and usability criteria for an effective markup tool dedicated, but not limited, to students and professors.

The System features includes: Markup(drawing) on a web page, Color and size changing options of the pen, and a saving option which takes a screenshot of your work done on any web page that allows you download onto any directory you choose. Some other features on the works are adding shapes on the page, text, annotation container box and cropping sections of a web page to save.

II. User Stories/cases:



III. High-level Architecture Diagram:

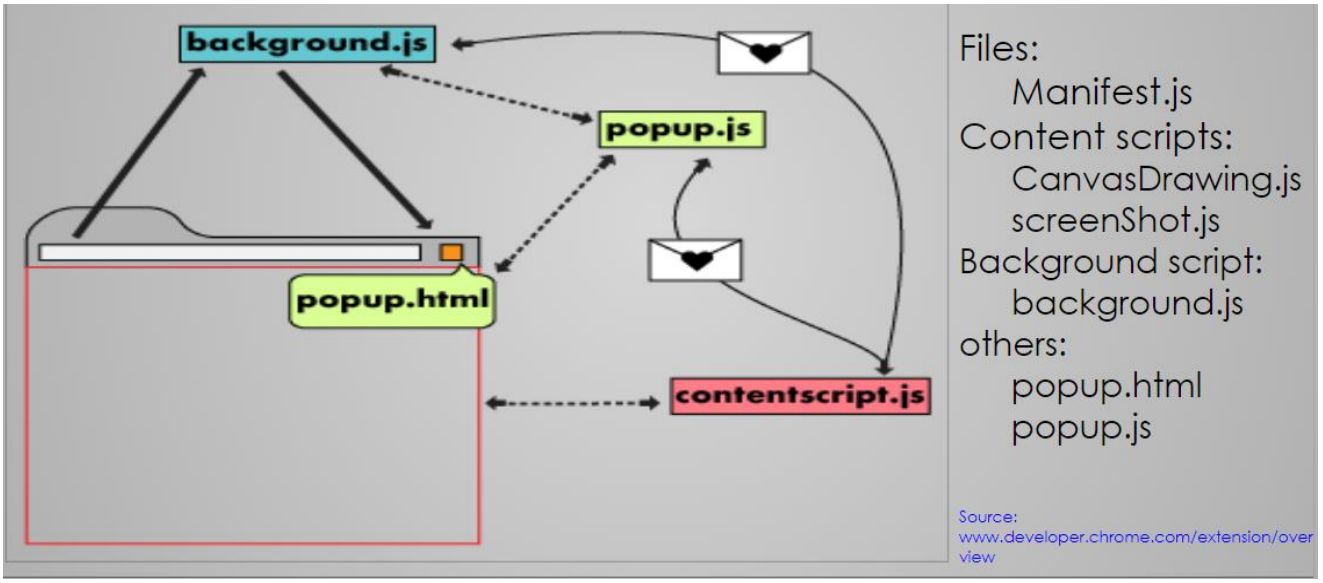


Figure 1: Required and interaction of files for the Chrome extension

The Chrome extension requires there to be at least two files in order to interact with the chrome browser. The files are the manifest file, and either the background script or content script file.

The manifest file gives the browser information about the extension and grants permission for use of functions such as adding hotkeys or granting access to certain APIs. The background script or Content script cant be used either individually or as a component, they both have access to the Chrome API but the background script has full chrome API access and the content script is the file that interacts with the web page DOM by injecting scripts into the page. The popup files correspond to the limited user interface that the extension can have.

IV. Design implementation:

In designing my extension application, the libraries used were kept at a minimum and in doing so only the Math library was used in order to efficiently grab the width and height of the visited web site for canvas measurement in certain scenarios were a page didn't include a scrollbar. The API's used were the Chrome API, JavaScript API and the HTML Canvas API. The Chrome API is necessary for the extension to work properly on the Google Chrome Browser.

