A Method for Improving the Usability of Web Browsers by Automating Information Retrieval

Drake Deaton

MSc Dissertation

Date: 17/09/2019

Student ID: C1873019

Supervisor: Dr Padraig Corcoran

Degree: Computing & IT Management

Institution: Cardiff University, School of Computer Science & Informatics

Table of Contents

1 Abstract

Excessive tab use by users, and the ever-expanding amount of content online, raises the concern that perhaps users can't consume content via their web browser as efficiently as they should be able to. As part of a solution to this concern, this project presents a browser extension built with the aim of improving the usability of web browsers by creating a method for automating the retrieval supplementary information to geographical content found on web pages. With this solution, the amount of navigation that users have to perform to retrieve content across different web pages is reduced, and thus the efficiency of using the web browser is improved.

In this report, after discussing some relevant background material (browser extensions, usability, NLP, and IR), the project's requirements, assumptions, constraints and considerations (development strategy, business concerns, and software utilized,) are delineated. Then, the project's timeline and the software developed throughout each of its phases are explained, and finally, concluding reviews of the software architecture, project testing, and project reflections, are discussed.

Ultimately, the usability testing for the extension suggested that the functionality of this software does potentially have merit, but that more engineering is required on this software in order for it to reach its initial aim of improving web browser usability. However, if the appropriate software revisions are made, a future version of this software could be successful in improving web browser usability.

2 Project Introduction

For decades web browsers have played a significant role in how people interact with the internet. Whether for accessing a website, retrieving information, or using a web app, it is through web browsers that many people reach the internet services that they're interested in. Given this close relationship between web browsers and the internet, improving the usability of web browsers should therefore largely improve people's ability to interact with internet content. Furthermore, because over half the world's population

now uses the internet (BBC, 2015), even an incremental improvement in web browser usability has the potential to affect many people, and as such, it's important that web browsers continue to improve their usability.

Usability can be defined as, "the effectiveness, efficiency and satisfaction with which … users [can] achieve specified goals…" (ISO, 2002); how learnable, how error-free, how memorable – and ultimately, how useable, the particular method for using something is. Good usability can drive a product's success (such as the success of the Google Chrome browser over its competitors[1] – a success that can be largely attributed to the improvements[2] in usability that Google's browser developed over its rivals) and bad usability can harm a product's success (such as the loss of the company Wesabe[3] to their rival, Mint – a loss Wesabe's founder partly attributed to the superior usability of their rival's product (Hedlund, 2010)). This correlation between software usability and business performance, shows just how important consumers find usability.

In terms of this project, the emphasis is on observing usability from an 'efficiency' standpoint, that is to say, how efficiently web browsers enable users to perform tasks. In their modern incarnations, web browsers could be said to already possess good efficiency, but as this project shows, there are methods through which their efficiency could be improved. In specific, browsers could be doing more to reduce how frequently users have to perform additional navigation to retrieve supplementary content. For example, when users are reading content on a web page, in order to find supplementary content related to something they've read, users often end up opening several additional tabs in order to view this content. If web browsers provided tools that allowed users to retrieve supplementary information without leaving their current web page, this would reduce the need for users to open additional tabs. Excessive tab use is, in fact, a common problem amongst many web browser users, as demonstrated recently in a study of 10,000 Firefox users monitored over a three-month period. In this study, over 25% of participants used on average twenty or more tabs at once, with close to 50% using at least four tabs or more at once (Mozilla,

[1] Google Chrome has a global market share of desktop internet browsers of over 70% (StatCounter, 2019a).
[2] For more on Chrome's original improvements in usability, see the section: 'Usability In Web Browsers' (section 3.2.1).
[3] Wesabe was a personal finance management company.

2017). If this study is representative of browser users generally, then it can be inferred that a method for reducing excessive tab use in web browsers could greatly impact how efficiently a large percentage of browser users are retrieving content on the internet. A method for reducing how frequently users have to perform additional navigation is therefore an improvement to browser usability that's worth investigating.

This project will investigate such a method by creating a web browser extension that allows users to retrieve supplementary content without leaving their current web page. More specifically, this extension will allow users to retrieve supplementary information about text on a web page that pertains to geographical locations. To better illustrate this, take the following scenario (fig. 1):

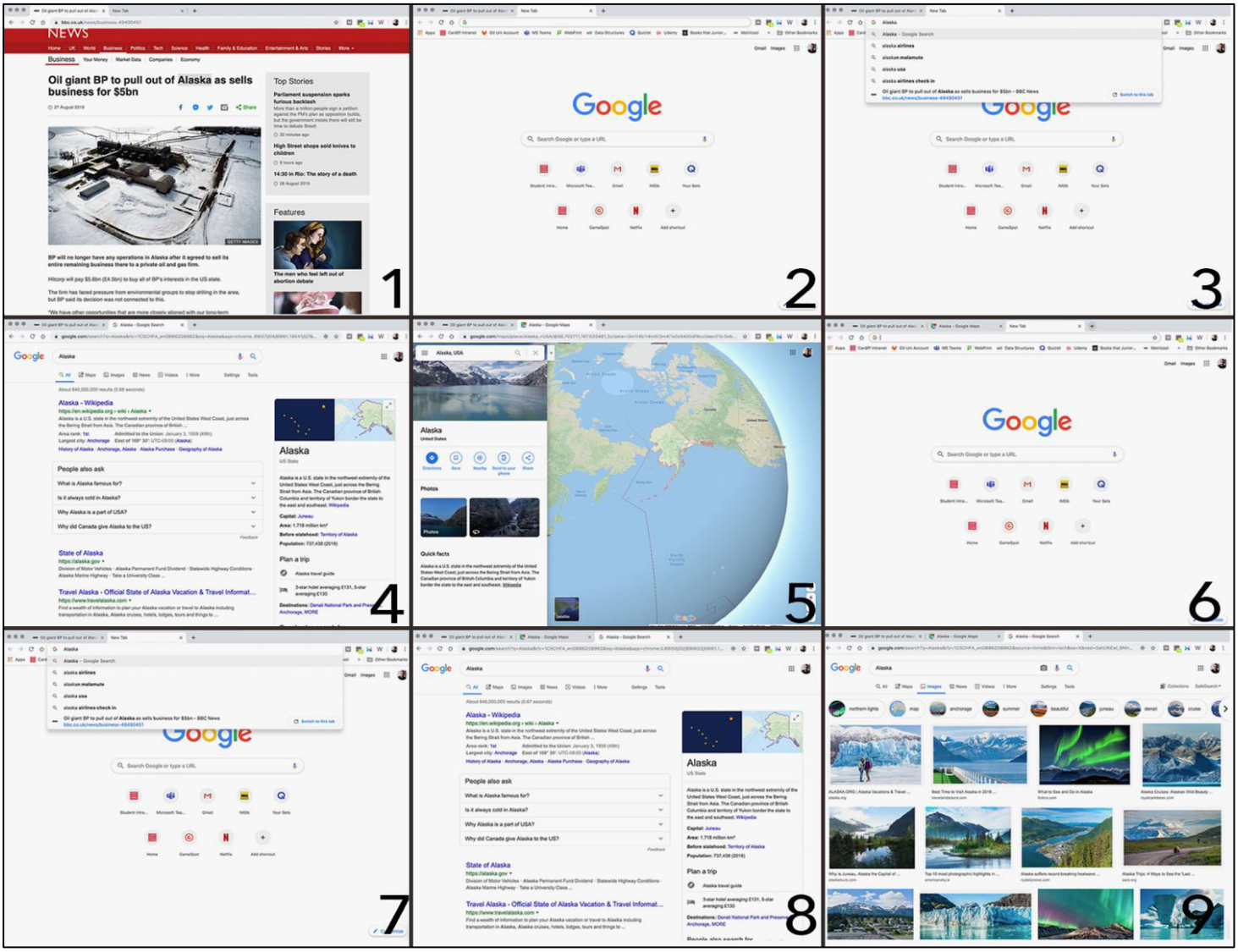


Figure 1: Illustration of how looking up supplementary information to specific text read on a web page is traditionally performed. 1: A user reads some geographical text on a web page. 2: The user opens a second tab. 3: The user types in the location's name. 4: The user runs a search. 5: The user opens a map to see where the location is. 6: The user opens a third tab. 7: The user again types in the location's name. 8: The user runs another search. 9: The user navigates to some images from the location to see what the location looks like.

1. A user reads some geographical text on a web page (fig. 1.1).
2. The user opens a second tab (fig. 1.2).
3. The user types in the location's name (fig. 1.3).
4. The user runs a search (fig. 1.4).
5. The user opens a map to see where the location is (fig. 1.5).
6. The user opens a third tab (fig. 1.6).
7. The user again types in location's name (fig. 1.7).
8. The user runs another search (fig. 1.8).
9. The user navigates to some images from the location to see what the location looks like (fig 1.9).

With the extension built for this project, using a web browser to accomplish this scenario becomes much simpler (fig. 2):

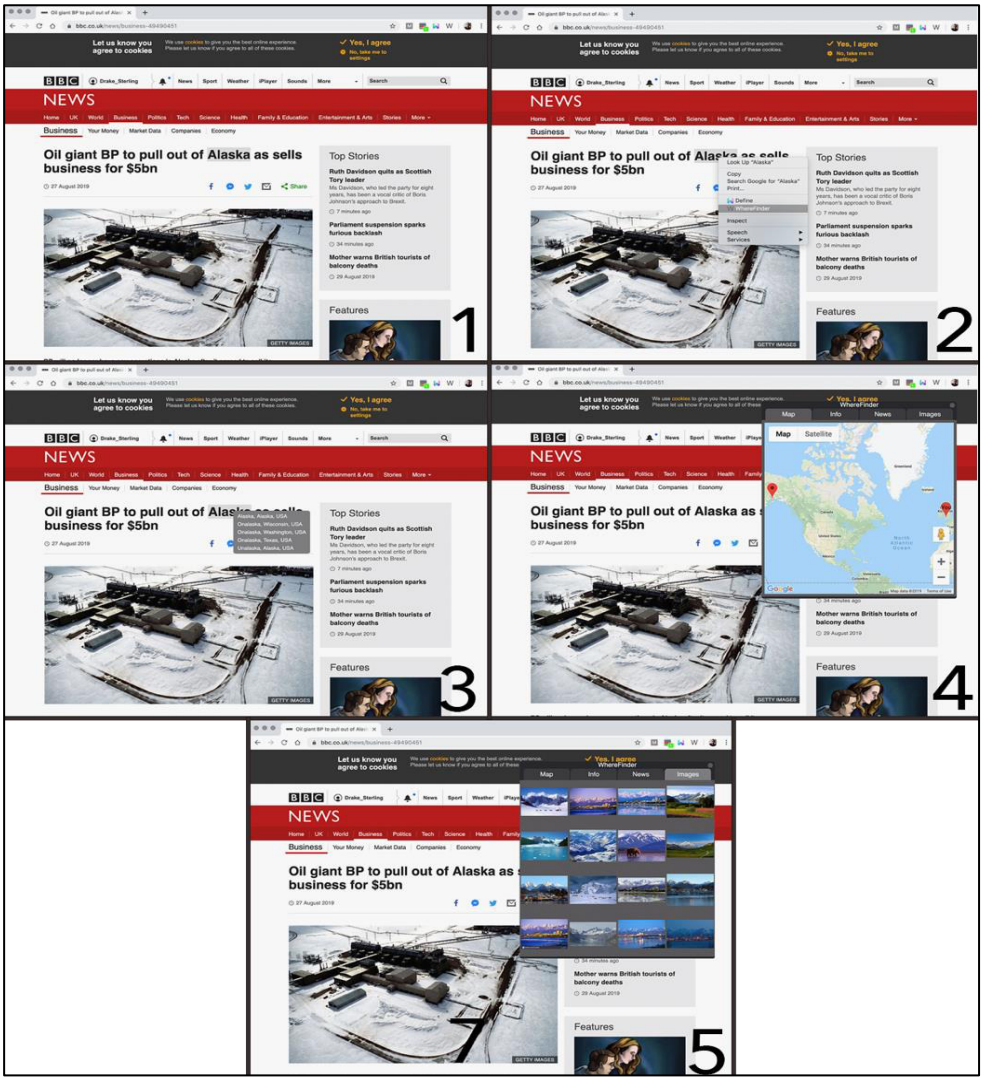


Figure 2: Illustration of how looking up supplementary information to specific text read on a web page is performed using this extension built for this project, WhereFinder. 1: A user reads some geographical text on a web page. 2: The user highlights the text, opens the browser's context menu, and clicks on the 'WhereFinder' extension. 3: The extension creates a second context menu with a list of all the locations that match the input text. 4: The user selects an option from the context menu, and the 'WhereFinder' display opens, displaying a map showing the location of the place. 5: The user can switch tabs within the WhereFinder display to find more supplementary information, such as related images, local new articles, or the place's Wikipedia page.

1. A user reads some geographical text on a web page. (fig. 2.1)
2. The user highlights the text, opens the browser's context menu, and clicks on the 'WhereFinder' extension. (fig. 2.2)
3. The extension creates a second context menu with a list of all the locations that match the input text. (fig. 2.3)
4. The user selects an option from the context menu, and the 'WhereFinder' display opens, displaying a map showing the location of the place. (fig. 2.4)
5. The user can switch tabs within the WhereFinder display to find more supplementary information, such as related images, local new articles, or the place's Wikipedia page. (fig. 2.5)

What makes the latter approach superior is that it 'pulls' content to the user, rather than requiring the user to navigate to the information – a change which simplifies the process of retrieving the information for the user by enabling the user to remain on the web page they were originally reading.

Hereafter, the rest of this report will offer a multitude of lenses through which this project can be better understood. The topics that will be discussed, include:

- Project Context[4]: An exploration of useful background reading for understanding this project.
- Project Requirements[5]: A review of the functional and non-functional requirements of the software.
- Project Assumptions & Constraints[6]: A review of the limitations pre-emptively placed on this project.
- Project Approach[7]: An appraisal of this project's:
    - Development Strategy Considerations[8]: The strategies considered for this project.
    - Business Considerations[9]: The 'Risks[10]' and 'Disbenefits[11]' of this project.
    - Software Considerations[12]: The software used in this project

[4] Section 3.
[5] Section 4.
[6] Section 5.
[7] Section 6.
[8] Section 6.1.
[9] Section 6.2.
[10] Section 6.2.1.
[11] Section 6.2.2.
[12] Section 6.3.

- Project Timeline[13]: An overview of the project's timeline, including a 'Timeline Breakdown[14]' that offers a lower-level look at the functionality built at each phase of the development schedule.
- Project Architecture[15]: An overview of the architecture of the software, including a UML Diagram[16].
- Project Testing[17]: A breakdown of testing done on the project, including 'Software Testing[18]', and 'Usability Testing[19]'.
- Project Conclusion[20]: A summation of the project, including discussions on 'Reflections[21]' about the work that's been done on this project, and 'Recommendations[22]' for future work.

2.1 Project Aim & Objectives

Aim:

Develop a browser extension that improves the usability of web browsers by creating a method for users to retrieve supplementary content about geographical locations on web pages more efficiently.

Objectives:

1. Process input text with natural language processing (NLP) to determine/isolate geographical content.
2. Match input text against a database of geographical locations.
3. Use input text to retrieve contextual information, including:
   3.a. Geospatial data (maps)
   3.b. Wikipedia data
   3.c. News articles
   3.d. Images
4. Build a user interface (UI) that displays the results from the input text at the browser-level.

[13] Section 7.
[14] Section 7.1.
[15] Section 8.
[16] Section 8.1.
[17] Section 9.
[18] Section 9.1.
[19] Section 9.2.
[20] Section 10.
[21] Section 10.1
[22] Section 10.2.

2.2 Project Benefits

Benefits to users include:

1. Minimising the need for users to open separate tabs to look up supplementary content.
2. Reducing the number of steps it takes users to gather more information about content they encounter on web pages.
3. Improving the user experience (UX) of web browsers.

Benefits to the Computer Science field include:

1. The demonstration of an improvement in web-browser usability.
2. The demonstration of service composition.
3. The demonstration of a method of automating information retrieval across multiple content sources.
4. The demonstration of a browser extension that combines NLP, machine learning, databases, and several information retrieval (IR) systems, to create a single comprehensive IR system

2.3 Target Market

This product should appeal to users of browser extensions more than non-users of browser extensions simply because the former are the types of consumers who typically engage with this type of software. All major browsers support extensions and are continuing to develop their capabilities. Google, for instance, have recently announced a new version of their Chrome extensions, 'Manifest v3', launching this year (Google, 2018). The continued support of extensions provides evidence that the companies behind the world's leading web browsers (Google, Mozilla, Apple, Opera Software, and Microsoft) believe there to be continued value in the technology, which suggests that new products developed for the browser extension market have the potential to be valuable themselves.

In terms of market size, as of 2019, the number of extensions hosted on the Chrome Web Store has surpassed 180,000 (Google, 2018), and coincidentally, on Mozilla's Firefox browser there are over 180,000 extension downloads per day (Mozilla, 2019). Within this market, the most successful extensions have over

ten-million people using their services[23]. These statistics suggest that a new product has the potential to find a sizeable user base, if it can demonstrate value to users. The improvements to browser usability that this extension project aims to facilitate may create such value to a sub-set of extension users, and if so, this extension could find a healthy corner of the extension market for itself to occupy.

