Streamlining Book Requests with Chrome

Abstract

This article starts by examining why a Chrome Extension was desired and how we saw it making the workflow for requesting new items both easier and more accurate. We then go on to outline how we constructed our extension, looking at the folder structure, third party scripts and services that combine to make this achievable. Finally, the article looks at how the extension is regulated and plans for future development.

By Dr. Rachel Schulkins and Joseph Schulkins

The Chrome Extension – an Introduction

The idea for the Book Request extension developed after a team meeting, in which several possibilities were raised to help standardise the format of book requests reaching Acquisitions from in-house colleagues, primarily the Academic Liaison. The problem was that there was no clear format for sending book requests which resulted in disparate and insufficient information for purchasing. Some of the emails reaching Acquisitions contained unnecessary verbalism whereas others failed to provide basic information, all of which hindered the process. To address these, Acquisitions needed a tool that would be able to provide us with all the information we needed to process requests as efficiently and quickly as possible, while also speeding the process for the Librarians submitting the request. The initial idea was for librarians to fill out a PDF form and email it as an attachment to Acquisitions. We concluded the meeting with the intention of looking into replacing the PDF with an online form. But it occurred to us that even the idea of an online form can be improved on. Why not create some sort of an online mechanism that would automatically gather a book's metadata, search that book against our current Library holdings, and finally send the book details to the Acquisitions Team to purchase, thus making the process more efficient and accurate .

We developed the Chrome extension to specifically work with amazon.co.uk. We selected amazon.co.uk because Amazon lists the majority of books published worldwide. Whether it is an out of print book or a recently published manuscript, Amazon will likely have the book's metadata which can then be imported using the extension. The book request process established with the extension is quite simple. Once a librarian downloads the extension onto his/her Chrome Browser, the extension becomes a primary point of contact for submitting requests. When a librarian wishes to submit a book request, s/he searches amazon.co.uk for the book. Once the librarian locates the book, s/he clicks on the extension, which then gathers the book's metadata and automatically populates the book request form with the following details: Title, Author, ISBN, Edition, Publisher, Year of publication. This leaves the librarian with the responsibility of providing the following criteria: format, Department/Fund, Location, Requesting Party, and reservation details. The need for typing in any of the information is minimal as the majority of the fields are made up of dropdown menus.

Though designed with amazon.co.uk in mind, the extension is compatible with other online bookstores such as abe.com and Blackwell's (http://bookshop.blackwell.co.uk/) and it can even extract some data from publishers' websites. The disadvantage of working with a different website other than amazon.co.uk is that the extension cannot extract the books' metadata in a single click, which means the librarians will have to fill the form manually. Nevertheless, the fact that the request form embedded in the extension is a mouse click away makes it easily accessible to users. Another major advantage of working with the extension is related to its ability to check the book's ISBN against existing stock items in the Library, prior to purchasing. Whereas in the past, the librarians had to manually check every book request reaching their inbox against our library holdings, this task is now being performed by the extension. The librarians see a message alert notifying them whether a book is already available in the library or whether the extension could not perform the check (this is usually happens when the extension is used on websites other than amazon.co.uk). The same message is displayed in the email sent to Acquisitions, who can then follow the necessary process to complete the request depending on the book's status. This has proven to be a great time saver for librarians, who now have a single point of contact where they can perform all the necessary steps prior to submitting a book request: check book requests against existing holdings; extract the books' metadata and populate the book request; submitting their request to Acquisitions.

Getting the librarians to test the extension was quite easy as they could see its immediate benefits. Within three months the extension became the main tool for submitting book request across the library. It was adopted by the Academic Liaison Team, Collection Management as well as Customer Service Team. Having the majority of book requests reaching Acquisitions go through the extension allows for important data to be collected. Prior to the extension there was no way of monitoring requests live. Instead, we relied on our Library Management Systems to run lists a few time a week according to different variables. Now, the extension collects and analyzes incoming data live. Every request submitted using the extension is stored on the Wufoo data page and the information gathered is automatically analyzed and presented in various charts and graphs to reflect fund usage, location of material (across several sites on campus) and book format (print or ebook). The data gathered can then be used to analyze efficiency, best practices and to monitor trends across the academic year. For example, the data allows one to check the amount of requests submitted in a given time and to see how many orders were placed in the same amount of time. It allows us to see the time gap between request, execution and reception of material, and adjust and amend departmental procedures to maximize productivity and improve customer satisfaction.