Within the Chrome API, the stable APIs used were tabs , runtime and storage. Using the tabs API, the most important examples in my implementation:

```
    chrome.tabs.executeScript(tabs[0].id, {file: "CanvasDrawing.js"}, func
tion(){
```

This line of code from my file allows injecting JavaScript code into the browser page by calling executeScript from the tabs API

.

```
        chrome.tabs.sendMessage(tabs[0].id, {greeting: "theClearAction"});
```
Calling sendMessage allows message passing or passing data to other content script files from your extension.

```
 chrome.tabs.captureVisibleTab(null, {format : "png"}, function(data) {
```
Calling captureVisibleTab allows Chrome to get a data image url of the visible we page from the current tab the user is in. This data was used in order to capture a screenshot of the visible page.

Using the runtime API:
```
      chrome.runtime.sendMessage({greeting: "take screenshot"},function(re
sponse)
```
Similar to tabs.sendMessage except that runtime only allows to send single messages to background scripts for handing events in the extension.

```
 chrome.runtime.onMessage.addListener(
```
calling onMessage from a background script listens/receives the message/data passed from the sendMessage object.

Using the storage API:
```
chrome.storage.local.set({
  "theColor": Color
 });
```
This code stores and tracks changes to user data in this case the color of the pen as data.

```
  chrome.storage.local.get(["theColor","theRange","theStop"], function (it
ems){
```
This line of code retrieves that data in another file in order to be used and applied in my content script. Adding the code items.theColor gives the data result.

The JavaScript API was used in conjunction with the HTML Canvas API since JavaScript code was used to manipulate HTML withing the document. In using the canvas API, the core algorithm was developed that provided the main feature of the extension which was coded in the JavaScript language.

In my one and only HTML file called popup, this was the design aspect of my user interface system that pops up after clicking my extension icon which provides the tools for the user to use. Communicating with those variables and elements through JavaScript I was able to get a responsive web site featuring my functions with the canvas API.
Before using the canvas API, I needed to create a container for the canvas so that I can use percentage values as a style attribute to cover the entire area of a web page:
```
      var canvasContainer = document.createElement('div');
document.body.appendChild(canvasContainer);
```

Once that was done, I then created the canvas element using the canvas API and appended it to the div container as follows:
```
 var myCanvas = document.createElement("canvas");
canvasContainer.appendChild(myCanvas);
```

Adding the proper needed attributes to myCanvas I then added the following code to position this canvas on top of the web page to be able to draw over it with a transparency background:

```
myCanvas.style.zIndex = "2030034534533";
```
the z-index number value used could be any high number in order to prevent any element from the background page to run over the canvas which is what we want to avoid.

Afterwards, the 2D rendering context was used from the Canvas API which allows drawing on the surface of the canvas element. Initiated as follows:

```
      var context = myCanvas.getContext('2d');
```
This allowed attributes such as strokeStyle which added color to the drawings, and linewidth to chance pen drawing size.

Using the myCanvas variable for the canvas element, mouse event listeners were added to handle properly drawing over the page as the user desires.
For example, to listen as soon as the mouse was clicked and properly set that position of the mouse cursor the following algorithm snippet was used:

```
myCanvas.addEventListener('mousedown', e => {
        x = e.offsetX;
        y = e.offsetY;
```
the offset property of the mouse event provides the X coordinate position of the mouse pointer which is on the horizontal axis.

Calling a function “draw” I passed those values on to the canvas in order to use the 2d properties mentioned earlier for drawing as desired.

```
context.beginPath();
context.moveTo(x1, y1);
context.lineTo(x2, y2);
context.stroke();
```
the above code sets the path from those given coordinates and starts drawing by calling stroke().

Finally, the screenshot function which was implemented in the background script used the combination of canvas API, JavaScript API and the Chrome API. For example, a canvas was used to draw the content of the web page:

```
      var image = new Image();
var context = canvas.getContext("2d");
      context.drawImage(image, 0, 0);
```

The visible content data was grabbed from the following code:

```
chrome.tabs.captureVisibleTab(null, {format : "png"}, function(data){…
```

V. Deployment guide/screenshots:

The following is the deployment guide in developer mode. Installation can be done from the chrome web store once my extension is fully ready and deployed:
First, we navigate to chrome://extensions, then in the upper right corner we toggle the switch on to developer mode.

Figure 2: switch developer mode on

Then we click “load unpacked”, which popups your files, selecting which folder your extension files belong to.