2.4 Competition

Similar browser extensions to the one this project aims to build are in existence. There is one similar product available on Firefox[24], and on Chrome there are in fact several[25], all of which have around 20,000 users. However, this project will be different from these competitors (table. 1) in a number of important ways:

Competitor's Functionality	This Project's Functionality
Competitor extensions open retrieved maps information in a separate tab[26].	This extension opens retrieved content within the user's current tab.
Besides map displays, no additional information (i.e. Wikipedia, news, images) is retrieved.	This extension retrieves a variety of supplementary content (maps, news, images, Wikipedia).
Competitors fail to disambiguate input text thoroughly, meaning the maps they retrieve only display the first match they find. This is an issue because many locations have the same name[27].	This extension disambiguates locations by leveraging a location database to find all potential locations that a query might match and then allowing the user to select which specific match they're interested in.

Table 1: Comparison between competitor's functionality versus this project's functionality.

[23] AdBlock (Google, 2019a), Adobe Acrobat (Google, 2019b), Avast SafePrice (Google, 2019c).
[24] 'Search on Google Maps' (Mozilla, 2019b).
[25] 'Google Maps select and search' (Google 2019d), 'Open in Google Maps' (Google, 2019e), 'send to google maps' (Google, 2019f), 'Quick Maps' (Google, 2019g).
[26] It's worth noting that one competitor, 'Quick Maps', does open a map window within the current tab, but when testing the product, the software was defective, and the extension failed to load any usable map.
[27] For example, the input text 'New York' could match:
- New York, England.
- New York, Mexico.
- New York, USA.
- New York, New York, USA.

3 Project Context: Background Reading

The follow section will provide background information on a number of key topics that this dissertation covers, including; 'Web Browser Extensions' (section 3.1), 'Usability' (section 3.2), 'NLP' (section 3.3), and 'Information Retrieval' (IR) (section 3.4).

3.1 Web Browser Extensions

Since the first web browsers at the start of the 1990's, many different browsers have spawned and competed with one-another[28]. The intensity of this competition has oscillated over the years[29], but this environment has generally helped accelerate innovations in web browser technologies – among which is included, browser extensions. These are simple software modules built using standard web-technologies (HTML, CSS, JavaScript) that can use "the same web APIs as JavaScript on a web page," but can also access other JavaScript APIs that allow extensions to leverage some browser-level functionality (Mozilla, 2019c).

While extensions have existed for some time[30], a uniform approach to extensions has only recently begun emerging after the launch of Google's Chrome extension APIs[31] (Google, 2009). After this, other major competitors slowly began releasing browser extension APIs which conformed to the majority of the conventions Google had implemented[32]. Though there isn't yet a cross-platform standardisation of these APIs, there is a group within the World Wide Web Consortium (W3C) working towards creating such a standardization – one that offers "actual interoperability rather than mere similarity" between different browsers extension protocols (W3C, 2019a). There are many hurdles in achieving this though, such as the markedly different approach to browser extensions that Apple's browser, Safari, takes (Apple, 2019).

[28] For one an up-to-date history of the internet, see *How the Internet Happened* (Mccullough, 2018).
[29] Over the years, there have been periods where browser innovation has been stifled due to the market being temporarily dominated by a single competitor (Mccullough, 2018). Moreover, some of this market domination, was due to unfair competition on grounds outside of technological innovation (Spinello, 2005).
[30] One of the earliest browser-extensions were possible on Internet Explorer 5 (Microsoft, 2017), which released in 1999 (Microsoft, 1999).
[31] Th success of this Google's method as a uniform approach can be largely attributed the Chrome's dominance of the web browser market (StatCounter, 2019).
[32] Mozilla's Firefox, for instance, followed suit only a few short years after Google's Chrome APIs (Needham, 2015).

Because it's the most widely used approach to developing browser extensions, further explanations on the technology of browser extensions will refer specially to Google's Chrome APIs, although an understanding of how most of the other top browsers are implementing browser extensions can be inferred through this.

3.1.1 How Extensions Work: APIs

Developers can access Chrome's browser API's by first creating a Chrome object within a JS script. Once a Chrome object is created, specific Chrome API's can then be called by adding a specific API's name after it (fig. 3). Which of the many[33] Chrome API's are available to a script depends on:

- A script's execution context
- The keys/permission declared in the manifest
- The extension's Content Security Policies[34] (W3C, 2019b)

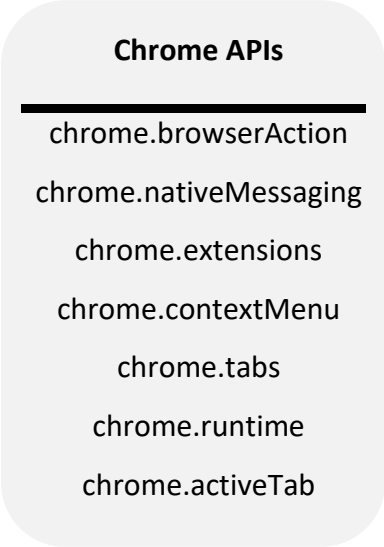


Figure 3: List of several standard Chrome APIs

3.1.2 How Extensions Work: Script Contexts

As mentioned, a script's execution context can influence which browser API's are accessible to that script. These script contexts can be divided into three main groups (W3C, 2019b):

[33] A full list of the Chrome's browser's APIs is available on Google's *Chrome APIs* webpage (Google, 2019h).
[34] Extensions have their own CSPs, but CSPs on web pages may also effect which browser APIs are permissible.

1. **The Browser Extension Context:** These are scripts that run at the browser-level (fig. 4). There are many types of such scripts[35], including 'background scripts' that by default continually run whenever a browser is open. The majority of Chrome's APIs are available at this context level.

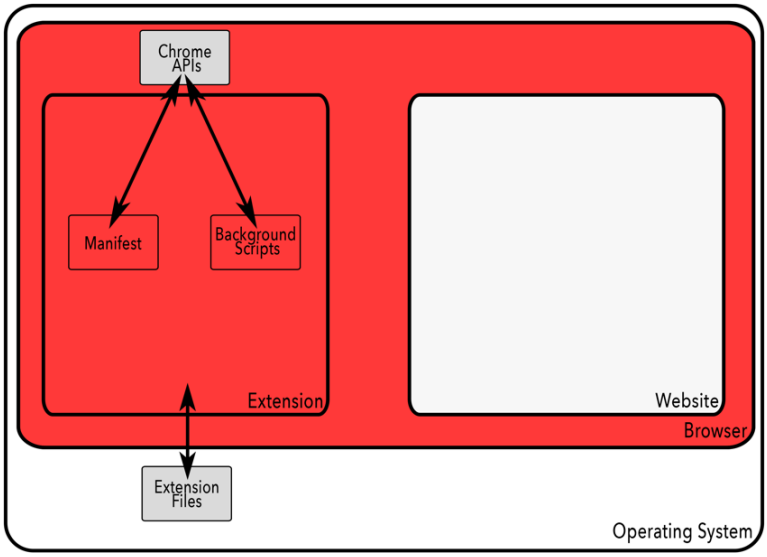


Figure 4: Illustration of the Browser Extension Context

2. **The Content Script Context:** These are scripts that are injected into web pages (fig. 5). These 'injected scripts' are executed on the web page, but within an isolated environment, meaning they can't access any variables created by any other scripts running on that web page. Selecting which web pages content scripts are injected into must be declared in the extension's manifest beforehand.[36] A limited number of Chrome's APIs are made available in this context (Google, 2019i)

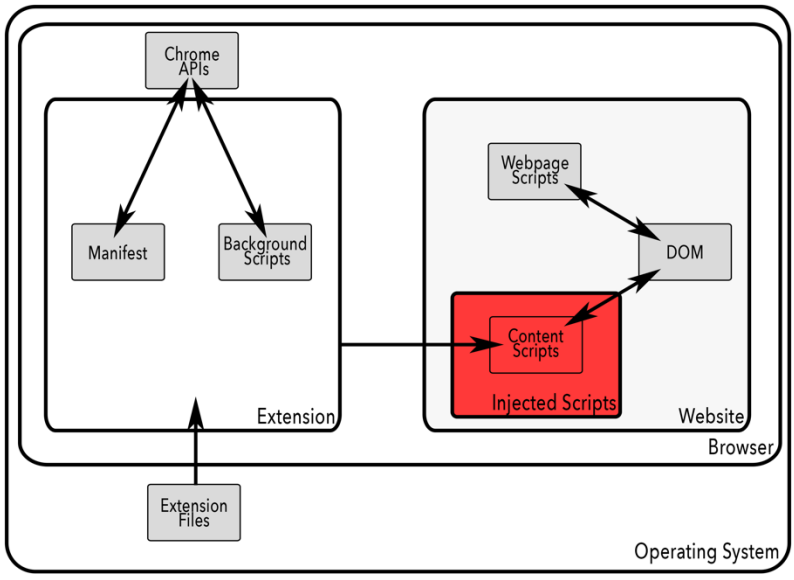


Figure 5: Illustration of the Content Script Context

[35] There are other types of scripts beyond those mentioned here. A full list can be found at on the *W3C Community Draft Report* (W3C, 2019b).

[36] For an example of these declarations, see the 'How Extensions Work: Manifests' (section 3.1.3). In particular, read the description of the 'content_scripts' object, 'matches'.

3. **<u>The Hosted Resources Context:</u>** Most hosted websites and hosted applications can interface with an extension using a browser's messaging APIs[37] (fig. 6, fig. 7).

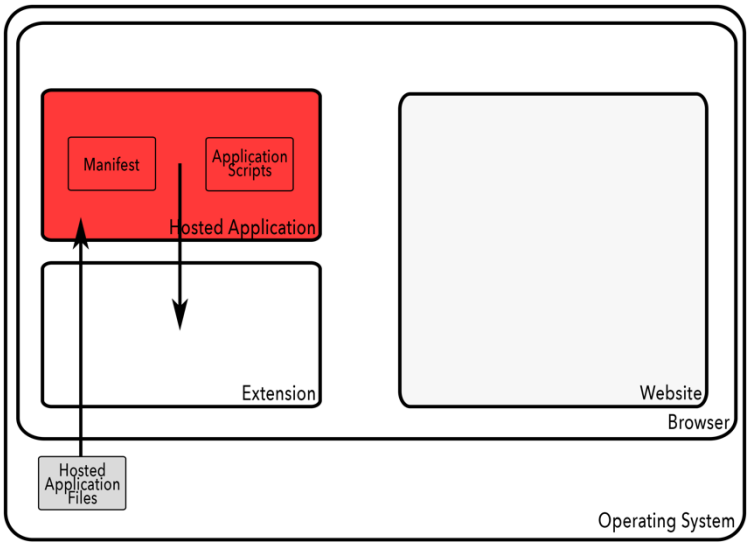


Figure 6: Illustration of the Hosted Resource Context. Depicting a native application hosted locally on the user's machine passing a message to an extension.

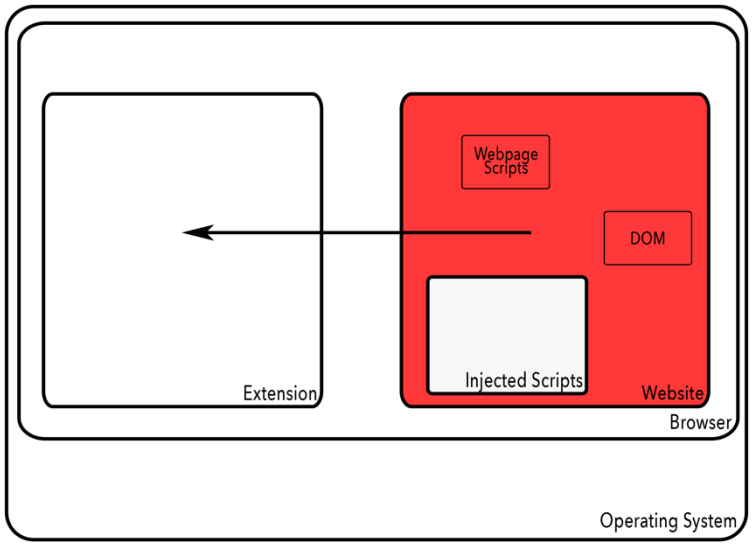


Figure 7: Illustration of the Hosted Resource Context. Depicting a website passing a message to an extension

[37] For example, JavaScript's Window API - i.e.: *window.postMessage().*

3.1.3 How Extensions Work: Manifests

A manifest file is a mandatory JSON-formatted file that allows developers to specify any aspects of an extension which require declaration[38]. The following example manifest file details some notable manifest fields (fig. 8):

```
{
   "name": "",                          // Name of the extension.
   "version": "",                       // Version of the extension.
   "manifest_version": 2,               // Version of manifest (currently 2).
   "description": "",                // Short description of extension.
   "homepage_url": "",               // Extension homepage (e.g. GitHub page).
   "icons": {                           // Specifies the extension icon.
      "16": "",                      // File location for 16x16 version.
      "48": "",                      // File location for 48x48 version.
         "128": ""                        // File location for 128x128 version.
   },
   "browser_action": {               // Specifies display and functionality
      "default_icon": ""             //   related to the extension's icon on
   },                                   //         the browser toolbar.
```

[38] A full summary of all Chrome's Manifest fields can be found online (Google, 2019j).

```
  "permissions": [],              // Used to declare specific Chrome APIs[39].
  "content_scripts": [            // Used to declare specifics on an exten-
    {                                 //   ions content scripts, including:
      "matches": [""],            //   - Which URL's to run scripts on.
      "css": [""],                //   - Path's to CSS scripts to run.
      "js": [""]                  //   - Path's to JS scripts to run.
    }
  ],
  "web_accessible_resources": [], // Paths for the hosted resources context.
  "background": {                       // Used to declare background scripts.
    "scripts": [""]               // A list of background scripts.
  },
  "content_security_policy": ""   // Allows changes to be made to an
}                                                 //    extension's CSP.
```

Figure 8: Example manifest file

3.1.4 How Extensions Work: Summary

These three concepts described above are central to the protocols that most modern browser extensions employ:

1. **JavaScript Browser APIs:** Used to provide access to browser-level functionality.
2. **Script Contexts:** Used to determine if and how these APIs can be used.
3. **Manifests:** Used to provide information and permissions to the extension. The manifest is effectively a foundation, or "point of entry", for an extension (Cloud, 2015). When an extension is installed, the browser looks for the manifest in the root directory and won't install the extension unless it finds the manifest.

[39] Many Chrome API's require that they're declared before the extension can access them.

Once an extension's been built, the specifics of how it's installed or uploaded to a browser's extension-store vary between each web browser. On Chrome (fig. 9), installing an extension manually can be done by:

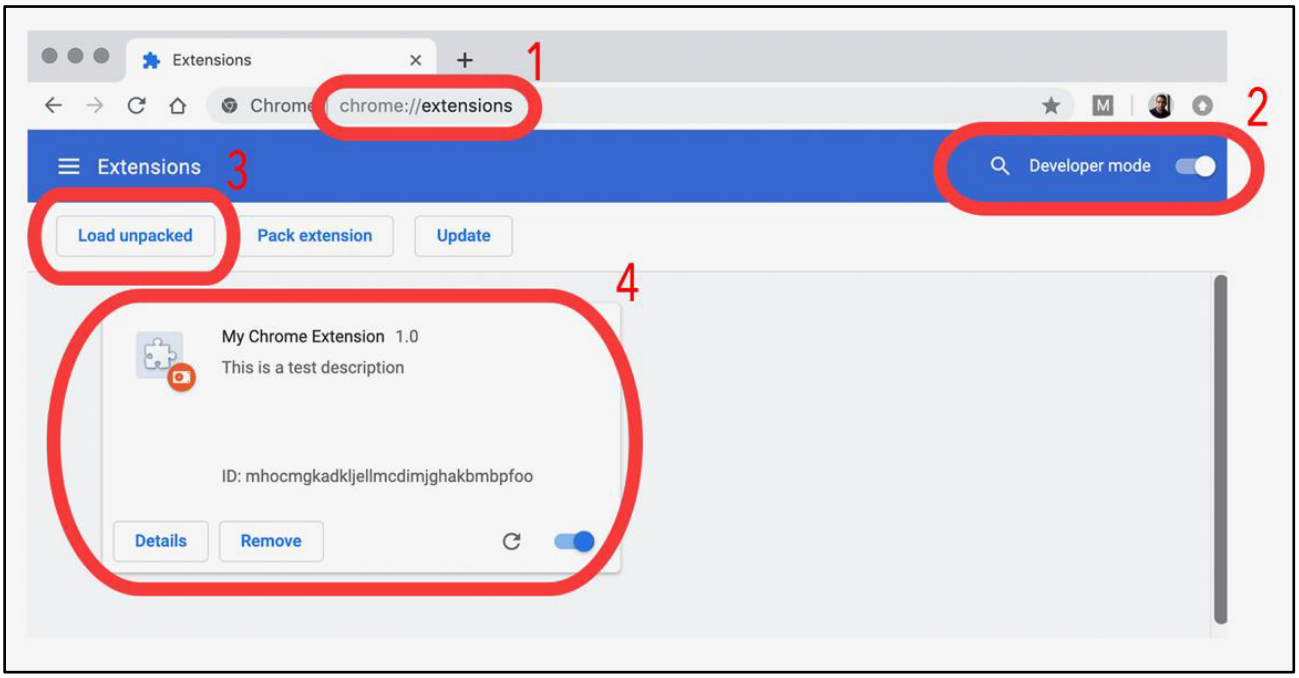


Figure 9: Visual guide for uploading a Chrome extension to Chrome. 1: Navigate to chrome://extensions. 2: Enable developer mode. 3: Load the extension directory. 4: Once loaded the extension will be given an extension ID and will appear listed.