Design and Implementation

A Chrome Extension is essentially a collection of Javascript, CSS and HTML that you can add to the Chrome browser. These little pieces of software allow you to add functionality or change the way Chrome behaves. Extensions published to the Chrome store can then be downloaded by either a select few that the developer allows or by the wider Chrome community. Given that the only requirements are an understanding of JavaScript, CSS, HTML and the subtleties of the development process, an extension can be manufactured relatively quickly and painlessly.

That said, the hardest part in developing the extension was the flow of communication between the extension and the open window; however the official documentation[1] covers this well as do numerous excellent tutorials. Indeed, the official documentation and the tutorial by Gabe Berke-Williams 'How To Make a Chrome Extension'[2] provided the basis upon which I developed our extension.

Our extension consists of:
manifest.json — this has the settings of the extension such as location of background scripts and the default actions of the extension
src/bg/background.js — used to handle messages returned from the active page to the extension:

```
/**
 * A background listener waiting for the extension to be triggered
 * @param  object message  sent by src/inject/inject.js
 */

chrome.runtime.onMessage.addListener(function(message,sender,sendResponse){

  /**
  * Set bibdetails as result of processing the returned message
  * the dataProcess function uses regular expressions to
  * chop up the message into the relevant bibliographic
```

[1] What are extensions? [Internet]: Google [cited 2015 03/15]. Available from: https://developer.chrome.com/extensions
[2] How To Make a Chrome Extension [Internet]: Berke-Williams, Gabe [cited 2015 03/15]. Available from: https://robots.thoughtbot.com/how-to-make-a-chrome-extension

```
  * parts
  * @type object
  */
  var bibDetails = dataProcess(message);

  /**
  *  Function to check our catalogue against the submitted
  *  bibliographic details and then populate the form using
  *  the bibDetails object
  */
  checkCat(bibDetails);

});

/**
 * Insert the inject scripts into the current window
 */
chrome.tabs.executeScript(null, {file:"/src/inject/jquery.min.js"});
chrome.tabs.executeScript(null, {file:"/src/inject/inject.js"});
```

Figure 1. A snippet of our background.js showing how it injects scripts and handles the returned message

src/browser_action/browser_action.html — this html page controls what the user sees when they click on the extension and provides links to JavaScript and CSS resources that are to be used to process the data retrieved from the active page
src/inject/inject.js — used to communicate between the extension and the active page using Chrome’s sendMessage method:

```
/**
* Set some basic variables to capture, taking a broad stab at where the
* title is located.
*/

var title, url, pub, isbn;
title = jQuery('title').text();
pub = jQuery('#detail_bullets_id ul li:contains("Publisher")').text();
isbn = jQuery('#detail_bullets_id ul li:contains("ISBN-13")').text();

/**
* Get the current URL. Typically this will be amazon.co.uk, but having this means users
* can request from elsewhere and someone in our Acquisitions team
* can go and view the item directly
*/

url = window.location.href;

/**
* Send the details from the webpage to the handling script to populate the form
*/

chrome.runtime.sendMessage({method:'setTitle', title: title, url: url, pub: pub, isbn:
isbn});
```

Figure 2. The inject script that sends information about the current page back to the extension. This is triggered

when the extension button is clicked.

src/inject/jquery.min.js — to add jQuery functionality (if it doesn't exist) to the active page making it easier to access elements
css directory — has the bootstrap and alpacajs css files
js directory — has AlpacaJS[3] (a JavaScript form which provides excellent functionality), Bootstrap, isbnjs[4] (a JavaScript library that validates isbns and allows conversion between 10-13 digit)
fonts directory — contains Bootstrap glyphicons
icons directory — contains png's of the plugin's icon at various sizes to be used by the browser

A core part of the extension is the form it uses to capture the metadata and crucially we would then need to find a way to store any of the data captured by the form, develop some statistical reporting and set up email alerts of when new items were submitted. Whilst other tools may be available, having previously used Wufoo[5] forms Joseph knew that it could handle all of this quite comfortably. When we created the form we did so exactly as we wanted it for the Chrome extension with an idea that the form would be embedded. But as it turned out this approach was impractical and instead we made use of the Wufoo API.