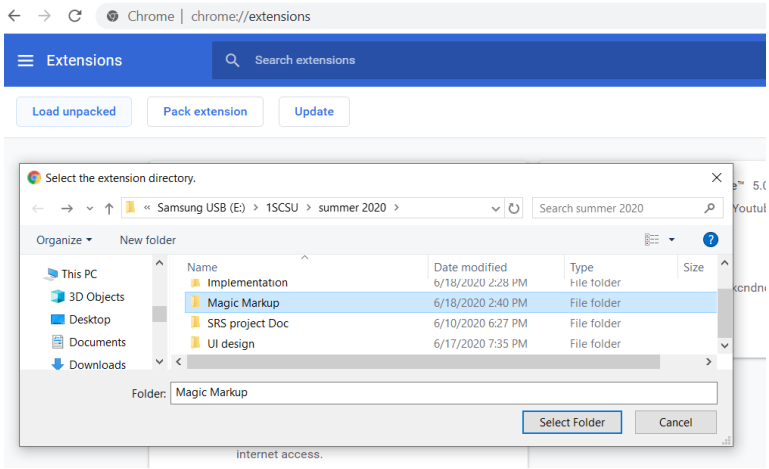


Figure 3: Loading the extension package onto Chrome browser

Finally, after selecting the folder the container appears with the extension name and details. Swtich the toggle to the on(right) position. Once that is done then we click on a new tab and press the extension icon in the upper right hand corner next to the web browser search box and a popup will appear with your extension which loads once clicked.

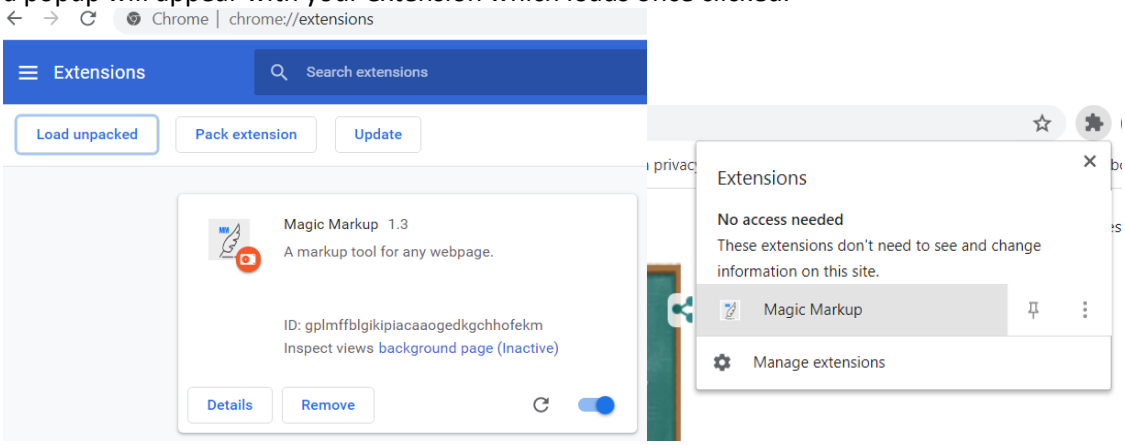


Figure 4: Extension details after loading package(left) and Extension icon installed(right).

Clicking the extension icon loads my user interface system:

