

Criterion C: Development

Techniques Used

The development of the price tracker required many techniques including the ones mentioned during criterion B. I will go over each technique, the rationale, sources (when applicable) and annotated screenshots for evidence.

XMLHttpRequests

The core functionality of the chrome extension relies on the program's ability to actually fetch the information about a product from the Amazon website. Thankfully, since JavaScript is the language being used in the chrome extension there are many different approaches to scraping web pages (retrieving the HTML of a website). I chose to use the XMLHttpRequest method rather than the alternatives due to the fact it has been around for the longest period of time and is relatively easy to understand.

```
36 function scrapePage(url){
37
38   var xhttp = new XMLHttpRequest();
39   xhttp.onreadystatechange = function() {
40     if (this.readyState == 4 && this.status == 200) {
41
42       // Parse using DOM Parser
43       let parser = new DOMParser();
44       let htmlDoc = parser.parseFromString(xhttp.responseText, "text/html");
45
46       // For book listings there are multiple prices.
47       let price = ""
48       try {
49         price = htmlDoc.getElementById('priceblock_ourprice').innerHTML;
50       }
51       catch(err) {
52         try{
53           price = htmlDoc.getElementById('priceblock_dealprice').innerHTML;
54         }
55         catch(err){
56           alert('Sorry I couldn\'t find the price of this product. :C');
57         }
58       }
59
60
61       //remove any dollar signs ($) that may mess with parsing the string to a float.
62       let formatPrice = price.replace("$", "");
63
64       console.log(formatPrice);
65
66       // save new price.
67       saveNew({url:formatPrice});
68     }
69   };
70   xhttp.open("GET", url, true);
71   xhttp.send();
72 }
73
74
```

As can be seen from the screenshot above the an XMLHttpRequest is used to retrieve the HTML of a given URL by sending out a "GET" http request. The callback function "xhttp.onreadystatechange" is used

to specify what happens when a response is given from the server, which the website is located on.

Furthermore, to prevent possible errors with processing a malformed response the program checks to make sure the process from the XMLHttpRequest object has finished by testing the the readystate which if at "4" signifies that the process is completed.

In the same line the program tests to make sure the return HTTP code is 200, which would mean the request has completed. The complete list of HTTP response codes is documented by standard RFC 2616 by the IETF (Internet Engineering Task Force).

DOM Parser

The retrieved HTML however is useless if the program cannot access the relevant information.

Luckily this is possible thanks to the usage of the DOMParser, another built-in JavaScript object. This allows the program select particular elements based on their class, id or tag name as if the JavaScript was modifying the HTML as an embedded script.

```
42 // Parse using DOM Parser
43 let parser = new DOMParser();
44 let htmlDoc = parser.parseFromString(xhr.responseText, "text/html");
45
46 // For book listings there are multiple prices.
47 let price = ""
48 try {
49     price = htmlDoc.getElementById('priceblock_ourprice').innerHTML;
50 }
51 catch(err) {
52     try{
53         price = htmlDoc.getElementById('priceblock_dealprice').innerHTML;
54     }
55     catch(err){
56         alert('Sorry I couldn\'t find the price of this product. :C');
57     }
58 }
59
```

The variable "parser" holds an instance of the DOMParser which is then referenced in the subsequent function calls. In addition a basic algorithm is used to insure that a price is scrapped from the page by using a "try-catch" pair which attempts to execute code and then upon an error executes another block of code rather than pause execution of the script.

This is necessary due to Amazon being inconsistent in the classnames of their prices between both book sales and regular products.

Chrome.storage.sync

Chrome.storage.sync is the built-in function for chrome browsers which in this program is used to store the URLs and previously saved prices of Amazon products. This method stores data in a dictionary format. In which values are unordered and may be referenced by non-integer indexes. The

list of page URLs and prices are stored themselves in their own dictionary which is referenced by the string literal "AmazonURLS". So the expanded data structure may look something like this:

```
{  
  "AmazonURLS":{  
    "amazon.com/bike": "$34.11",  
    "amazon.com/bucket": "$23",  
    "amazon.com/thing": "$99.99"  
  }  
}
```

The sync method was chosen over the more commonly used Chrome.storage.local method as sync allows the data to be accessed from across browsers. Meaning the program will remember products saved on a different chrome browser instance.

```

77 function saveNew(test){
78     // {"urlhere": "18.95"}
79     // only one at a time for now
80
81     chrome.storage.sync.get({"AmazonURLS": {}}, function(data){
82
83         console.log(Object.keys(data.AmazonURLS));
84
85         console.log(Object.keys(test)[0]);
86
87         // save unique urls
88         if(Object.keys(test)[0] in data.AmazonURLS){
89
90             console.log("URL already stored.");
91             for (let key in test){
92                 let tempPrice = test[key];
93                 console.log("Temp price is " + tempPrice);
94                 let tempUrl = key;
95                 console.log("Temp url is " + tempUrl);
96                 comparePrice(tempUrl, tempPrice);
97             }
98
99         } else {
100
101             for(var key in test){
102                 let value = test[key];
103
104                 let newData = data.AmazonURLS;
105
106                 // add the new value to data
107                 newData[key] = value;
108                 chrome.storage.sync.set({"AmazonURLS": newData});
109
110             }
111
112         }
113     });
114 }