Knowing that we would use Wufoo via their API, we settled on AlpacaJS as a suitable front end alternative for the form and it would be this that would be filled in when the user clicked on the extension. AlpacaJS uses jQuery[6] and JSON to arrange the form elements and is extremely extensible, not only does it lend itself well to validation and conditional processing, but it will allow you to define custom functions and callbacks. Whilst AlpacaJS is good in that it allows for great control of the form, there are a few pinch points that Joseph encountered when using it and the official documentation was not very helpful. In the end it took several hours combing sites such as StackOverflow[7] to find help or solutions that Joseph could work into our form.

So with the front end form constructed and the Wufoo form in place to receive

---

[3] Alpaca - Easy Forms for jQuery [Internet]: Gitana Software, Inc. [cited 2015 03/18]. Available from: http://www.alpacajs.org/
[4] isbnjs: An ISBN JavaScript Library [Internet]. Google Project Hosting [cited 2015 04/09]. Available from: https://code.google.com/p/isbnjs/
[5] Wufoo [Internet]: Wufoo [cited 2015 04/15]. Available from: https://www.wufoo.com/
[6] jQuery [Internet]: jQuery [cited 2015 04/10]. Available from: http://jquery.com/
[7] alpaca javascript dom update [Internet]: StackOverflow [cited 2015 06/02]. Available from: http://stackoverflow.com/questions/24146694/alpaca-javascript-dom-update

entries, we needed something that would be able to send the information to the Wufoo API and to receive back any response. As such we pointed the customised AlpacaJS form to a PHP script. This script has our API key, API endpoint and maps each entry from the form to a variable and, using CURL, this information is sent to Wufoo. If a 'success' message is received by the CURL function, then the PHP script displays a success message in the Chrome Plugin, however should there be an error, the information regarding the CURL error and the attempted values would be sent to Joseph with a message saying that there had been a problem displayed to the user.

Before the plugin, it was standard practice for those both requesting and processing the requests to check each item in the catalogue to see if we already had a copy. If the item was found to be in the catalogue and the person requesting the item hadn't checked or indicated the reasons why extra copies were needed, then this would spark a back and forth conversation (usually over email). As such Joseph was asked whether an automatic catalog check could be built into the form. Again, using PHP Joseph was able to create a quick and simple script that received the ISBN as a get request and queried our mobile catalogue for results. If the search failed (i.e. nothing with that ISBN was found) an html page would be loaded that had content wrapped in a class of 'errorArea' and using DOMXpath() (since getElementsByClassName doesn't exist in DOMDocument), the script would be able to see whether the checked ISBN existed or not. With the existence of this item now set to either 'true' or 'false', the PHP script would return this as JSON so that the extension could parse it. The extension would then inform the user that the item already existed and append this information to the notes field. Because of this all parties are aware if there is an item being requested that is already in the catalogue and it allows the person submitting the request to add any additional comments. This early communication helps eliminate the unnecessary email conversations. One great piece of feedback we have received about this particular feature was from a librarian who whilst placing a request was alerted that we already had an item that he was ordering in stock. Since he was fairly sure nothing with this title or authors existed in our catalogue he searched and found that we did have this particular item already, but that the title had changed slightly and so too had the authors.

```
/**
* An ISBN is sent as a Get request to this script and we will return JSON
*/
header('Content-Type: application/json');

$isbn = isset($_GET['isbn'])?$_GET['isbn']:'error';

/**
* Did we capture the ISBN from the webpage or has it been set to false
*/
if($isbn == 'error' || $isbn == false){
        echo '{"exists":"error"}';
```

```
    }
    else{

        /**
        * Our mobile catalog will be used to search against as
        * this has less styling elements to navigate
        */
        $url = 'http://m.library.liv.ac.uk/search/?searchtype=Y&searcharg='.$isbn;

        /**
        * Initialize a CURL object
        */
        $ch = curl_init();

        /**
        * Set url and other options
        */
        curl_setopt($ch, CURLOPT_URL, $url);
        curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);

        /**
        * Get the page contents
        */
        $output = curl_exec($ch);

        /**
        * Close curl resource to free up system resources
        */
        curl_close($ch);

        /**
        * Create a DOM parser object
        */
        $dom = new DOMDocument();

        /**
        * The @ before the method call suppresses any warnings that
        * loadHTML might throw because of invalid HTML in the page.
        */
        @$dom->loadHTML($output);

        /**
        * getElementsByClassName doesn't exist in DOMDocument so use xpath
        * to find the error section if applicable
        */
        $xpath = new DOMXpath($dom);
        $error_area = $xpath->query('//p[@class="errorArea"]');

        /**
        * If there is an errorArea, then we don't have the item
        */
        if($error_area->length > 0) {
            $exist = "false";
        }
        else {
            $exist = "true";

        }

        /*
        * return the exist variable wrapped in JSON
        */
        echo '{"exists":"'.$exist.'"}';
}
```