1. Navigate to chrome://extensions.
2. Enable developer mode.
3. Load the extension directory.
4. Once loaded the extension will be given an extension ID and will appear listed. This ID is the name of the directory for the extension, such as:

```
<Username>/Library/Application Support/Google/Chrome/Default/Extensions/<ExtensionsID>/
```

Now, this extension directory will be executable whenever Chrome is loaded.

3.2 Usability

Usability is heavily linked to the study of Human–Computer Interactions (HCI) – a branch of the science of ergonomics which focuses on developing standards for interactions with visual display terminals (VDTs). More broadly, usability can be understood as a portion of the UX; a turn-of-phrase which encompasses "all

the users' emotions, beliefs, preferences, perceptions, responses, behaviours and accomplishments that occur before, during and after using a product" (Helms, 2006). More specifically though, usability itself can be understood to be composed of the following five attributes (Nielsen, 1993a):

1. **Learnability:** A system should be easy, and quick, to learn.
2. **Efficiency:** A system should encourage productivity through how efficient it is to continually use.
3. **Memorability:** 'How to use a system' should be easy to recall, even between long lapses in uses.
4. **Errors:** A system should have a low error rate, and easy recovery from errors should be possible.
5. **Satisfaction:** A system should be pleasant to use and should subjectively satisfy its users.

These attributes explain what usability is, but don't explain how to implement it. For this, the *ISO 9241: Ergonomic requirements for office work with visual display terminals*, is perhaps most commonly referred to, although many other heuristic[40] guides to implementing usability are also available [41]. Additionally, there are specific guides to the usability heuristics of particular technologies too, such as guides to smartphone usability heuristics (Inostroza, 2015). There is also ongoing research into formulating a standardized methodology for creating specific usability guides – a meta-standard for developing usability standards, if you will[42].

Usability isn't simply part of the software development phase either. Ideally, usability engineering takes place in a cyclical fashion "throughout the lifecycle of the product" (Nielsen, 1993c). To describe the usability cycle, the same process models that are typically used to describe software development can be used, such as the Life Cycle Model, or Spiral Model. Despite the

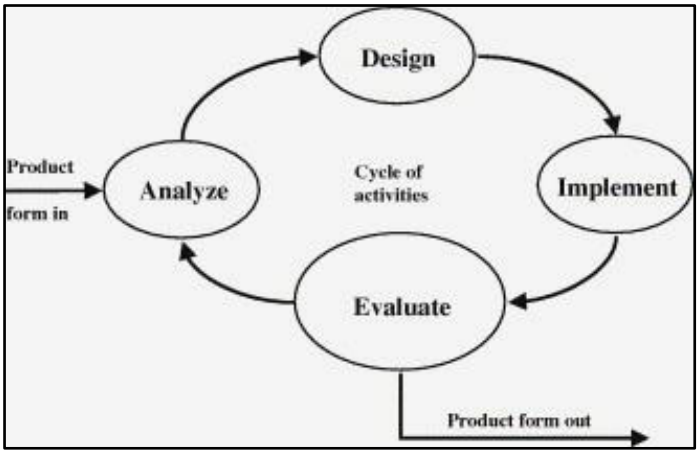


Figure 10: Diagram depicting the four types of development activities (Helms, 2006).

[40] Heuristics can be understood as "a way of solving problems by things yourself" (Cambridge, 2019).
[41] Some other guides are; (Mayhew, 1992) and (Nielsen, 1993d).
[42] This can be read about in papers such as; (Van Greunen, 2011) and (Hermawati, 2015). For a review of all the research in this area, see; (Quiñones, 2017).

different models available, they can be all be generally understood as “consisting of the same four types of development activities” (Helms, 2006) (fig. 10). With these four development activities, the usability of a technology can be understood as a component of the product that continues to improve iteratively over time. As Nielsen said, “One potential opportunity for long-term progress in usability is that new user interface generations can build on the capabilities of earlier generations, while adding new interaction capabilities” (Nielsen, 1993a). This iterative process towards improving usability can be demonstrated in web browsers, a technology whose usability has continued to improve over the years and will likely continue to improve well in the future. Web browser extensions themselves represent another step on this iterative path.

3.2.1 Usability in Web Browsers

The improvements to usability that Google’s Chrome UI brought over its competitors when it first launched was noticeable (fig. 11):

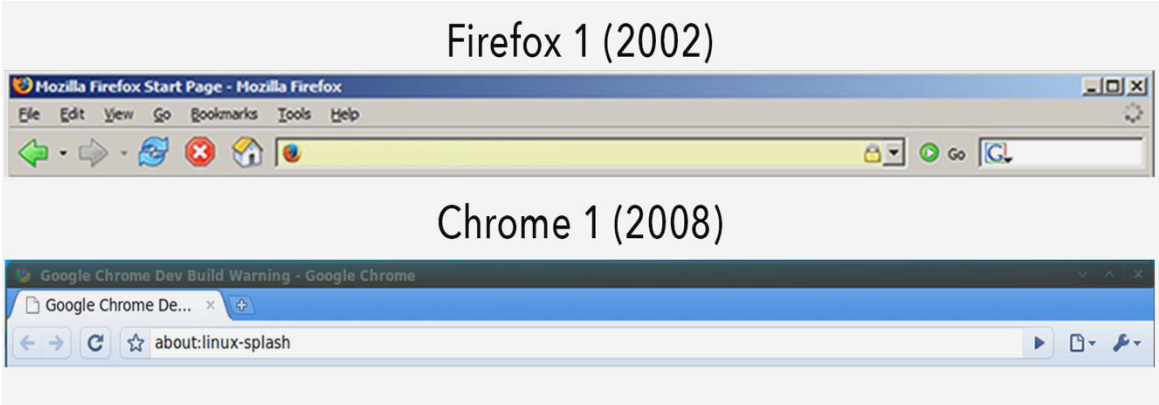


Figure 11: Comparison of Firefox Browser UI with Chrome’s browser UI.

As stated in the *ISO 9241*[43], good ergonomic design is especially important when “use is intensive” (BSi, 2008). With respect to how intensively browsers are used in modern times, Google’s decision to overhaul and simplify the UI was particularly appropriate (fig. 11). Compared to its rivals, Google removed everything but what was essential from its UI, generally only leaving UI functionality that was essential to

[43] It’s worth acknowledging that within the section of the ISO that’s being reference, it’s remarked that the “user interface of different types of user agents (such as web browsers)”, isn’t “the subject of this part of ISO 9241.” However, the ISO then further states that some of its “guidelines could apply to those systems as well” (BSi, 2008).

navigation. Because of how popular Chrome has become, it's likely that one of the many reasons that contributed to its success was its overhaul the traditional browser UI. Arguably, Chrome's UI's focus on simplicity created a browser UI with better learnability and efficiency than its competitors. This logic seems reasonable, given the fact that most other modern browsers have since copied Chrome's UI (fig 12):

Figure 12: Comparison of Edge & Firefox browser UIs, after copying Chrome's UI.

Beyond UI, the evolution of browsers have brought many other advancements which have also improved web browser usability. Some of these other improvements include; HTML 1-5, CSS 1-3 JavaScript (ECMAScript 1 - ECMAScript 2018), XML, AJAX, and mobile support. These are all advancements in how people interact with the internet that web browsers helped facilitate. Browser extensions are themselves a web browser advancement, and through them, developers are able to improve browser usability in interesting ways. There are many examples of such extensions. For instance, there's an extension that will display every tab a user has open simultaneously (Google, 2019p). There are also many extensions for improving the UI of popular websites like YouTube (Google, 2019q), or Facebook (Google, 2019r), and as mentioned previously, this project too aims to improve the usability of web browsers.

3.3 NLP

Natural Language Processing is a sub-discipline of artificial intelligence and computational linguistics that is focused on the development of systems that implement natural language understanding (Oxford, 2016). This has many modern practical applications (such as machine translation, speech technologies and writing assistance) which many browser extensions are now leveraging (such as Google Translate, Read Aloud, and

Grammarly)[44]. For this project, however, NLP is applied for a different application, text analytics; "the detection of useful content or information in textual sources, typically either via classification or extraction" (Dale, 2019). NLP has many functional applications within text analytics[45], but it perhaps best known for its impact in Big Data (Ray, 2018), and in particular, sentiment analysis (SA), where in modern business, SA "systems are being applied in almost every business and social domain" (Liu, 2012).

Beyond Big Data, there are also other ways to leverage NLPs text analytics applications, such as its use in this project, which leverages NLP for information extraction (IR) via entity recognition (ER). ER is a task accomplished by a sub-process within the NLP pipeline called a Named Entity Recognizer (NER). NERs identify entities and classify them by their type, such as 'Mount Everest' as a 'location', or 'USA' as a 'geopolitical entity'. Incorrect classifications can occur however, but current state-of-the-art NLP models are continuing to improve the accuracy of their classifications, as demonstrated by recent NLP benchmark tests, such as the Generalized Language Understanding Evaluation (GLUE) benchmark[46] (GLUE, 2019). Furthermore, the accuracy of entity recognition should continue to improve in the coming years, as suggested by a recent a meta-analysis of the area which detailed recent NER advancements (Yadav, 2018).

[44] ((Chrome, 2019m), (Chrome, 2019n), (Grammarly, 2018)).
[45] For a brief review of these applications, see *Functional Applications of Text Analytics Systems* (Simske, 2019).
[46] For an overview of GLUE, read *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understandin*g (Wang, 2019).

In terms of this project, the specific NLP system that this project uses is a NLP library called 'SpaCy'; a free and simple NLP library written in Cython (SpaCy, 2019a).

Within SpaCy, entity recognition happens after a few other key NLP subprocesses, including (fig. 13):

1. **Tokenization:** The chopping up of documents of text into individual words, also known as tokens.
2. **Parts-of-Speech Tagging:** The assigning of the parts of speech to each token, such as 'noun' or 'verb'.
3. **Dependency Parsing[47]:** The assigning of syntactic labels which describe the dependency relationship between tokens, such as 'subject', 'object' or 'predicate'.
4. **Lemmatization:** The assigning of a standardised, uninflected tenses to each token, such as assigning the lemma 'eat' to 'eating', 'eaten' or 'ate'.

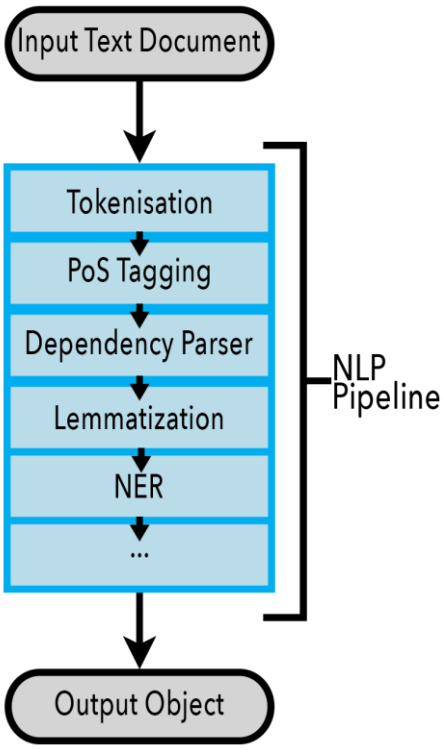


Figure 13: Diagram of NLP pipeline.

These are some of the most essential NLP subprocesses[48]. In SpaCy, throughout these NLP subprocesses, in order to make accurate classifications most of these subprocesses (with the exclusion of tokenisation) leverage predictions made using SpaCy's statistical language model[49]. SpaCy provides many pre-trained models, and additionally allows developers to perform their own training (SpaCy, 2019b). This project will use two language models built by the SpaCy team; 'en_core_web_lg' & 'en_core_web_sm'. These models use GloVe vectors[50] (trained on Common Crawl (SpaCy, 2019c)) to create vector representations of input text, and then use a Convolutional Neural Network (CNN) (trained on the OntoNotes corpus (LDC, 2013)) to make semantic and syntactic predictions on the text. According to the SpaCy team, this language model yields an NER precision of above 87% (GitHub, 2019a).

[47] It's worth noting that SpaCy's dependency parsing was at one point considered the fastest parser in the world, at least according to one 2015 paper (Jinho, 2015).
[48] Other subprocesses in the NLP pipeline include; sentence boundary detection and text categorization.
[49] Statistical language model's are used to provide predictions on words/phrases that are similar to each other.
[50] GloVe is an unsupervised learning algorithm for "obtaining vector representations for words" (Stanford, 2019), wherein words that are mapped closely together have more semantically similar than words mapped far apart from each other.

Using the SpaCy NLP library, this project extracts NEs of a geographical type, and then uses that information to perform IR in order to display; maps, Wikipedia information, news, and images, all related to the NE that was extracted.

3.4 Information Retrieval

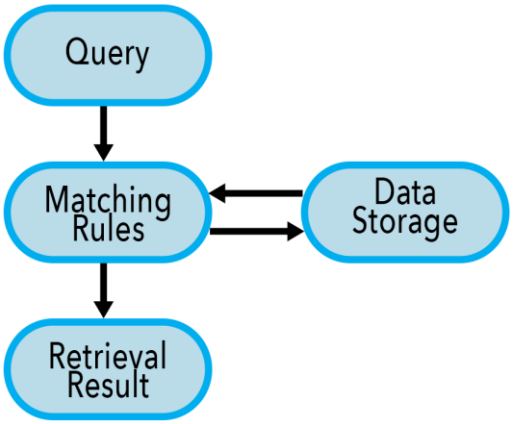


Figure 14: Diagram of Simple IR System.

Information Retrieval covers the processes involved in the retrieving information and storing information (fig. 14). Some common IR applications are (Büttcher, 2010):

1. **Web Search:** This involves processes which match user queries with web pages. This involves steps such as; finding matches, evaluating matches (i.e. scoring each match, removing identical matches), generating summaries for matches, and returning matches.
2. **Desktop Search:** This is similar to web search, but these types of searches require a direct interface with the file system of one's own operating system.
3. **Text clustering:** Systems responsible for grouping documents by their similarity.

IR has many more applications beyond these[51], but its most popular use is unquestionably web search, with Google handling over 1.2 trillion searches per year in 2012, and by extrapolation[52], likely more than 2 trillion search per year in 2019 (Google, 2012). This sheer volume of queries is made even more complicated by the vast amount of ever-growing content on the web, with Google claiming that their index of the internet contains "hundreds of billions of web pages" (Google, 2019). Building IR systems that are both effective and efficient[53] at searching data of this magnitude is one of the many challenges that make IR a popular area of research.

[51] For a more expansive breakdown of IR applications, read; Büttcher's *Information Retrieval*, pp. 2-4 (Büttcher, 2010).
[52] Between 2012-2018, then number of internet users worldwide increased by more than 60% (StatCounter, 2019b), and it's reasonable to assume that this growth would correlate with an increase in web searches.
[53] Efficiency and effectiveness are considered the "two principal aspects to measuring IR system performance" (Büttcher, 2010).

Within the field of IR, one area of IR research that's received much attention is the investigation of using NLP to create 'entity linking'[54] across documents, a process wherein similar NEs from disparate text documents are linked together to improve the performance of IR tasks, such as text clustering and web search.

Beyond this, another field of IR research is query expansion (QE), an area which aims to tackle the IR "mismatch problem" wherein a user's query "and the document collection don't [*sic*] use the same words for the same concepts" which leads to missed matches when performing IR (Dahab, 2017). QE techniques overcome this by expanding the original query with related words so that the query is less susceptible to this mismatch problem. Some methods of QE involve leveraging Wikipedia and WordNet for 'semantic enrichment' of an initial query. Some recent research has even leveraged both of those resources simultaneously – an approach which has yielded "significant improvements" over other "related state-of-the-art approaches" to QE (Azad, 2019).

This field also ties in well with another popular area of IR research, event queries. These are queries which focus on IR pertaining to events, such as news articles. Event query research is needed because online content providers are generating such an exorbitant amount of new event information every day that specific solutions for searching this type of information efficiently and effectively are needed, particularly as event type information similarly suffers from the mismatch problem. Some of the approaches being used recently to improve event query IR are; modifying text clustering to leverage spatiotemporal information from events (Norouzi, 2019), and using NLP key phrase extraction from events (Gupta, 2019). Both of these approaches can also be combined, as demonstrated in one recent paper, which created a "bi-clustering model for clustering the documents and key phrases simultaneously" (Zhoa, 2019).