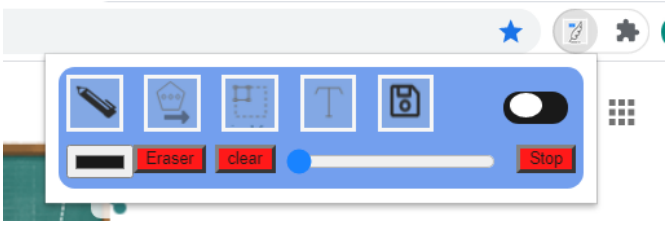


Figure 5: Extension system UI popup after clicking icon

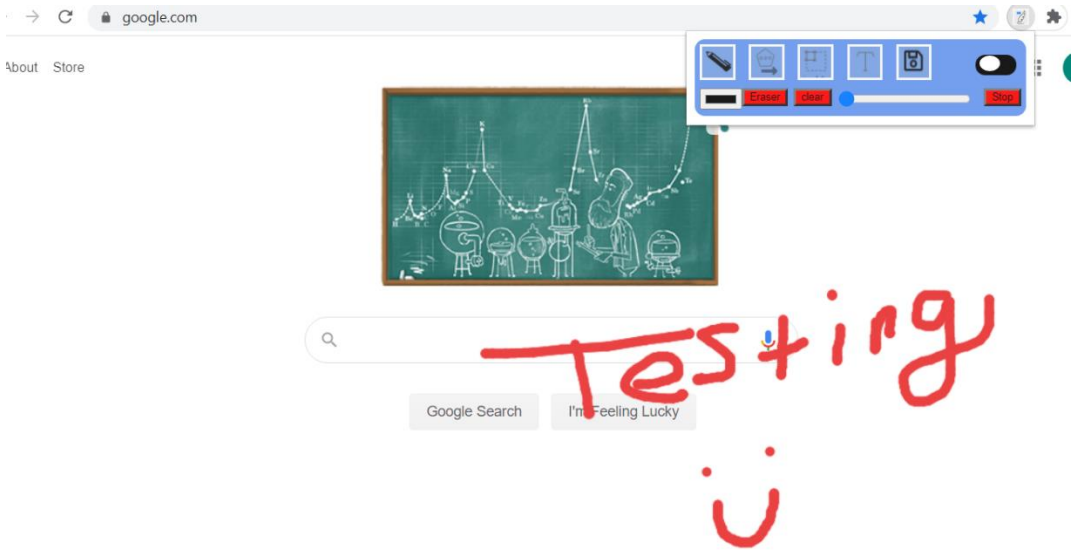


Figure 6: Result after clicking drawing icon feature and drawing over web page

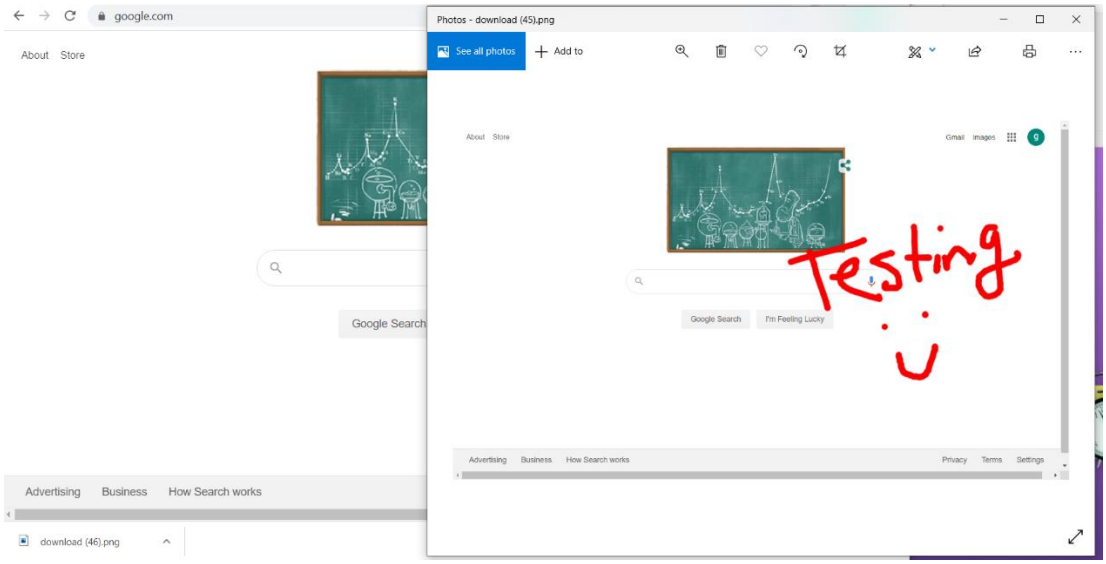


Figure 7: Saving screenshot of web page after clicking screenshot icon

VI. State of the implementation:

The feature completed is the main feature of doing markup on a web page giving the user the ability the draw over any web page as desired. Along with this feature the user can also change the color of the markup pen and the size. Another feature added is the ability to clear the entire drawings on the web page and resume drawing with the previously set markup properties.
A stop feature was added to not only clear the canvas but to stop the drawing function in order to resume browsing the web page making it responsive again.
A screenshot/saving feature was implemented that captures the markup work done on the web page by the user which works as a saving option.