```

Chrome.tabs.query

This is another built-in chrome function which returns information relating to the tabs a user has opened in their browser.

```

196
197 // initialize variable to hold the url in this context
198 let tabURL = "";
199
200 chrome.tabs.query({lastFocusedWindow: true, active: true}, function(tabs){
201     // test if url can be retrieved
202     if(tabs[0].url != undefined){
203         console.log(tabs[0].url);
204         tabURL = tabs[0].url;
205
206         // do stuff with this new data
207         if (tabURL.startsWith("https://www.amazon.com/")){
208             console.log("Amazon Link Recieved.");
209             scrapePage(tabURL);
210         } else {
211             console.log("NON-AMAZON LINK.");
212             // return err msg;
213             alert("Sorry this is a non-amazon page so I can't track this product.")
214             port.postMessage("NONAMAZON");
215         }
216     }
217 });
218

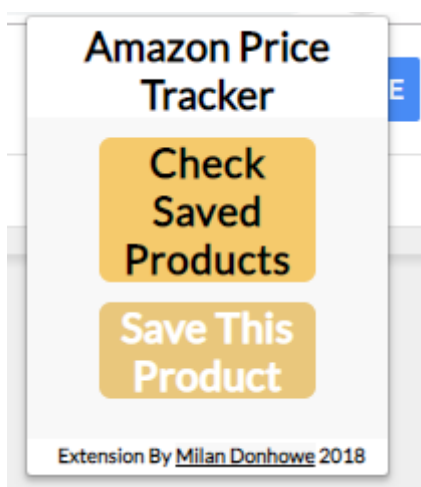
```

It was used to find the url of the user's window. The URL then goes through an if/else statement pair to determine if the url is from Amazon and can have the price data scrapped.

Long Lived Connections

The chrome extension needs to let the background.js script know when to execute various functions based on the user's input in the HTML files. Like when the user presses a button in the extension to save a new product the background script needs to check the tab url, see if the url is an amazon url that go through the algorithm to determine if the price can scrapped and saved OR if the product has already been saved in storage.

The extension achieves this through Long Lived Connections which is a fancy term for having one script msg another script. See screenshots below for reference.



When the button is clicked by user, it triggers an event listener for the javascript file for the HTML file, popup.js, which sends a message to the background.js script that is running in the background.

```
12 window.onload = function(){
13
14     // attach functions to buttons on HTML
15     var loadListen = document.getElementById("load").addEventListener('click', loadUrls);
16     var savelisten = document.getElementById("save").addEventListener('click', savePage);
17
18 }
19
20
21
22 let loadUrls = () => {
23     // load saved urls and iterate through them
24     console.log("loadUrls called");
25 }
26
27 let savePage = () => {
28     // get the current page, check if amazon product and if price is extractable.
29     console.log("savePage called");
30
31     let tabURL = "loading...";
32
33     // ask background.js to do the heavily lifting
34
35     let port = chrome.extension.connect({
36         name: "Get Current Tab URL"
37     });
38
39
40
41     port.postMessage("TABURL");
```

This message is then received in the background script.

```

189 | // GRAB TAB URL
190 | chrome.extension.onConnect.addListener(function(port){
191 |     console.log("Message recieved. Sending response.");
192 |
193 |     port.onMessage.addListener(function(msg){
194 |         if (msg == "TABURL"){
195 |
196 |
197 |             // initialize variable to hold the url in this context
198 |             let tabURL = "";
199 |
200 |             chrome.tabs.query({lastFocusedWindow: true, active: true}, function(tabs){
201 |                 // test if url can be retrieved
202 |                 if(tabs[0].url != undefined){
203 |                     console.log(tabs[0].url);
204 |                     tabURL = tabs[0].url;
205 |
206 |                     // do stuff with this new data
207 |                     if (tabURL.startsWith("https://www.amazon.com/")){
208 |                         console.log("Amazon Link Recieved.");
209 |                         scrapePage(tabURL);
210 |                     } else {
211 |                         console.log("NON-AMAZON LINK.");
212 |                         // return err msg;
213 |                         alert("Sorry this is a non-amazon page so I can't track this product.")
214 |                         port.postMessage("NONAMAZON");
215 |                     }
216 |                 }
217 |             });
218 |
219 |         }
220 |     });
221 | });
222 |
223 | });

```

Regular Expressions

Some regular expression logic is used for displaying the saved product name. Since only the URL and price is stored a simple regular expression was used to extract the product name or part of the name from the URL to be displayed visually for the user.



```

24 // regex recipe:
25 var findName = /(?!<www.amazon.com\/)\w*/
26
27 let name = url.match(findName);
28 let newHTML = "<tbody> " + "<tr><td><a href=" + url + " target='_blank'" + String(name) + "</a></td>" + "<td>" +
29 let listing = document.getElementById('table').innerHTML += newHTML;

```

innerHTML

Used both in DOMParser to retrieve the HTML data from amazon websites and used to dynamically change the markup of the extension page. The evidence for the usage can be seen in the above images.