[54] For a review of much of the research on entity linking, read Hasibi's, *Entity Linking in Queries: Efficiency vs Effectiveness* (Hasibi, 2017).

3.4.1 How This Project Uses IR

This project incorporates some of the concepts described above by utilizing; NLP, SQL, and leveraging the IR systems within several APIs; Google Maps API, Bing News API, Bing Images API, and MediaWiki[55] (fig. 15).

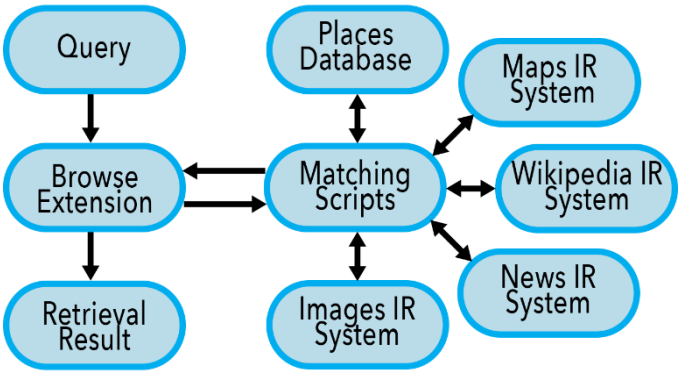


Figure 15: Diagram of IR in this project.

First, NLP identifies the NE within any input text. If it's a location, or geopolitical entity, then the NE is extracted. This NE is used to search a database in order to find all the potential places that the user may be interested in. All these matches are then sent back to the user, so that they can specify the precise place they're interested in from a list of potential matches.

Second, using the text from the user's chosen 'location-of-interest', the query is expanded from the original NE, to the full name of the location[56]; this way, more relevant queries can be made using the IR APIs mentioned above (i.e. maps, Wikipedia…). Each of these APIs performs IR using the full name of the 'location-of-interest' and then returns the matches (fig. 16). Finally, all of these results are displayed to the user in a small popup window.

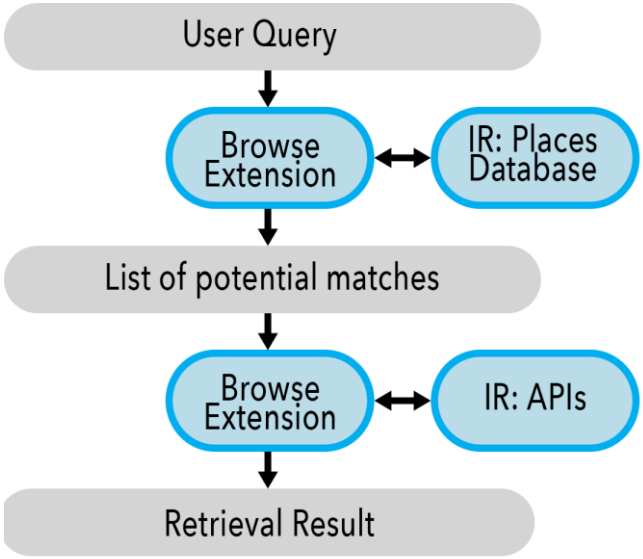


Figure 16: Diagram of sequence in which this extension uses different IR systems.

4 Project Requirements

Requirements describe what a system will do (functional requirements), and how a system will do it (non-functional) (Stack Overflow, 2017). In order to reach the 'Aim & Objectives' (section 2.1) laid out in the 'Project Introduction' (section 2), it's crucial that the functional & non-functional requirements are

[55] (Google, 2019o), (Microsoft, 2019a), (Microsoft, 2019b), (Wikimedia Foundation, 2019).
[56] For example, the NE 'New York', might be expanded to 'New York, New York, USA'.

delineated before development of the software begins. This will help to inform the system design of the software, and will help ensure the project is completed successfully. Moreover, the failure to produce sufficient requirements is a common reason that many software projects don't succeed (Hubert, 2001).

4.1 Functional & Non-Functional Requirements

Functional Requirements

1. User should be able to highlight any 'highlight-able' text on a webpage and right-click to reveal the project extension (WhereFinder) in the browser's context menu.
2. After submitting their input, if there are multiple potential locations that match the input text, the user should be able to clarify which location they're specifically interested in.
3. The extension should generate a pop-up window in the browser displaying information returned from the user's query, including; a map, Wikipedia information, relevant news articles, and relevant images.
4. The extension display window should consist of a map as its main display, with a series of tabs at the top of the window to access more supplementary content.
5. A pin drop displaying the location-of-interest, and the user's current location, should be displayed on the map.

Non-Function Requirements

1. Although some APIs and libraries might implement other languages, the extension will be coded using a combination of: HTML, CSS, JS, Python, and SQL.
2. The extension should use an SQL database to find all potential matches to the user's input text.
3. The code written for execution within the browser context should handle communication and display functionality only
4. The code written for execution within the browser context shouldn't handle the heavy processing tasks (NLP, IR) of the extension – these tasks should occur on a separate Python server.
5. All information collected and passed through the extension should be stored as JSON.
6. The Python server should use NLP to extract and identify the geographical location of interest.
7. The Python server should be able to find additional content related to the input query using the IR systems of several APIs; Google Maps API, Wikimedia API, Bing News Search API, and Bing Image Search API.

8. The Python server should provide a response with all the information necessary to fulfil the query that was sent in the request.
9. The extension must be free to run, meaning any APIs used must not incur any costs.

5 Project Assumptions & Constraints

The following assumptions and constraints have been made about this project:

Assumptions

- Users want browser extensions that improve the usability of their browser.
- The project will leverage the guidance of the project supervisor, Dr. Padraig Corcoran.
- The software in this project will be solely written by one person.
- Free APIs may not provide the optimal software solutions but will be sufficient for meeting this project's 'Aim & Objectives' (section 2.1).
- Each task delineated in the timescale will be completed on-time or will vary in their completion by no more than a few working days.
- The user knows how to use a web browser.
- The user isn't colour-blind.
- The user can read English.

Constraints

- The development schedule of this project must not be longer than 8 weeks.
- The browser extension must be free to operate.
- Due to the short development schedule, the scope of this project will be limited, meaning that this extension will be developed solely for the Chrome browser. It will not work on any other browser.
- Due to the limited timescale, software testing will be limited to ensuring that the software is functional.
- Due to the limited timescale, optimisation will not be a focus for this project.
- Due to the limited timescale, application security will not be a focus for this project.

6 Project Approach

As explained, this project will build an extension that extends the usability of the web browser by adding new functionality that will enable users to retrieve supplementary content about geographical locations without requiring them to leave their current web page. In this section, the approach to this project is discussed in terms of; 'Development Strategy Considerations' (section 6.1), 'Business Considerations' (section 6.2), and 'Software Considerations' (section 6.3).

6.1 Development Strategy Considerations

For this project, different development strategies were considered. The Agile model was considered first. This approach would make the development of this project a much more iterative process as one of the tactics in Agile development is to "Deliver working software frequently" (Agile Alliance, 2001). Moreover, Agile projects tend have fewer defects (Rico, 2009), which makes an Agile approach particularly appealing.

Secondly, the Waterfall approach was considered. This approach would make this project's development much more linear, as in the Waterfall model "each phase is considered to be completed" (Ivins, 2019b) before moving on to the next phase. This type of approach can work well for small projects. One of the main advantages of this approach is that it makes the project's development schedule easy to understand. However, at the start of a project it can often be difficult to ascertain what the ideal set of requirements for the project are. Often throughout a project's development, the requirements can themselves develop, as one's nuanced technical understanding of the project emerges. This can be a problem in the Waterfall model because a Waterfall approach to development leaves "little scope for incorporating change" (Ivins, 2019b) as a project progresses.

After deliberation, the Waterfall model was chosen over the Agile model for this project. The reasoning behind this choice was predominantly because of the large amount of API/library documentation that would have to be learned during this project's development. Because time is needed to learn, practice, and

implement functionality using these technologies, ensuring that enough development time can be scheduled for all of these tasks will be much simpler using the Waterfall model. Therefore, a Waterfall modelled development schedule for this project has been created, which can be viewed in the section, 'Project Timeline' (section 7).

6.2 Business Considerations

In any project, it can advantageous to consider the risks and disbenefits of undertaking the project beforehand so that the appropriate mitigations can be put in place before the issues arise. This helps to ensure the success of the project. With this in mind, the follow section is broken into two subsections; 'Risks' (section 6.1.1), and 'Disbenefits' (section 6.1.2).

6.2.1 Risks

The ISO define a risk as the "effect of uncertainty on objectives" (BSi, 2018). Effective risk management is helpful because it helps one "increase their chances of achieving their objectives (Ivins, 2019a). This section will delineate this project's risks (table. 2). Each risk is broken down into its constituent parts:

- **Risk Identification:** Explains what the risk is.
- **Risk Type:** Categorizes the risk by its charactertics[57].
- **Risk Analysis:** Explains how the risk arises (a 'likelihood' scale (1-10) is provided to indicate the likelihood of the risk arising).
- **Risk Evaluation:** Explains why the risk poses a problem (a 'severity' scale (1-10) is provided to indicate the severity of the risk, should it arise).
- **Risk Treatment:** Explains how to treat the risk (a 'solution' scale (1-10) is provided to indicate how effective the proposed treatment is in mitigating the risk).

[57] For the purpose of the project, risks can be divided into five types:
- Technical Risks: Issues related to technology implementation.
- Project Risks: Issues associated with the actual development phase of the project.
- Legal Risks: Incidents that could negatively impact the project due to a lack of understanding of regulations and laws.
- Ethical Risks: Incidents that could negative impact the project due to a lack of awareness on ethical circumstance around the situation.
- Business Continuity Risks: Serious incidents which can greatly impact the entire project that should ideally be handled in the planning stages of a project.

Risk Identification	Risk Type	Risk Analysis	Risk Evaluation	Risk Treatment
Some of the APIs may be difficult to learn.	Project	**Likelihood: 5** Some of the APIs may have poor documentation, or bugs which are unaccounted for.	**Severity: 7** If this happens, it could disrupt the development schedule, which could impede, and even halt, progress on the project.	**Solution: 4** Ensure that enough time has been allocated in the development schedule for learning the APIs.
Errors within certain execution contexts might prove difficult to debug. (I.e. debugging script injections into remote web pages).	Project	**Likelihood: 9** Script execution contexts in web development can sometime make it almost impossible to know what's gone wrong because the errors aren't easily visible from where the execution was made.	**Severity: 5** These situations can lead to some difficult sections of development, but there is typically a solution to be found.	**Solution: 8** Use Google Chrome's developer console and write commands within injected scripts that print messages to the console throughout the code's execution.
Some of the APIs may not work precisely as intended.	Project	**Likelihood: 2** Humans are prone to making errors or overlooking certain design details, so there is a chance that an API doesn't function precisely as intended.	**Severity: 3** Should these situations arise, there should be a way to work around the issue by implementing a different API, or finding another workaround.	**Solution: 9** Aim to only use APIs and libraries that come from larger development teams, with good documentation, or that are very widely used, and so are likely therefore to be less prone to error.

The software architecture of this project may not be appropriate for what is needed as the project grows.	Project	**Likelihood: 8** Due to factors like the short development time and the use of the Waterfall model, it's possible that decisions made on the project architecture early on are inefficient for what is needed as the project progresses.	**Severity: 5** For a project of this size, changing the software architecture after a program is built could be very difficult and time-consuming. Additionally, a sub-standard architecture would reduce the efficiency of the extension and could cause latency in its use.	**Solution: 7** Practice building several simple Chrome extensions so that a solid understanding of how to architect an extension is understood.
Taking a Waterfall approach to development may make it difficult to make large changes to code written earlier in the software's development.	Project	**Likelihood: 9** This is a well-known risk of the Waterfall model. As a project's complexity grows, a Waterfall approach makes it difficult to change earlier work.	**Severity: 5** While this shouldn't inhibit the project for being completed, it will likely lead to some inefficiencies in the code.	**Solution: 7** Ensure that an understanding of how all of the extension different scripts perform messaging passing is grasped early on in the project, because communication could easily be an area which causes complication later on in the project.

Tasks within the development schedule might exceed the time they've been allocated to complete.	Project	**Likelihood: 9** Because there's so many separate activities in the development timeline, it's likely the timescales suggested for some of the activities are incorrect.	**Severity: 6** Because the development schedule is short, if this issue arises multiple times, then then this could harm the quality of the final product as the remaining tasks may need to be rushed to complete the project on time.	**Solution: 7** Attempt to complete certain tasks early if possible, and make sure the development schedule isn't underestimating the time it takes to complete any of its tasks.
Some of the APIs necessary for building the extension may require financial costs.	Project	**Likelihood: 5** Many API's are only free to use up to a certain number of requests. However, the costs of making requests is only likely to become an issue if this extension sees a full release and becomes popular.	**Severity: 2** The costs per request is typically done in the thousands, so even if using the APIs requires some spending, unless the extension because extremely popular, the cost is unlikely to be particularly high.	**Solution: 10** Set aside a small amount of capital to use in case usage costs from an API are incurred.

The project code could become corrupted or lost.	Project/ Business Continuity	**Likelihood: 2** Due to complications like a system failure or a disk failure, this project's code could become unusable.	**Severity: 10** If this happens, the project may not be completed on time, and could fail to reach its aim.	**Solution: 9** Keep a remote repository of all of the project's directories and ensure that the remote repository is kept up-to-date.
The entire development schedule might take much longer to complete than the time allocated to complete the software.	Business Continuity	**Likelihood: 2** Due to the complexity of the project, there's a chance that the development schedule doesn't leave enough time for the project to be completed, meaning that the deliverables of this project will be unfinished.	**Severity: 10** If this happens then the integrity of this entire project could be questioned, which would also undermine all the work that's been completed for the project.	**Solution: 9** Ensure that the development schedule is followed as closely as possible, and that the project supervisor is referred to if there's any significant issues with the project's development.
In the future, Google may extend the functionality of the Chrome browser to perform similar functionality to this extension.	Business Continuity	**Likelihood: 8** It's reasonable to think that eventually Google might build Google Maps functionality into their Chrome browser themselves.	**Severity: 10** Google's service would likely be superior to that of this extension, and would reduce, or even eliminate, the need of this extension to exist.	**Solution: 10** This risk is unavoidable, but it would ultimately be a testimony to the foresight of this project if this did occur.

Even after testing, there may be bugs left in the extension.	Technical	**Likelihood: 6** There's a chance that certain bugs will be overlooked during the development of this extension. These bugs could result in the extension being unusable.	**Severity: 7** It will reflect poorly on the usability of the product, but upon restarting the browser, the extension will also restart and should then function as normal, until another such crash incident arises.	**Solution: 8** Ensure that a large amount of software testing is completed on the final product so that the number of defects in the software's performance can be minimized.
The extension may be susceptible to security threats.	Technical	**Likelihood: 10** This extension has the potential to be open to many security threats, such as; interception, interruption, modification, or fabrication.	**Severity: 10** This could result in a user of this application having their personal data compromised.	**Solution: 8** This project's lack of focus on security was acknowledged in the 'Project Assumptions & Constraints' (section 5), but if this extension does see a commercial release, additional testing will need to be done to ensure that it's secure.

Some of the functionality of the extension may be considered unlawful.	Ethical/ Legal	**Likelihood: 5** New internet regulatory laws, such as the GDPR, could be expanded upon in a way that would make the way this extension operates unlawful in one way or another.	**Severity: 1** This extension only collects a very limited amount of personal data (a user's location), so If there is an issue, amending the software to adhere to any regulatory problems should be straightforward.	**Solution: 9** Be prepared to alter aspects of the software, should an issue arise.

Table 2: Risks Table for this project

6.2.2 Disbenefits

Disbenefits are project outcomes that will be "perceived as negative by a stakeholder" – inevitable drawbacks incurred from completing a project (Bradley, 2006). This section will delineate this project's disbenefits (table. 3).

Disbenefit Identification	**Disbenefit Analysis**	**Disbenefit Evaluation**	**Disbenefit Treatment**
Some users will instinctively feel that the extension reduces the 'learnability' of the Chrome browser.	Because this extension changes the functionality of the Chrome browser somewhat, this will require additional 'learning' from users, some of whom will likely oppose the additional learning required.	**Severity: 4** Ideally, any reduction in learnability will be offset by the other benefits that using this extension will create.	**Solution: 7** Ensure that the extension's UI is as simple as possible so that the extension's learnability is as high as possible.

Some users may believe that there is additional functionality that the extension should have.	Different users will have varying expectations, and some will invariably want this extension to have functionality that it doesn't.	**Severity: 6** This extension was designed to perform limited IR, but there might be some particularly advantageous functionality that's missing.	**Solution: 6** Run user testing, and if the extension fails to perform well in these tests, then consider changing the functionality of the software.
If the extension is to be released on browsers other than Chrome, time will have to be spent making changes to the code before the extension will work on those browsers.	As there is no standardized method for building extensions, there exists differences between building extensions on different browsers that will need to be accommodated for if this extension is to see a release on other browsers.	**Severity: 2** As explained in the 'Project Context' (section 3) section, with the exception of Apple's Safari, most other browsers use a fairly similar method for building extensions.	**Solution: 7** Read documentation on porting extensions before attempting to do so, and develop an appropriate plan for porting the product.
The Content Security Policies of certain web pages will restrict certain modifications to the web page.	Web pages can sometime decide to omit certain APIs on their page, which could mean that this project is unable to operate on certain web pages.	**Severity: 1** This is unavoidable. There will always be certain pages on which this extension can't run. For example, Google doesn't allow any extensions to execute Chrome APIs if the URL scheme is 'chrome://'.	**Solution: -** No solution is currently possible for this disbenefit.