The other features being added are not complete include adding shapes anywhere on the web page, cropping sections of a web page, adding text anywhere on the web page, annotate web pages, and an eraser function which clears certain parts of the markup done on the web page.

VII. Testing and Evaluation:

The testing of my extension system was made possible by browsing through different types of websites that had different contents. This enabled me to find bugs in my system that interfered with certain parts of a web page. For example, navigating websites that had long pages enabled me to make sure that the canvas area was covering and overlaying on top of the entire web page area without any parts cut off. I have not tested on filed uploaded directly through chrome browser yet. For the most part navigating through many websites and testing out my extension system worked well except for one instance where scrolling down a web page that loads new content and further extends the web page causes my system to not read the extended web page size so the canvas cuts off at a certain point. This becomes clear in website such as youtube.com, where scrolling down will load new videos and thus making the web page longer and after a certain height the canvas cuts off and no longer allows markup below it. Aside from the issue mentioned above, the extension system is usable in most web pages.

VIII. Lessons Learned:

Overall, designing the user interface of my system went well although the extension limited how the user interface can respond with the browser such as a limited icon size and what the popup interface can do.
If I were to do this extension again, I would focus less on how many features I want but focus more on a smaller number of features. That way I can spend more time on creating a more efficient feature as opposed to just having multiple features that do not function properly. As a result, I underestimated the work needed to be do not on one main feature.

IX. Version 2:

If I had an addition month or two, I would implement the features I mentioned above: adding shapes to the web page, annotation box, adding text on a web page and cropping sections of a web page. In addition to those features I would also like to make the pen markup for drawing with more customization such as glowing effects and combination colors, and allow

defining words withing a web page on the fly by highlighting a word which popups up its' definition.

X. Professional development/Reflection:

Upon taking this course I knew I wanted to challenge myself more and dive into developing something slightly different from what I have taken in my college courses and choosing this project of developing a google chrome extension definitely met my goals. I had never developed code interacting with the Chrome API which was vital to this project and aside from this I only had a basic understanding of the JavaScript language and JS web API. JavaScript was the primary language used in my project as a result I become more comfortable in using objects and notation such as JSON. Getting a deeper understanding of the JavaScript language taught me to be flexible in my choices for programming with my other languages that I am more comfortable with: Python and Java. Being flexible in more languages opens more opportunities in creating a variety of different projects and being able to read other developers code.

To make this project a success I had to learn Various APIs such as the Chrome API, JavaScript API, Web API, and Canvas API. In doing so I learned the importance of being able to understand and reference API documentations such as its syntax and examples provided as ideas to incorporate into my code. A major challenge was in understanding Chrome API due to limited details and examples which consumed more time than necessary in learning to incorporate my code with it but along came a new skill under my belt. Major supplements to acquiring further knowledge and understanding of a topic was sources such as stackoverflow.com, w3schools.com and developer.mozilla.org. I acquired the skill to spot accurate solutions or examples of certain problems in code because lots of search results can be misleading and waste time.

Unexpectedly, I learned tools along the way such as the chrome browser built-in DevTools which made it much easier to spot errors in my code and debug on the fly so that I wouldn't have to waste time changing the code from my IDE, saving, running it again to test it out. Aside from the DevTools tool the extension manager from the chrome extension site gave an error notification whenever the extension was not working properly. During this process I was able to better understand what errors meant in my code so that I can jump right to it and fix my bugs.

Doing independent research showed me the importance of having a drive for lifelong learning because the knowledge in technology is abundant and managing my time will greatly support this journey in computer science as a developer. Although reading about programming helps, I came to the realization that action triumphs learning. In building a project I acquire new knowledge and skills some of which were not expected, this is the route I will take in deeper understanding of technology.

I look forward to building more Chrome extensions soon and for other browsers such as Firefox and Microsoft edge. I will also move on to bigger projects that involve building web and mobile applications with languages that I am now more comfortable with JavaScript, Python, and Java.

GitHub Repository:

https://github.com/ortega361/Magic-Markup.git