Figure 3. Our catalog lookup script

From the start we wanted this extension to behave like any other on Chrome in that if/when updates were needed, we wouldn't have to ask those who had installed it to re-install a new version (best case scenario) or have to re-install the new version ourselves on everyone's Chrome (worst case scenario). The only way to have an extension that updates automatically is to add it to the Chrome Web Store. This has been a security change to prevent extensions from loading malicious software into the browser. You can still load extensions (albeit with the caveat that they won't update) into your version of Chrome by enabling 'Developer Mode' on the extensions page of Chrome (chrome://extensions), and this will enable you to upload a zip file of your extension. To publish to the webstore[8] you will need to have an account and pay a one time fee.

Before publishing though, Joseph found that we could set the extension as private (meaning it wouldn't be listed in the web store) but we could also either a) specify individuals to grant access to (an individual Gmail account would be required) or b) allow access to a group of users in Google Groups. For our situation the second one seemed ideal as it would mean we could administer access to the group independently of the extension and, since not all our staff have Google accounts, provide an ingress to the extension via their University email.

The submission process is relatively straightforward, first requiring you to upload an extension as a zip and then through a form supplying information about the plugin and its audience. Once you have completed these steps and selected to publish to your private group/individuals, you are redirected to the landing page of your extension. Copy the URL as this is what you will need to send to your testers so they can load the extension. The extension will now take an hour or so to become available for download. One important thing to note, when you are creating the zip either for the first time or for subsequent updates, you will need to make sure that the folder is zipped without any extra or hidden files and folders as these can sometimes cause errors loading or running the extension.

Missing conclusion.

Conclusion

[8] Publishing Your App [Internet]: Google [cited 2015 06/09]. Available from: https://developer.chrome.com/webstore/publish

To conclude, the extension has proven to be popular and its implementation a success for standardising book requests reaching the Acquisitions department. At the moment, it is an in-house tool to allow departments in the library to communicate their requests to the Acquisitions Team. However, the future objective is to develop the extension for students and academic staff outside the library. In the next academic year we will be developing the extension for other browsers and also explore the possibilities of employing the idea to develop a mobile app, to help enhance user experience and the ease of accessibility to the material staff and student require for their research.

Further, we will look to create specific webpage filters that will match the requesting url and adopt the appropriate filter to broaden the range of pages the extension can capture data from. In particular we will look at additional booksellers and high use publishers. Whilst to simplify the ordering process we will add to the notification emails to the Acquisitions Team, links to our suppliers formed of OpenURLs and the captured ISBN(s) to directly link to the requested item.

About the Authors:
Dr. Rachel Schulkins holds a Phd in English Literature from the University of Liverpool and currently works as a Principal Library Assistant.
Joseph Schulkins (joesch@liv.ac.uk) has been the Systems Librarian at the University of Liverpool since 2007.