Table 3: Disbenefits table for this project

6.3 Software Considerations

Choosing the appropriate software can help determine the success of a project. This section will delineate the 'Languages' (section 6.2.1), 'Applications' (section 6.2.2), 'Libraries' (section 6.2.3), and remote 'APIs' (section 6.2.4) used in this project.

6.3.1 Languages

The following languages have been used to build this extension:

1. HTML & CSS

As fundamental web technologies, HTML & CSS were used to build and style all the visual components of the extension, including; the WhereFinder context menu, the WhereFinder loading animation, the WhereFinder main display window, the displays within each of the WhereFinder tabs, and the Google Maps Iframe web page.

2. JavaScript

As another fundamental web technology, JavaScript was used to build all the functionality across the extension, including; message passing between scripts, script injections into webpages, event listeners and event functions, and generating the Google Maps display within the Iframe.

1. SQL

SQL was used to build an SQL DB, and was also used to communicate between the DB and Python. The database held an extensive list of locations, which a Python script would leverage the database to find all the potential places that matched the user's query.

2. Python

Python was used build two scripts. The first, was a custom Python module containing a text processor class with all of the methods needed to meet this extension's text processing needs. The second was a script to communicate with the rest of the extension, and execute the first script's methods. Python was chosen

over other languages because it's known to be good for "rapid prototyping" (Rossum, 1995), and as the development time for this project is limited, being time-efficient was particularly important.

6.3.2 Applications

The following applications were used to build this extension:

1. Visual Code

Visual Code (VC) is a well-known Integrated Development Environment (IDE) commonly used in programming. VC was particularly useful because of a number of its features. First, its side-by-side editing capabilities were particularly useful when trying to modify code on two scripts simultaneously. Second, VC's ability to load workspaces was also useful because it simplified directory navigation when moving between many different scripts across different directories.

2. Chrome

As mentioned previously, this extension was built for the Chrome web browser. As such Chrome was used heavily throughout this extensions during development to run the extension and monitor its functionality.

3. Xampp

Xampp is a simple solution to building web servers built in PHP and Perl. Its databases are built using MYSQL. Xampp was chosen as the simplest way to quickly deploy a database on a server, allowing more time to focus on other areas of development. The database deployed was an exhaustive list of locations, which a Python script queried during the text processing segment of this project's code.

4. GitHub Desktop

To maintain a remote repository, GitHub's servers was used (GitHub, 2019b). To access the remote repository, the GitHub Desktop application was occasionally used.

6.3.3 Libraries

The following libraries were chosen to build this extension:

Python

Library Name	Description
country_list	Used to check country abbreviations.
geocoder	Used to find current latitude and longitude coordinates of the user.
json	Used to pack and unpack JSON objects.
langProModule	Custom built module used to process input text.
logging	Used for debugging.
pymysql	Used to communicate with an SQL server.
re	Used for regular expression.
requests	Used to send HTTP requests.
spacy	Used for NLP to process the user's input text.
states	Custom built module used to check USA state abbreviations.
struct	Used to communicate between scripts by reading & writing to standard streams[58].
sys	Used predominantly to access standard streams (i.e. stdin & stdout).
time	Used to create delays in scripts to account for asynchronous events.

Table 4: Python Libraries

Other Libraries

Library Name	Description
jQuery (JS)	Used to interact with DOMs & create functionality on the web page context.
OpenSSL	Offers tools for working with the Secure Sockets Layer (SSL) Transport Layer Security (TLS) protocols. In this project its cryptography capabilities were used to create a hash of a JS script. For more details, read 'Make WF Main Display' (section 7.1.2.7).

Table 5: JavaScript Libraries

[58] For more details on this, see the section 'Connect Host (Native) Application' (section 7.1.2.3).

6.3.4 APIs

The following APIs were used to create this extension. For a list of the specific functions from these APIs, see the appendix, 'API Functions' (section 13.3).

JavaScript

API	Description
chrome.extension	Provides general functions that "can be used by any extension" context, such as methods for message passing (Google, 2019k).
chrome.contextMenus	Gives access to the browser's right-click context menu, enabling the adding of items to it, and allowing developers to specify the contexts in which the items appear (i.e. only when text is highlighted).
chrome.tabs	Allows interaction with the browser's tab system to "create, modify, and rearrange tabs in the browser." (Google, 2019l).
chrome.runtime	A multipurpose API for accessing the extension's background page, returning details about the manifest, listening & responding to events within the extension lifecycle, and converting relative paths to URLs (Google, 2019l).
JavaScript Maps API (URL)	From the Maps JavaScript API (Google, 2019s). Used to create and manipulate a Google map object. The API is loaded using a HTML 'script' tag with an attribute specifying an endpoint where the required JS file can be fetched. The URL was: https://maps.googleapis.com/maps/api/js?key=<userID> &libraries=drawing,places&callback=initMap **Key:** Is the users unique ID (The ID itself has been omitted). **Libraries:** Requests for Google Maps libraries, of which there are several. For this project, 'drawing' was used to add map markers, and 'places' was used to access data on specific locations (i.e. latitude and longitude coordinates) (Google, 2019u). **Callback:** The name of the callback function that you wish to use to execute the API's functions.

Table 6: APIs used in this project.

Python

API	Description
wikipedia	Python library built on the MediaWiki API. Used to send HTTP requests to Wikipedia.
msrest.authentication	Bing API used to validate the identity of the HTTP request sender so that authorized requests to Bing's Search APIs can be made.
azure.cognitiveservices.search	Bing API used to access endpoints for Bing's news search requests and image search requests.
Place Search (URL)	Endpoint from Google's Places API. Used to determine the 'Place ID'[59] of a particular location. The request to the URL was sent using Python's 'request' library. The endpoint the request was sent to was: https://maps.googleapis.com/maps/api/place/findplacefromtext/<userID>

Table 7: Python API's used in this project.

[59] A 'Place ID' is essentially a unique ID given to every known place within Google Maps (Google, 2019t).

7 Project Timeline

The following Waterfall timeline was devised to complete this project:

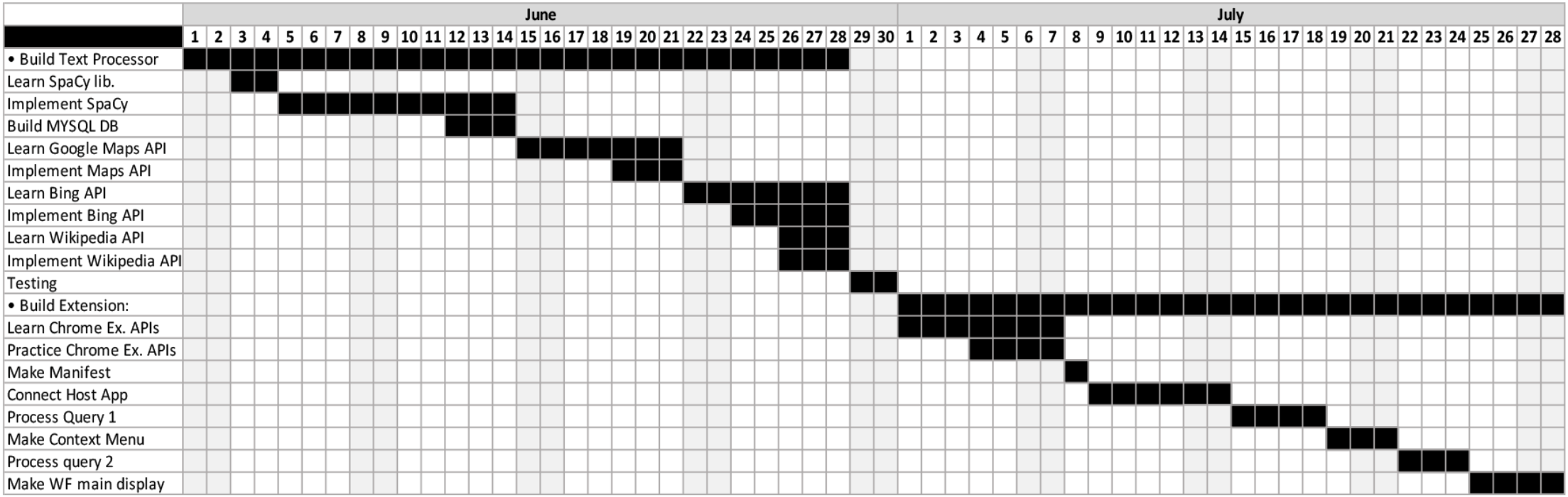


Table 8: Project Timeline

7.1 Timeline Breakdown

The particulars of each phase in the timeline will be detailed below. This lower-level discussion of the code references only code that is necessary to understand how the functionality of each phase has been achieved. Less crucial code is omitted. For a more detailed review of the software's code, see the extensive commenting within the code itself. Or, for a higher-level review of the software architecture, see the section, 'Product Architecture' (section 8).

7.1.1 Phase 1: Building the Text Processor

All of the text processing was done using a local host application written in Python. Writing the text processing in a different language was decided for two reasons:

1. To leverage different NLP libraries than what was available in JS[60].
2. To keep that text processing separate from the rest of the extension, so that the JS scripts can focused on browser-related functionality such as context menu functions, injecting scripts, creating/displaying HTML, and message passing across the extension.

Additionally, the language processor was built using an object orientated programming (OOP) approach to make the text processor more modular. Using the text processor's functions grouped together became particularly useful later in the project.

7.1.1.1 Learn & Implement SpaCy

Learning SpaCy was a quick process because of its excellent documentation and tutorials. In implementing SpaCy, there were several key considerations:

[60] Some JS libraries were considered, such as wink.js, NaturalNode, and nlp-compromise.

1. Choosing an NLP language model

SpaCy can be loaded with many different language models. For this extension, the two models chosen were ‘en_core_web_lg’, and ‘en_core_web_sm’ – the latter model, is smaller and therefore processes text much quicker than the former. However, the larger model labels NE’s correctly more often, so both models were used throughout development, depending on the need at that particular time (fig. 17).

```
import spacy
nlp = spacy.load('en_core_web_sm') # 'en_core_web_lg'
```

Figure 17: Code from 'langproFile.py'.

2. Checking length of input text

To ensure that the user isn’t sending an excess of text to the text processor[61], an additional process was added to the NLP pipeline which checked the number of words being processed. This function will halt the text processing if more than 20 words are sent to the text processor. This simple function is added to the very beginning of SpaCy’s NLP pipeline (fig. 18).

```
def lengthChecker(doc): …
spacyInstance.add_pipe(lengthChecker, first=True)
```

Figure 18: Code from 'langproFile.py'.

3. Creating a dictionary to store data about the input text

As input text is processed, any information gathered required a place to be stored. For this, a ‘location dictionary’ was initialized at the beginning of the text processer class (fig. 19).

```
self.loc = {'GCS':[], 'label':'', 'input':'', 'locBase': '','ownLoc': geocoder.ip('me').latlng, 'runQueryU':
False, 'DBSearchResults': []}
```

Figure 19: Code from 'langproFile.py'.

[61] This could cause ambiguity issues on which information should be extracted.

4. Extracting NEs

Another function was designed to check which type of NE had been extracted from the text (fig. 20). If an appropriate NE was available[62], then the relevant information was added to the location dictionary ('self.loc').

```
def locationGetter(self, doc):…
```

Figure 20: Code from 'langproFile.py'.

5. Checking for latitude and longitude coordinates

Latitude and longitude coordinates represent a location and thus it was important to ensure that this extension could process them as it would any other location. However, latitude & longitude coordinates aren't extracted as a NE using the NLP techniques employed thus far in this project. Therefore, a function that would use regular expressions to check if the input text contained latitude & longitude coordinates was added (fig. 21).

```
def GCSChecker(self, txt):…
```

Figure 21: Code from 'langproFile.py'.

[62] The only appropriate SpaCy NE's for this extension are location-based ones, like 'LOC' and 'GPE'

7.1.1.2 Build MYSQL DB

During this phase, a few of the larger tasks were:

1. **Setting up a server with MYSQL**

XAMPP was used to run a server with a MYSQL DB. XAMPP provided a UI accessible via the localhost (127.0.0.1) URL (fig. 22).

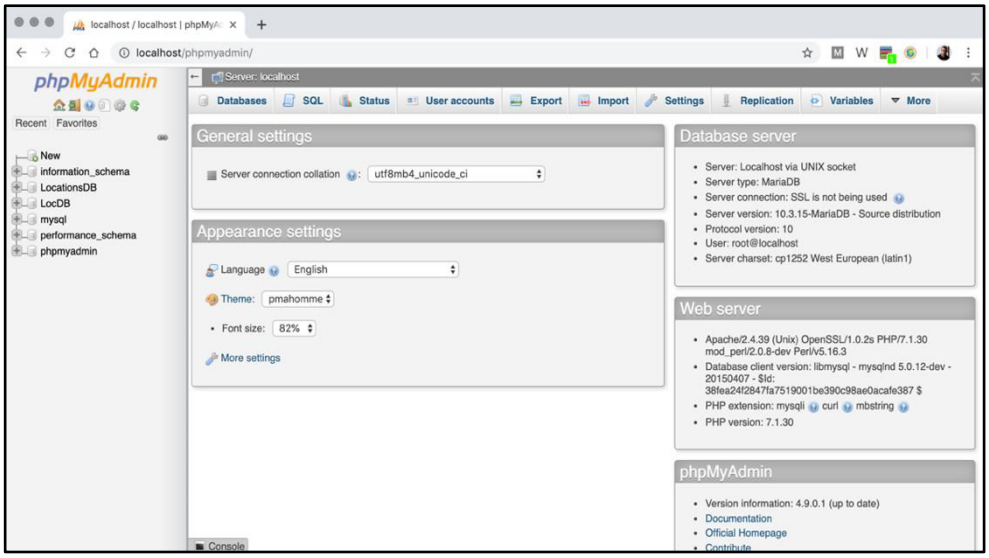


Figure 22: XAMPP DB UI

On this server, a DB table was created and populated using a SQL script of locations (Google, 2019z) called 'lokasyon.sql'. This script was the largest free collection of location names found. More robust location DB's are available (i.e. (SimpleMaps, 2019), (GeoDataSource, 2019)), but to download their contents would incur costs that extend beyond the fiscal limitations of this project.

2. **Establishing a connection between Python & MYSQL**

Checking a user's input text against the locations DB required creating a method within the Python text processor class for establishing communication with the DB (fig. 23). This was done using the Python library, 'PyMySQL'.

```
def connectDB(self):
    connection = pymysql.connect(…)
    return connection
```

Figure 23: Code from 'langprofile.py'.

3. Retrieving/Processing query results

Several functions were required to; retrieve the DB query, evaluate what to do with the search results, process/store the results, and to generate a list of the results (fig. 24).

```
def pyMySQLSearch(self): …
def evalResults(self, searchType): …
def processResults(self, results): …
def resultsListGen(self): …
```

Figure 24: Functions from 'langproFile.py'.

7.1.1.3 Learn & Implement Google Maps API

A couple of the challenges in this phase involved:

1. Learning the Google Maps API

After some investigation it was determined that the only information that the text processor should gather from Google Maps would be a location's 'PlaceId'. This id created by Google would be used later when generating the Google Maps display in JS. Conveniently, requests for PlaceId's can be made to Google for free (Google, 2019aa), although users of Google's APIs still need to create a Google Cloud Platform account and use their unique user ID key when sending requests to any Google APIs (fig. 25).