HTML/CSS files

The entire user interface for Chrome Extensions is built off of HTML, (Hypertext Markup Language) and CSS (Cascading Style Sheet) which are the primary technologies used for visual web development. This was helpful as there was plentiful documentation on the usage but also because the technologies themselves are designed specifically for being user interfaces.

The popup.html is the first and main visual file loaded which has the elements of the page organized.


```

1  <!DOCTYPE HTML>
2
3  <html>
4
5  <head>
6      <title>Amazon Price Tracker</title>
7      <link type="text/css" rel="stylesheet" href="popup.css">
8      <link href="https://fonts.googleapis.com/css?family=Lato" rel="stylesheet">
9  </head>
10
11 <body>
12
13     <h1>Amazon Price Tracker</h1>
14
15     <a href="products.html"><button class="btn" id="load">Check Saved Products</button></a>
16     <button class="btn" id="save">Save This Product</button>
17
18
19     <script src="popup.js"></script>
20
21     <h2>Extension By <a href="https://github.com/MilanDonhowe" target="_blank">Milan Donhowe</a> 2018</h2>
22
23 </body>
24
25 </html>

```

The CSS file specifies parameters for the rendering engine of the chrome browser. This includes the font-size, color of text, background color and other visual aspects.


```

12  =====
13  = Page-Wide Styles =
14  =====
15  */
16
17  * {
18      font-family: 'Lato', sans-serif;
19      font-size:130%;
20      text-align:center;
21      font-weight: bold;
22      color: #000;
23      /*background-color: hsl(61, 100%, 50%);background-color: rgba(244, 188, 66, 0.8);*/
24      background-color: #f7f7f7;
25      padding:0px;
26      margin:0px;
27      /*was 180px*/
28      width:180px;
29  }
30  /*
31  =====
32  = Button Styles =
33  =====
34  */
35  .btn{
36      border: 0px;
37      border-top:0px solid black dashed;
38      border-bottom:0px solid black dashed;
39      margin-top:10px;
40      width:60%;
41      background-color: rgba(244, 188, 66, 0.8);
42      border-radius:6px;
43      text-align:center;
44      color: #000;
45      letter-spacing:0.02em;
46  }
47  .btn:hover{
48      background: #e8c476;
49      color: #fff;
50  }
51

```

Manifest.json This is the configuration file for the chrome extension. Specifies various parameters like the icon and

permissions of extension.

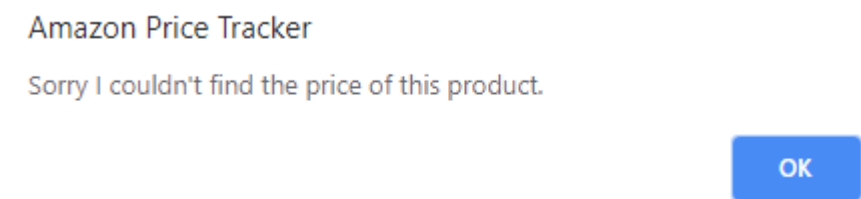


```
1 {
2   "name": "Amazon Price Tracker",
3   "version": "1.0",
4   "description": "Tracks Amazon Product Prices",
5   "manifest_version": 2,
6   "background": {
7     "persistent": false,
8     "scripts": ["background.js"]
9   },
10  "browser_action": {
11    "default_popup": "popup.html",
12    "default_title": "Track Amazon Products!"
13  },
14  "icons": {
15    "32": "images/logoA32.png",
16    "48": "images/logoA48.png",
17    "128": "images/logoA128.png"
18  },
19  "permissions": [
20    "storage",
21    "tabs",
22    "<all_urls>"
23  ]
24 }
```

alert()

A common JavaScript function which is used to notify user of errors in the extension or new deals detected.

```
55 catch(err){
56   alert('Sorry I couldn\'t find the price of this product.');
```



Works Cited:

Browser Statistics, www.w3schools.com/jsref/jsref_obj_regexp.asp.

“Chrome.storage.” *Chrome: Developer*, developer.chrome.com/extensions/storage.

“How to Fetch URL of Current Tab in My Chrome Extension Using Javascript.” *Stack Overflow*, stackoverflow.com/questions/18436245/how-to-fetch-url-of-current-tab-in-my-chrome-extension-using-javascript.

“Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.” *IETF Tools*, tools.ietf.org/html/rfc7231#section-6.

“Parse an HTML String with JS.” *Stack Overflow*, stackoverflow.com/questions/10585029/parse-an-html-string-with-js.

“XMLHttpRequest.readyState.” *MDN Web Docs*, developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/readyState.