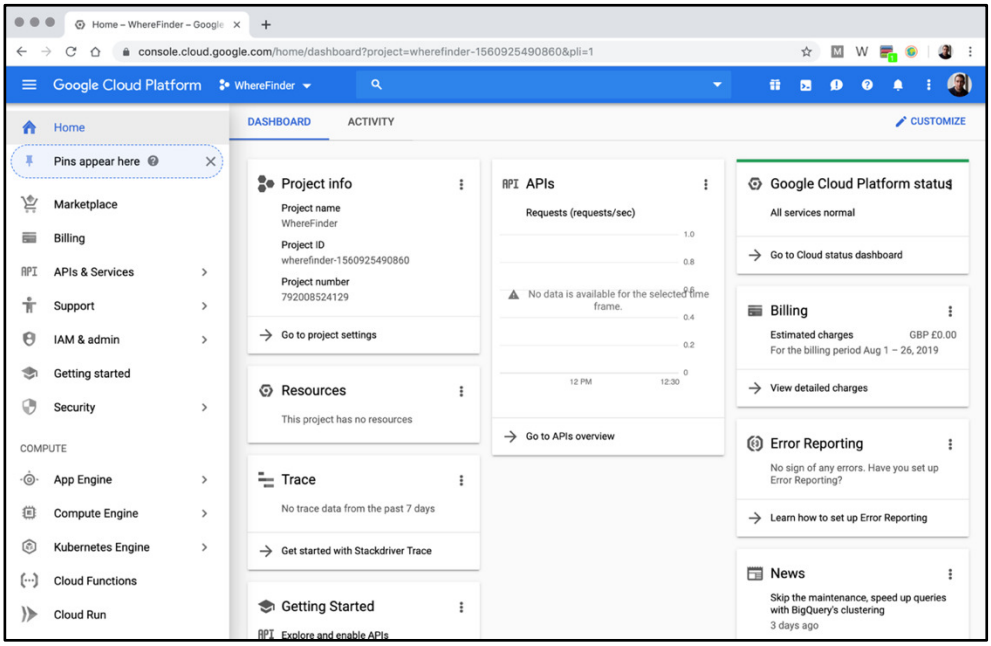


Figure 25: Google Cloud Platform Developers UI

2. Making HTTP Requests for PlaceIDs

To make a request to Google, three functions were made; two to create a request URL, and one to send the request using Python's 'requests' library (fig. 27, fig. 28).

```
https://maps.googleapis.com/maps/api/place/findplacefromtext/json?inputtype=textquery&key=
<PrivateUserKey>&input=<inputQuery>
```

Figure 26: URL to Google PlaceID API

```
def prepParams(self): …
def prepURL(self): …
def placeIDFetch(self): …
```

Figure 27: Functions from 'langproFile.py'.

7.1.1.4 Learn & Implementing Bing API

1. Learning the Bing API

Using Bing APIs requires a Microsoft Azure account (fig. 28), which comes with $200 of free requests to their APIs (Microsoft, 2019g). Bing API endpoints can be reached by specific URL endpoints, but can also be accessed via specific 'Bing requests' libraries built for specific programming languages. To make requests to Bing, their Python API libraries were leveraged (fig. 29).

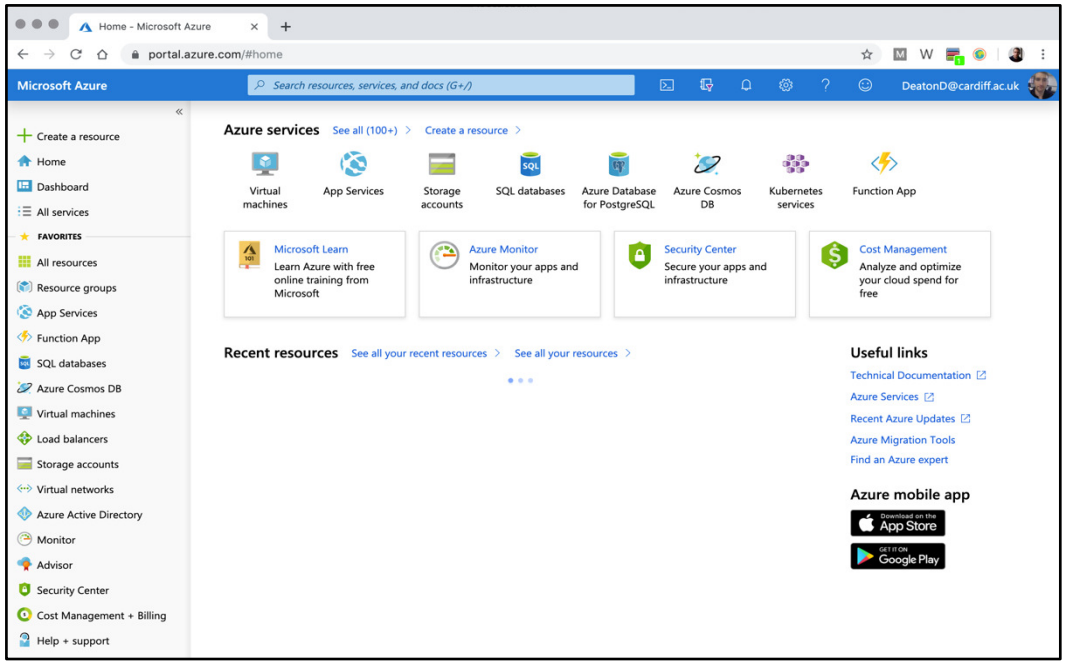


Figure 28: Microsoft Azure Platform Developers UI

```
from azure.cognitiveservices.search.newssearch import NewsSearchAPI
from azure.cognitiveservices.search.imagesearch import ImageSearchAPI
```

Figure 29: Code from 'langprofile.py'.

2. Implementing Bing News & Images APIs

Two separate functions were built to request news articles and images from Bing (fig. 30).

```
def newsFetch(self):…
def imagesFetch(self):…
```

Figure 30: Functions from 'langprofile.py'.

7.1.1.5 Learn & Implement Wikipedia API

This phase was relatively straight forward because rather than working directly with the WikiMedia APIs (Wikimedia Foundation, 2019), it was decided that a faster approach would be to leverage a Python library built to perform requests to Wikipedia (PyPI, 2019). Leveraging this library was particularly useful because it allowed development to move quickly on to the more challenge phase of this project (building the extension). All that was required to perform requests to Wikipedia, was a single function (fig. 31).

```
def wikiFetch(self):…
```

Figure 31: Function from 'langproFile.py'.

7.1.1.6 Testing Build

For details on testing, please see the section, 'Integration Testing' (section 9.1.1).

7.1.2 Phase 2: Building the Extension

The second phase of this project focused on developing the functionality outside of the text-processing.

7.1.2.1 Learn & Practice Chrome Extension APIs

During this phase, a number of small extensions were built to gain a familiarization with the systems that underpin browser extensions. The knowledge gained in this phase was explored in the section, 'Web Browser Extensions' (section 3.1).

7.1.2.2 Make Manifest

The first step in creating any extension is making a manifest file. The manifest file of this extension was continually updated throughout this development phase as the project progressed. An explanation on each of the objects within this extension's manifest file can be found in the section 'How Extensions Work: Manifests' (section 3.1.3).

7.1.2.3 Connect Host (Native) Application

The script's that communicate with each other, are the 'background.js' file, and the 'langpro.py' file. To establish a connection between these scripts required overcoming a number of challenging hurdles.

1. Establishing a connection

In order to initiate a connection with a local application (i.e. the Python text processor), the local application must have its own manifest file created for itself and placed into Chrome's directories.[63] Then, a Chrome API can be used, referencing the name of the native application's manifest, which Chrome then looks for in its own directories, and uses to establish a connect with the application (fig. 32).

```
function initPort() {
    port = chrome.runtime.connectNative('com.langpro');
};
```

Figure 32: Code from 'background.js' file.

[63] For the precise location, read the 'Product Architecture' description of the 'WhereFinder Host Application Manifest Directory'.

2. Receiving/sending a message to/from Chrome

The only communication protocol that Chrome supports for native message passing is stdin/stdin standard streams. Therefore, a Python script was created to handle communication between the text processor and the Chrome browser, called ‘langpro.py’. When Chrome establishes a connection with ‘langpro.py’, it reads each line in the file through until it hits a Boolean loop created within the ‘langpro.py’ file (fig. 33).

```
while True: …
    try: …
    except: …
```

Figure 33: Code from 'langpro.py' file.

This loop then uses the ‘stdin’ and ‘stdout’ functions within the ‘sys’ library in order to read/write data to/from the stream. Most of these the sys methods are contained within two functions (fig. 34); one towards the start of the loop to read incoming messages, and one at the end of the loop to write outgoing messages.

```
def unpackMsg(incom_msg_bytes): … # to read
def sendMsg(msgDict, msgTypeIn): … # to write
```

Figure 34: Functions from 'langpro.py' file.

3. Sending/Receiving a message to/from the host application

On the JS end, the Chrome APIs abstract away some of the more technical steps of reading and writing to the standard stream. Three functions handle the majority of this communication (fig. 35).

```
function sendWFData (WFData) {
    port.onMessage.addListener(portListener);
    messageOut = messageProcessor(WFData,'out_req');
    port.postMessage(messageOut);
};
function portListener(msg) { … };
```

Figure 35: Functions from 'background.js' file.

7.1.2.4 Process Query 1

With the scripts connected, the next step was to implement the first half of the Python text processor's methods; running NLP, and using the extracted NE to find any location matches within the locations DB.

1. JSON object

As this extension passes messages between the content script, the background script, and the python script, more information is gathered at each stage. To keep track on of this information, a single JSON object is passed around, and any new data is appended to it. For a full breakdown of the data gathered by this JSON object, see the appendix, 'JSON IR Object' (section 13.1).

2. Sending input text, receiving processed text

As mentioned, when a message is sent to the native application, it contains a JSON object. This object contains information that the native application uses to choose whether to execute functions associated with the first, or second, phase of the text processing (fig. 36).

```
def processMsg(incom_msg_dict, msgType):
    if msgType == 'out_req': …    # processing phase 1
    if msgType == 'out_query':    # processing phase 2
```

Figure 36: Function from 'langpro.py' file.

Then, using the text processor made in phase 1, an instance of the text processor class is created to process the incoming text. The text processor class's functions for NLP, and querying the locations DB, are called (fig. 37).

```
processedText = langpro(incom_msg_txt) # creating instance of processor class
processedText.initNLP()
processedText.initDBSearch( … )
```

Figure 37: Code from 'langpro.py' file.

Data gathered from processing the text is added to the JSON object, which is passed back to the 'background.js' script.

7.1.2.5 Make Context Menu

Working with context menus required a number of steps:

1. Adding the extension to the Chrome context menu

This required that some information about the button be passed into a specific Chrome API that handle's changes to its context menu (fig. 38).

```
var contextMenuItem = {
    "id": " "WF", // ID of item
    "title": "WhereFinder", // Name to appear on button
    "contexts":["selection"] // When to appear (i.e. when text is selected)
};
```

Figure 38: Code from 'background.js' file.

Then, an event listener is added to the browser so that if the extension button is pressed, a series of functions are executed which pass on the selected text, and the mouse position[64] of the context menu event, to the text processor (fig. 39).

```
chrome.contextmenus.onClicked.addListener((clickData) => {
    if (clickData.menuItemId == "WF"){
        …  // functions for passing relevant data to text processor
    };
```

Figure 39: Code from 'background.js' file.

2. Creating WF's own context menu

Once the extension has done the initial processing, any potential matches are sent back to the extension's 'background.js' script, where the results are processed by two functions; one function to create the list of options, and one generator to generate the formatted text for each option (fig. 40).

[64] Acquiring the mouse position required additional communication directly with an instance of the 'contentScript.js' file running directly on the web page. JavaScript Promises were instrumental in this stage. Please see the code for the specifics.

```
function prepUsrQry(msg) { … }
function* prepSrcRes(searchResults) { … }
```

Figure 40: Code from 'background.js' file.

Then, the HTML & CSS for a context menu is generated and injected into the current web page at the same mouse position as the original context menu event (fig. 41).

```
function WFContextMenu(queryOptions, tabId, mouseX, mouseY) { … }
```

Figure 41: Code from 'background.js' file.

A promise is then made (fig. 42) which waits for the user to make a selection from the context menu. If this promise is fulfilled, the background.js script will then pass the users selection back to the native application, asking the Python script to gather additional information about the text.

```
waitForMenuSelection(msg.textObj)
    .then( … )  // send user select to host application
    .catch(     ) // handle rejection
```

Figure 42: Code from 'background.js' file.

7.1.2.6 Process Query 2

Now, after the user selection, there is no ambiguity about which location the user is looking for information about. At this stage, the native application calls two functions from the text processor class which execute a series of API requests to Google, Bing, and Wikipedia, in order to gather more information about that specific location (fig. 43).

```
processedText.initGoogleAPI()
processedText.initOtherAPIs()
```

Figure 43: Code from langpro.py' file.

This information is then passed back to the ‘background.js’ script.

7.1.2.7 Make WF Main Display

Using the information gathered, the 'background.js' script then runs two JS 'async functions'[65] which each execute a series of promises.

1. Displaying the extension main window

The first function (fig. 44), executes four promises that inject the static HTML elements of the extension display into the web page and add JS functionality to the extension display.

```
async function WFDisplayMainDisplay(tabId, textObj)
```

Figure 44: Code from 'background.js' file.

2. Displaying the information retrieved from the input text

The second function (fig. 45), executes five promises which dynamically generates and injects HTML & CSS based on the information that's been gathered about the location, into the web page.

```
async function WFDisplayContent(tabId, textInfo)
```

Figure 45: Code from 'background.js' file.

3. Displaying the Map

In the second function, one of the promises that's executed is responsible specifically for the map display. During this promise, an Iframe is injected into the current web page which creates a 'view' into a separate web page, 'WFMapIframe.html', where the JS for the Google Maps display resides. Information to be displayed by the 'WFMapIframe.html' document (the location of interest, and the users current location) is passed into to the 'WFMapIframe.html' file via the Iframe element in the parent window, using the Iframe element's 'src' attribute, which allows the passing of a query at the end of the URL (fig. 46).

```
<iframe src="chrome-extension://inpfiiifhajlhicplcogoacfnmkmgpen/html/
WFMapIframe.html?placeId=<GooglePlaceID>;ownLoc=<OwnLat&Lat>"></iframe>
```

Figure 46: Code injected into a web page by 'background.js' file

[65] In JS, async functions return AsyncFunction objects and allow for special JS functionality (i.e. the await function).

Then, within 'WFMapIframe.html' a series of JS functions use the query passed to the document from the parent window to generate the Google Maps display (fig. 47).

```
<script>
function initMap () {
    queryStr = location.search.toString(); // retrieve src query
    …           // functions to display map
};
```

Figure 47: Code from 'WFMapIframe.html' file.

4. Inline JavaScript

One interesting challenge faced towards the end of building this extension was handling the inline JS written for 'WFMapIframe.js' script. Chrome extensions rely on a Content Security Policy (CSP) in order to mitigate a large class of potential cross-site scripting issues (Google, 2019bb) and one of the restrictions that browser extension CSPs create is a ban on inline JS. This proved problematic when writing the 'WFMapIframe.js' script, which used inline JS. The solution to this problem was to relax the extension's CSP via the extension's manifest file, within which the 'WFMapIframe.html' inline script was added to a list of permitted scripts (fig. 49). For security, the script must be identified within the manifest using a base64-encoded hash string of the script. To create this string, the script was encrypted with the 'OpenSSL' library (OpenSSL, 2019), using the following terminal command (fig. 48):

```
echo -n "<inlineScript>" | openssl dgst -sha256 -binary
```

Figure 48: Terminal code used to hash the inline script code.

Then, the outputted hash string was added to the extension CSP via the extension's manifest:

```
"content_security_policy": "script-src 'self' https://maps.googleapis.com 'sha256-
G3MXsE+rHzFcPab0T2fKKemDfZEKP8rHqurNcrcAlAo='; object-src 'self'"
```

Figure 49: Hash key of inline script put into the extension's manifest.

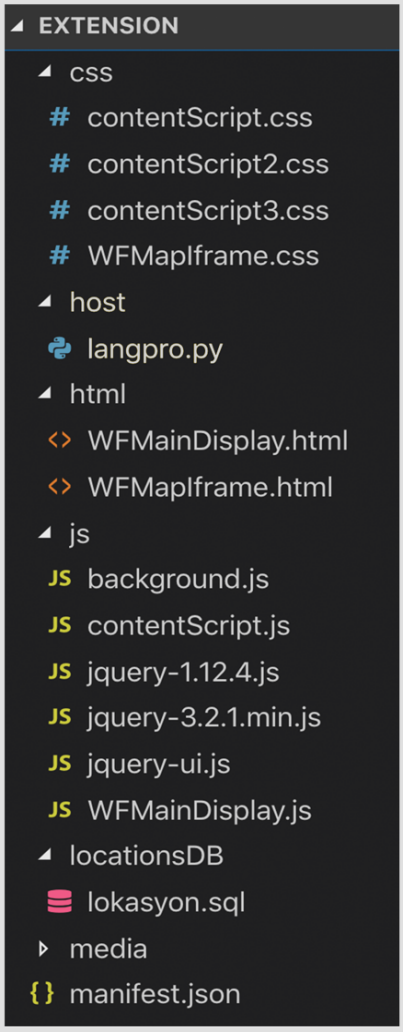
After this, the inline 'WFMapIframe.js' script was able to executed.

8 Project Architecture

This section overviews the location and contents of each directory within this extension (fig. 50, table. 9).

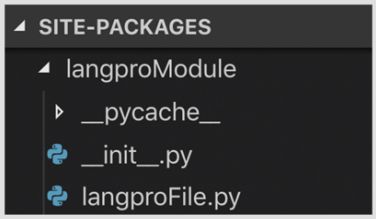
WhereFinder Main Directory:

/Users/Drake/Desktop/Dissertation/application/project/extension



WhereFinder Host Application Python Module DIrectory:

/Users/Drake/anaconda3/lib/python3.6/site-packages/langproModule/langproFile.py



WhereFinder Host Application Manifest Directory:

/Users/Drake/Library/Application Support/Google/Chrome/NativeMessagingHosts

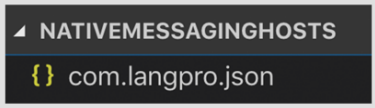


Figure 50: Directories overview.

Directory	Module Name	Description
extension/css/	contentScript*.css	Provides some of the style information for any content scripts that are injected into web pages.
	WFMapIframe.css	Provides the style information for the WhereFinder Map Iframe.
extension/host/	langpro.py	This script is a host application that communicates with the background.js script using standard streams. The langpro.py file is sent JSON objects that contain, amongst other data, the user's input text. The 'lanpro.py' file handles the text processing using the 'langproFile.py' module, and then returns a message back to the 'background.js' file.
extension/html/	WFMainDisplay.html	This HTML file contains all the static presentation elements of the WhereFinder main display window. It is injected into a web page via a function in the 'contentScript.js' file, triggered via the 'background.js' script.
	WFMapIframe.html	This HTML file contains the presentation elements and Google Maps API methods for displaying the WhereFinder's map. A 'view' of this document is created via an Iframe[66] element on another web page. Its injection is done via the 'background.js' script.
extension/js/	background.js	This JS script is responsible for: listening for browser level events (context menu clicks, message passing…), communicating with the host application, creating dynamic HTML, and injecting scripts into web pages.
	contentScript.js	This JS script is responsible for listening to events on web pages (context menu events, message passing), and HTML injection into web pages.

[66] For more information on Iframes, read the section, 'Make WF Main Display' (section 7.1.2.7).

	jquery-*.js	Files to provide access to jQuery. These scripts are injected into web pages via the 'background.js' script.
	WFMainDisplay.js	This JS file uses jQuery to add some additional functionality to the WhereFinder main display window, including adding the ability to; adjust the position of the entire display, close the window, and change which tab (i.e. map, info, news, images) within the WhereFinder display is currently displayed.
extension/ locationsDB/	lokasyon.sql	Contains a table of locations which has been executed on a MYSQL server.
media/	*	Contains any multimedia used in the extension. Accessed by the manifest and several HTML documents.
extension/	manifest.json	Provides information about the extension used by the Chrome browser and by the Chrome extension[67].
Site-Packages/ langproModule/	__init__.py	This empty file is required by Python in order the directory to be recognized as a module.
	langproFile.py	A custom Python module containing a class which has all the necessary functions to perform the text processing that this extension requires. These functions include; NLP, IR using a DB of locations, IR via HTTP requests (Google Maps, Wikipedia, Bing), and Getter & setter functions.
NativeMessagingHosts/	com.langpro.json	This is a manifest file specifically for the host application that describes how Chrome and the host application will communicate. A Chrome API is used in 'background.js' which informs Chrome of the name of this manifest, which Chrome then looks up in the Chrome directory, 'NativeMessageHosts'.

Table 9: Description of all the files within the Directory overview.

[67] For more information, see the section 'How Extensions Work: Manifests' (section 3.1.3).

8.1 UML Diagram

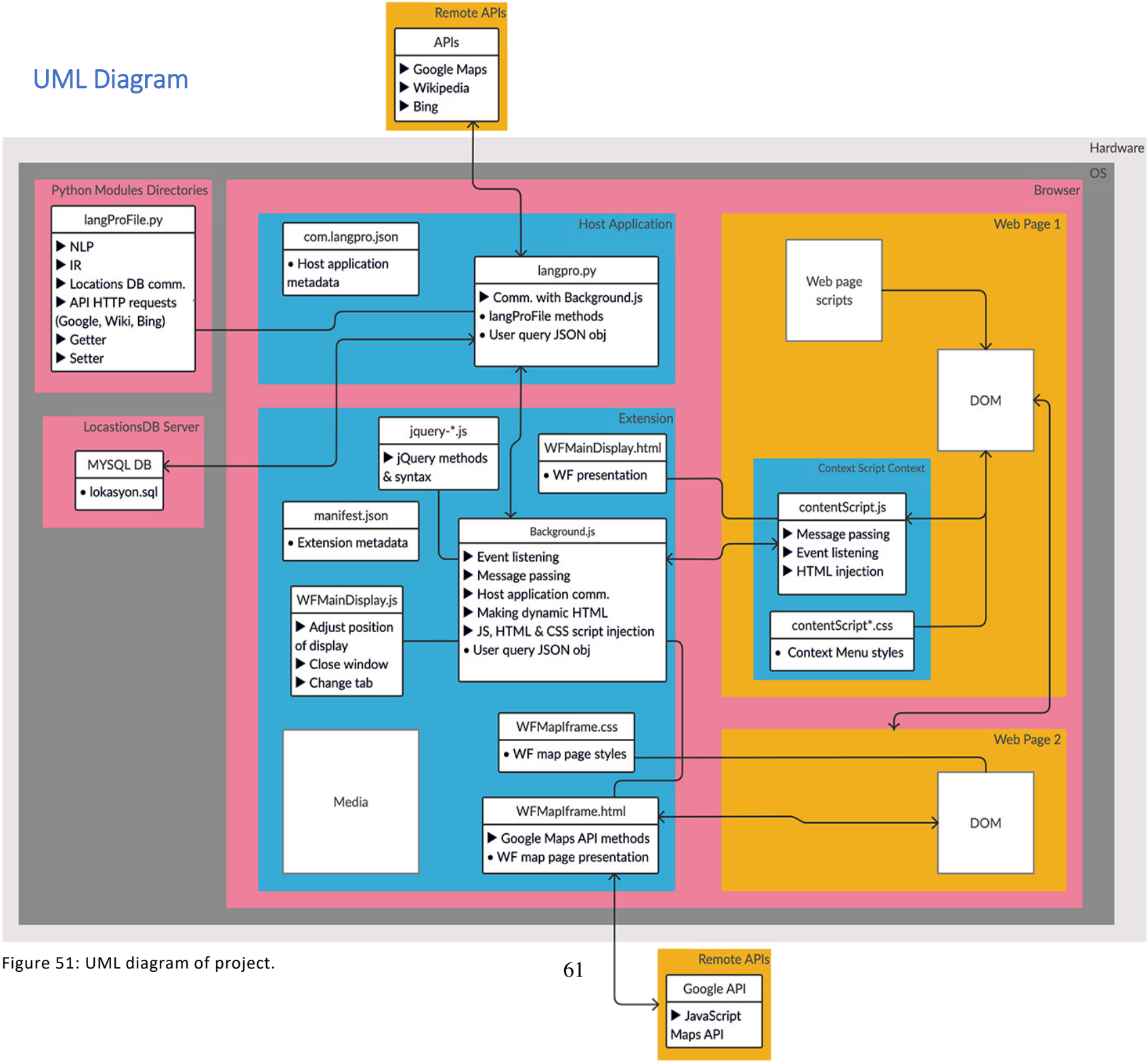


Figure 51: UML diagram of project.

8.1.1 UML Key

Underneath the above Unified Modelling Language (UML) diagram (fig. 51), the contexts in which the scripts are run has been also been illustrated. To better understand these contexts, a key is provided below in alphabetical order (fig. 52):

Context	**Description**
Browser	The browser context is that of a software application running on top of an OS.
Extension	Chrome reads an extension's manifest.json file and then runs the extension in parallel with the rest of Chrome's software – this is the extension context. Chrome creates a unique ID for all extensions and a unique URL where its directories can be reached. This extension's URL is: "chrome-extension://inpfiiifhajlhicplcogoacfnmkmgpen/"
Hardware	The hardware context is lowest level of the machine upon which higher levels of functionality are built (OS, software applications…)
Host Application	When communication is initiated with a host application installed locally on the user's computer[68], Chrome looks for a manifest for the native application, and then establishes a context for communication the native application, so that communication can occur.
LocationsDB Server	This a local sever running a MYSQL database.

[68] For more information, read the description of 'com.langpro.json' from the module descriptions above.

OS	This context sits logically on top of the hardware, but beneath software applications.
Python Modules Directories	When importing python modules into a script, Python will check certain directories for the module(s) in question – this context refers to those directories. To import 'langProFile.py', the directory used was: /Users/Drake/anaconda3/lib/python3.6/site-packages/langproModule/
Remote APIs	This context refers to the remote contexts accessed over the cloud to leverage certain APIs, specifically those of Google, Wikipedia, and Bing.
Web Page 1	This context refers to the web pages loaded by Chrome, which the extension is interacting with.
Web Page 2	This context refers specifically to a web page loaded for the 'WFMapIframe.html' file. An Iframe is an HTML element which provides a 'view' into another web page. Thus, when the WhereFinder main display is loaded onto a webpage, the Google Maps display that is visible is in fact a window the provides a view into a 'child window', in which is a separate web page.

Figure 52: UML diagram table describing the different contexts of the scripts

9 Project Testing

Testing “assesses the quality of a system,” to help “manage the risks to [a] system[‘s] quality” (Black, 2003). In this chapter’s two sub-sections ‘Software Testing’(section 9.1), & ‘Usability Testing’(section 9.2), the testing conducted on this project will be delineated.

9.1 Software Testing

Software testing can be explained as “the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component” (IEEE, 1990). The software testing for this project was divided into ‘Integration Testing’ (section 9.1.1) and ‘Test Cases’ (section 9.1.2).

9.1.1 Integration Testing

The following testing was performed on the ‘langproFile.py’ module. This integration testing was performed to ensure that the functions of the ‘langproFile.py’ module worked together consistently.

1. First, a group of test text inputs were created (fig. 53).

```
test1 = "I love New York.'
test2 = "It's always raining in Manchester."
test3 = "Tokyo 101."
test4 = "The beaches in Argentina are incredible."
test5 = "Tourism is Barcelona is on the rise."
```

Figure 53: Integration testing code part 1.

2. Second, each input text was tested by creating an instance of the ‘langpro’ class (fig. 54).

```
for i in testLi:
    test = langpro(i)
```

Figure 54: Integration testing code part 2.

3. Then, each instance of the ‘langpro’ class executed a series of tests designed to test each of the groups of functions that had been integrated together. Starting with the class’s ‘__init__’ function (fig. 55).

```
# Checking Variables exist that __init__ should’ve initiated.
# Expecting all the vars below to have been assigned

try:
        vars = [test.loc, test.checker, test.degRE, test.charCountRE,
        test.labelCheck, test.DBDict, test.searchResults, test.DBSearch,
        test.SQLRegEx1, test.SQLRegEx2, test.SQLRegEx3, test.con,
        test.bingNewsClient, test.bingImagesClient, test.wikiReg ]
        for var in vars:
      assert var is not None
   assert test.numOfResults is None
   print('1. vars - OK')
```

Figure 55: Integration testing code part 3.

4. Then, the ‘initNLP’ function (fig. 56).

```
# Checking initNLP works
# Expecting content to be extracted from input text into the 'loc'
# dict using the functions:
#    nlp, GSCChecker, locationGetter

try:
    originalLoc = test.loc.copy()
    test.initNLP()
    updatedLoc = test.loc
    if originalLoc == updatedLoc:
        raise Exception
    print('2. initNLP - OK')
```

Figure 56: Integration code testing part 4.

5. Then the ‘initDBSearch’ function (fig. 57).

```
# Checking initDBSearch works
# Expecting that potential location matches are fetched from
# locations DB using:
#   connectDB, pyMySQLSearch, evalResults, processResults

try:
    test.initDBSearch(test.loc['locBase'])
    if test.loc['DBSearchResults'] == None:
        raise Exception
    result = test.loc['DBSearchResults'][0]
    test.loc['inputFull'] = result['localName'] + ', ' + result['iso']
    test.loc['iso'] = result['iso']
```

Figure 57: Integration testing code part 5.

6. Then, the ‘initGoogleAPI’ function (fig. 58).

```
# Checking initGoogleAPI works
# Expecting that a placeID is fetched from Google API using:
#   prepParams, prepURL, placeIDFetch
try:
    test.initGoogleAPI()
    if type(test.loc['placeID']) != str:
        raise Exception
    print('4. initGoogleAPIs - OK')
except Exception as e:
```

Figure 58: Integration testing code part 6.

7. Last, the ‘initOtherAPIs’ function (fig. 59).

```
# Checking initOtherAPIs works
# Expecting that each li item to contain content from IR functions:
#   wikiFetch, newsFetch, imagesFetch
try:
    test.initOtherAPIs()
    li = [test.loc['wiki'], test.loc['news'], test.loc['images']]
    for i in li:
        if i == None:
            raise Exception
    print('5. initOtherAPIs - OK')
except Exception as e:
    print(e)
```

Figure 59: Integration testing code part 7.

8. For each time this script was run, the following results were printed (fig. 60):

```
1. vars - OK
2. initNLP - OK
3. initDBSearch - OK
4. initGoogleAPIs - OK
5. initOtherAPIs – OK
```

Figure 60: Integration testing code output.

To run additional tests, all that's required is a changing the input text in the first step. These results helped demonstrate that the 'langproFile.py' module could handle processing texts of many different varieties, and that the functions worked together effectively.

9.1.2 Test Case

A test case "is a definition of input values and expected output values for a unit under test" (Christensen, 2010). Using test cases can help determine if a system works properly and fulfils its requirements. The following test case was created for this project to ensure that it could execute the functionality it has been designed for successfully under test conditions (table. 10):

Test Case Id: 1.	**Test Case Name:** Testing the normal functionality of the WhereFinder extension.	**Test Purpose:** Demonstrate that the WhereFinder extension is able to take in input text and return supplementary content.	
Environment: Google Chrome web browser.			
Preconditions: The participant has an opened instance of the Chrome web browser, with the WhereFinder extension running.			
Test Case Steps: Basic Flow.			
Step No	**Procedure:**	**Response**	**Pass/Fail**

1	Right click within the Chrome browser to bring up its context menu.	The context menu should be visible, but the 'WhereFinder' extension will not be present.	Pass
2	Find highlight-able geographical text on a web page, highlight the geographical text, and right click on the highlighted text to bring up Chrome's context menu once again.	The context menu should be visible, and the 'WhereFinder' extension should now be present as one of its clickable options.	Pass
3	Click on the 'WhereFinder' box within the context menu.	The context menu should be replaced and a second context menu should appear with a list of potential matches for the geographical text that was highlighted.	Pass
4	Click on the box with the location that more supplementary content is required on.	The context menu should disappear and the WhereFinder main display window should appear within web page's display.	Pass
5	Navigate between the four WhereFinder tabs by clicking on each tab.	Each time a tab is clicked, the display should change to the new tab.	Pass
6	Navigate to the 'Map' tab an interact with the map: zoom in & out, click and drag around the map.	The map display should react to the interactions of the participant.	Pass
7	Navigate to the 'Info' tab and click on the title of the wikipedia page.	The summary from the Wikipedia page will be visible within the WhereFinder 'Info' tab, and by clicking on the title, a new tab will open to the entire Wikipedia page.	Pass
8	Navigate to the 'News' tab, scroll through the articles, and click on any one of their titles.	Each article will have information pertaining to the article in its own self-contained display. When an article title is clicked, a new tab will be opened to the location of the entire article.	Pass

9	Navigate to the 'Images' tab, scroll through the images, and click on any one of them.	Each image will be displayed in their own box. When an image is clicked on, a new tab will be opened to the location of the image.	Pass
10	Click the top bar of the WhereFinder box and drag the WhereFinder box around the web page.	The WhereFinder box is draggable and should move around the web page with the mouse.	Pass
11	Click on the circular button in the top right corner.	Clicking this button will close the WhereFinder box.	Pass
Related tests: The participant may take this test multiple times across different websites to ensure that the extension continues to function normally.			
Author: C1873019		**Checker: C1873737**	

Table 10: Test Case for the WhereFinder extension.

9.2 Usability Testing

Usability testing (UT) is the process of "observing users working with a product, [and] performing tasks that are real and meaningful to them" (Barnum, 2011). This is a crucial part of developing a product because when you "focus on the user and not the product, you learn what works for your users." Ultimately, the success of a commercial project is determined by its consumers; as is the case with this project – it's the users that will determine whether or not this project improves the usability of the web browser.

Based on the short development schedule of this project, further development will be needed before this project could see a commercial release. As such, the UT of this extension at this stage can be viewed as 'formative testing' whose results will help dictate the future direction of this software's development.

For the UT of this extension, this project has leveraged the NASA Task Learning Index (TLX), which is a "multi-dimensional scale designed to obtain workload estimates from one or more operators while they are performing a task" (Hart, 2006). The NASA-TLX has been chosen because of its established history as a method for testing computer users (Hart, 2006). Using the NASA-TLX, the "workload" that the test's participants judge the using of WhereFinder to require will provide insight into the usability of the

extension. In essence, the more demanding participants perceive the workload, the lower the usability of the extension.

The tests created utilize a simplified method of administering the NASA-TLX, known as the 'Raw-TLX' (RTLX)[69]. Some studies have shown the RTLX to be superior to the original NASA-TLX (Hendy, 1993), while other studies have shown the opposite (Liu, 1994), so by supposition, using the simplified method in this instance should still provide equally useful results.

To view the RTLX tests created for this project's UT, see the appendix section 'RTLX Test' (section 13.4). Otherwise, the following sections will provide a 'Test Overview' (section 9.2.1), the 'Test Results' (section 9.2.2), and the 'Test Conclusions' (section 9.2.3).

9.2.1 Test Overview

The RLTX test conducted for this project gave 10 participants a series of small tasks to complete, after which they were required to answer a short questionnaire in which participants were asked four questions about their immediate thoughts on the tasks they had just completed. Each of these questions covered a broad category[70]:

1. **Mental Demand:** How much mental and perceptual activity the participants felt was required (e.g. thinking, deciding, calculating, remembering, looking, searching…).
2. **Performance:** How successfully participants felt they were able to complete the task.
3. **Effort:** How hard participants felt it was to accomplish their level of performance.
4. **Frustration:** How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, replaced and complacent the participants felt during the task.

[69] The simplified version of the NASA-TLX used for this project has removed the 'weighting process' that is part of the original NASA-TLX. For more information on the weighting process, see the original NASA-TLX guide (Hart, 1986).
[70] These categories and their descriptions are based on those found in the NASA-TLX (Hart, 1986).

This short questionnaire was given to each participant twice; once after completing the tasks using a standard Chrome web browser (test A), and once after completing the tasks using the WhereFinder extension (test B).

Afterwards, the results of each participant's first test and second test were compared with one-another. When comparing the results, the essence of the comparison is thus; if test A results exceed test B results, then it suggests that the traditional browsing techniques of a web browser are superior to the browsing techniques available with the WhereFinder extension. Consequently, this would indicate that the extension built for this project largely doesn't succeed in its project aim of developing: "a browser extension that improves the usability of web browsers by creating a method for users to retrieve supplementary information about geographical locations on web pages more efficiently." Conversely, if test B results exceed test A results, then the inverse to the above notion may be true.

For evidence of the statistical significance of the test results, a between-groups analysis of variance (ANOVA) has been included for each comparison of test A and test B results. For this, the P-value of each comparison has been calculated. This tells us the likelihood that the observation could've been observed by random error; a p-value closer to 1 suggests that the difference between the groups is due to chance, whereas p-values closer to 0 indicate that the difference is unlikely to be due to chance.

9.2.2 Test Results

The results of the 'RTLX Test' (section 13.4) that was conducted provided this project with the following information (fig. 62, fig. 61):

Geolocation Information Retrieval Test: Without WhereFinder (Test A)				
Test Subject Num.	**Mental Demand**	**Perf.**	**Effort**	**Frustration**
1	2	10	2	1
2	3	10	2	1
3	2	8	1	1
4	3	10	2	2
5	2	9	2	1
6	2	10	2	2
7	1	10	1	2
8	3	9	2	2
9	1	10	1	1
10	2	10	1	1

Figure 61: RTLX test A results.

Geolocation Information Retrieval Test: With WhereFinder (Test B)				
Test Subject Num.	**Mental Demand**	**Perf.**	**Effort**	**Frustration**
1	2	8	4	4
2	2	7	4	4
3	2	10	3	1
4	2	6	5	4
5	1	9	4	2
6	3	7	4	3
7	3	10	2	1
8	1	9	2	1
9	2	10	2	3
10	2	7	3	4

Figure 62: RTLX test B results.

9.2.2.1 Mental Demand

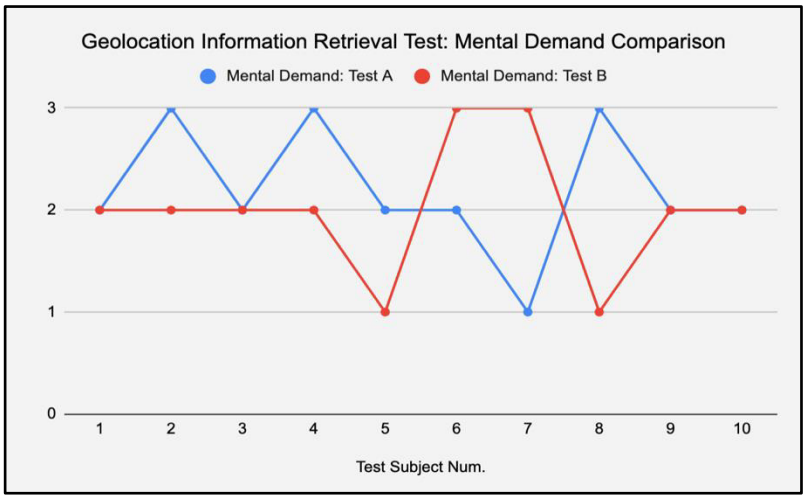


Figure 63: Graph 1, comparing test A & B 'Mental Demand' results

Participants rated the mental demand of test A and test B very similarly (fig. 63). However, on closer inspection, the mean and mode scores on test B provided slightly better results than those of test A (fig. 64). This slightly lower score may indicate that users found it slightly less mentally demanding to retrieve the information with the aid of the WhereFinder extension than without its aid. However, the P-value between these groups is 0.5, which suggests this comparison may not hold statistical significance.

	Test A	Test B
Mean	2.1	2
Mode	2(5)	2(6)
Median	2	2
Range	2	2
Var.	0.44	0.4
P-value	0.5	

Figure 64: Graph 1 statistics

9.2.2.2 Performance

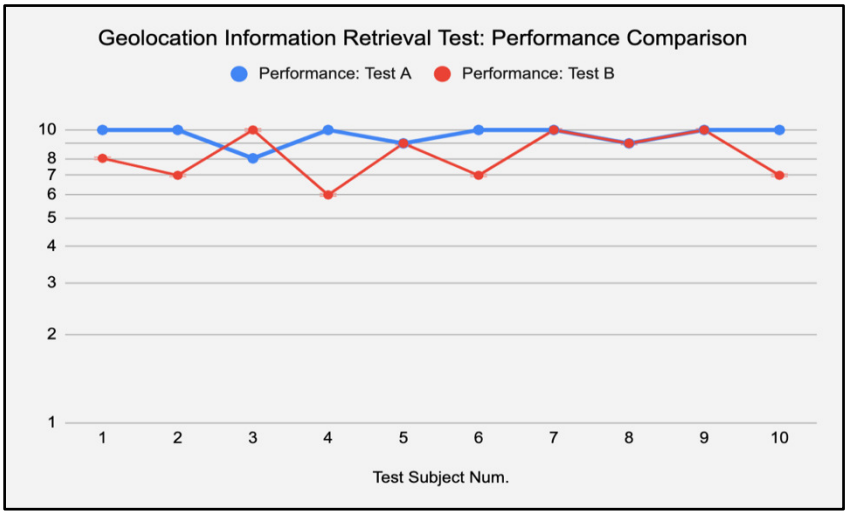


Figure 66: Graph 2, comparing test A & B 'Performance' results.

According to the mode scores of both tests (fig. 65), participants reported performing higher much more frequently in test A than in test B. In fact, on test B there were some significant drops in performance in a number of participants, with some test B scores dropping below 7. This means the variance of test B was more unpredictable, and its mean score was lower too. These results suggest that participants weren't able to perform as well using the WhereFinder extension, and that the experience of using the WhereFinder extension was quite inconsistent. Also, the P-value between these groups of 0.02 suggests this comparison has statistical significance.

	Test A	Test B
Mean	9.6	8.3
Mode	10(7)	10(3),7(3)
Median	10	8.5
Range	2	4
Var.	0.44	2.01
P-value	0.02	

Figure 65: Graph 2 statistics

9.2.2.3 Effort

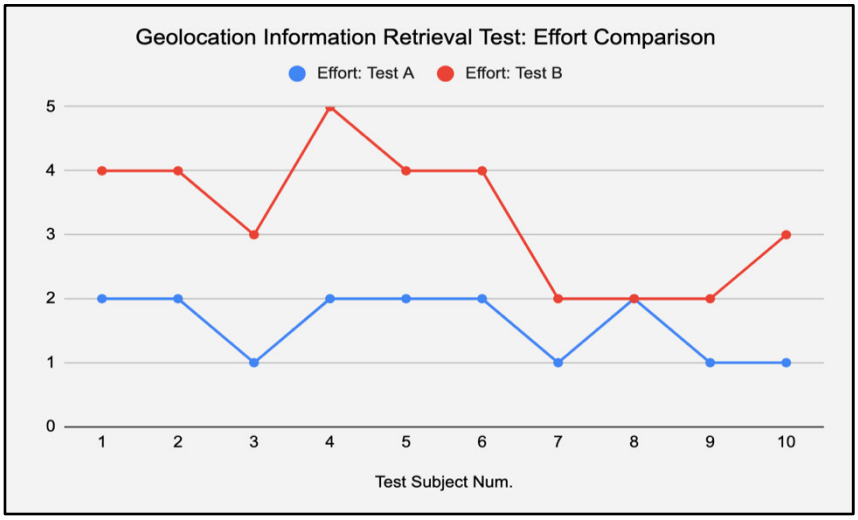


Figure 68: Graph 3, comparing test A & B 'Effort' results.

This category saw an obvious discrepancy between test A & test B results (fig. 67), with mode, mean, and median scores that were all noticeably higher on test B than on test A (fig. 68). This suggests that participants saw using the WhereFinder extension as more taxing than traditional web browsing methods. Additionally, the mode score of test A was more consistent, which implies that the experience of using the standard Chrome browser to complete the tasks was more reliable than the experience of completing the task with the WhereFinder extension. Also, the P-value between these groups of 0.0002 suggests this comparison has statistical significance.

	Test A	Test B
Mean	1.6	3.3
Mode	2(6)	4(4)
Median	2	3.5
Range	1	3
Var.	0.24	1.01
P-value	0.0002	

Figure 67: Graph 3 statistics.

9.2.2.4 Frustration

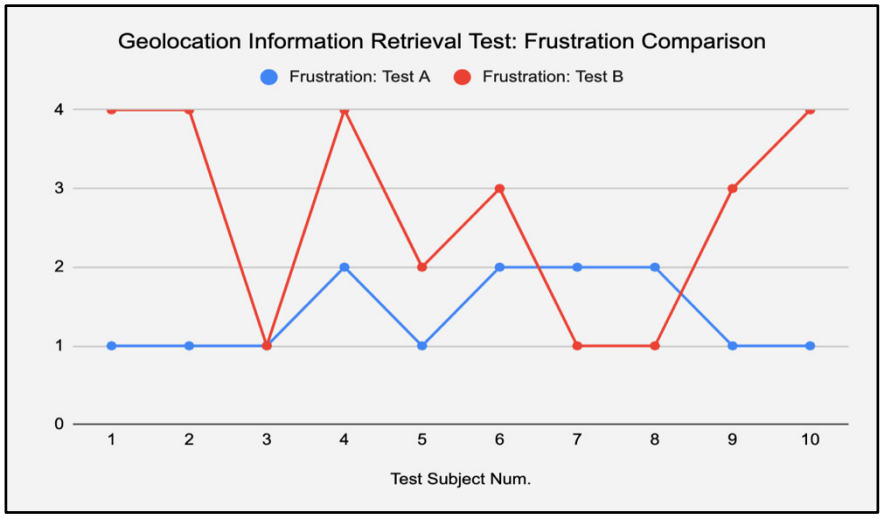


Figure 70: Graph 4, comparing test A & B 'Frustration' results

Test B's higher mode and mean scores further consolidate that test A was the preferred experience of the two tests (fig. 69). However, test B once again had a high variance, and also a high range, which again suggests an inconsistent experience across the participants taking the test (fig, 69). Interestingly, there were multiple participants reporting test B frustration levels as low as test A, which suggests that some of the tests went markedly differently than others. Also, the P-value between these groups of 0.01 suggests this comparison has statistical significance

	Test A	Test B
Mean	1.4	2.7
Mode	1(6)	4(4)
Median	1	3
Range	1	3
Var.	0.24	1.61
P-value	0.01	

Figure 69: Graph 4 statistics.

9.2.2.5 Aggregate Comparison

The below graph compares the cumulative scores from all 10 participants across three of the test's categories (categories in which lower scores indicate a more agreeable experience (Mental Demand, Effort, Frustration)) (fig. 71).

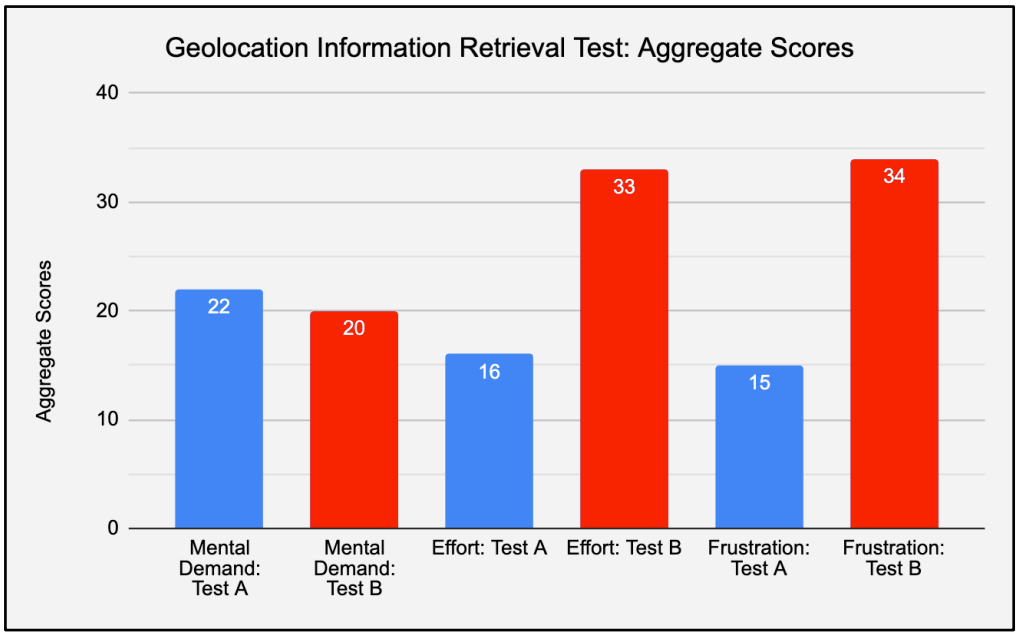


Figure 71: Graph 5, comparing test A & B aggregate results.

This comparison makes it clear that participants generally found using the WhereFinder extension to be a less agreeable experience than using traditional web browsing methods for retrieving information.

9.2.3 Test Conclusions

From these test results it's evident that in its current form, this software doesn't currently fulfil its original project aim of improving the usability of the web browser. This is evident because throughout the RTLX tests using the WhereFinder extension was perceived as requiring a higher workload than using traditional web browser methods. Regardless, this UT was denoted to only be formative testing, because at its current stage the software requires more development before it's ready for a commercial release. As a product under development, the sub-par performance of the WhereFinder extension in this RTLX test was to be expected.

What was encouraging about these results is the slight better results of test B over test A in the category of 'Mental Demand'. This suggests that the tools that WhereFinder provides for retrieving supplementary content may be less mentally demanding than traditional methods, likely because retrieving supplementary content using the WhereFinder extension requires less steps than traditional methods, as explained in the 'Project Introduction' (section 2). However, the statistical significance of this result is questionable, due to its ambiguous P-value of 0.5. As a result, more extensive testing would be needed to confirm the validity of this result.

On the other hand, the inconsistent test B results across the other 3 categories (Performance, Frustration, and Effort)) compared to the lower and less variable test A results suggest that test A provided a more agreeable experience overall. Additionally, the low p-values of all these categories suggest that the comparison of these results has statistical significance. However, these results were anticipated for a number of reasons. First, a more consistent experience on a standard web browser is likely due to the participants familiarity with the standard web browsing methods; no learning was required for test A, whereas participants had to learn how to use WhereFinder in test B. This is an issue because higher learnability can affect the amount of work 'using' software requires. Secondly, in its current form the WhereFinder extension has some latency in its loading time which limits the efficiency of using it. Were it able to load faster, the experience of using it would be improved. Third, in its current form, the WhereFinder extension has some bugs which cause defects in its performance on certain websites. This issue can lead to some inconsistent experiences in using the software, as shown from the high variance within test B's results.

Together, all of these issues combine to create a less consistent experience and agreeable experience when using WhereFinder than when using traditional web browsing methods. Nevertheless, all these issues could be improved in future iterations of this software, and in correcting these shortcomings, it's very likely that the results of a second round of RTLX testing could be provide quite different results.

10 Project Conclusion

The aim of the project was to:

> *Develop a browser extension that improves the usability of web browsers by creating a method for users to retrieve supplementary content about geographical locations on web pages more efficiently.*

What has been successful about this project is that the browser extension built for this project does enable users to retrieve supplementary content about geographical locations on web pages without requiring them to perform additional web page navigation. The extension will handle a wide range of geographical locations, and from the input text will display relevant supplementary information. Furthermore, WhereFinder also disambiguates locations that have the same name particularly well. This means that users can retrieve supplementary content from the specific location that they're interested in, rather than any other location with the same name[71] (fig. 72). The functionality presented in this extension offers a potential solution to excessive tab use in browsers because the WhereFinder extension pulls content into the user's current tab that they would otherwise have to navigate to themselves (fig. 73). The content that is retrieved includes; a map display, the location's wikipedia page, local news articles, and images related to the location (fig. 74).

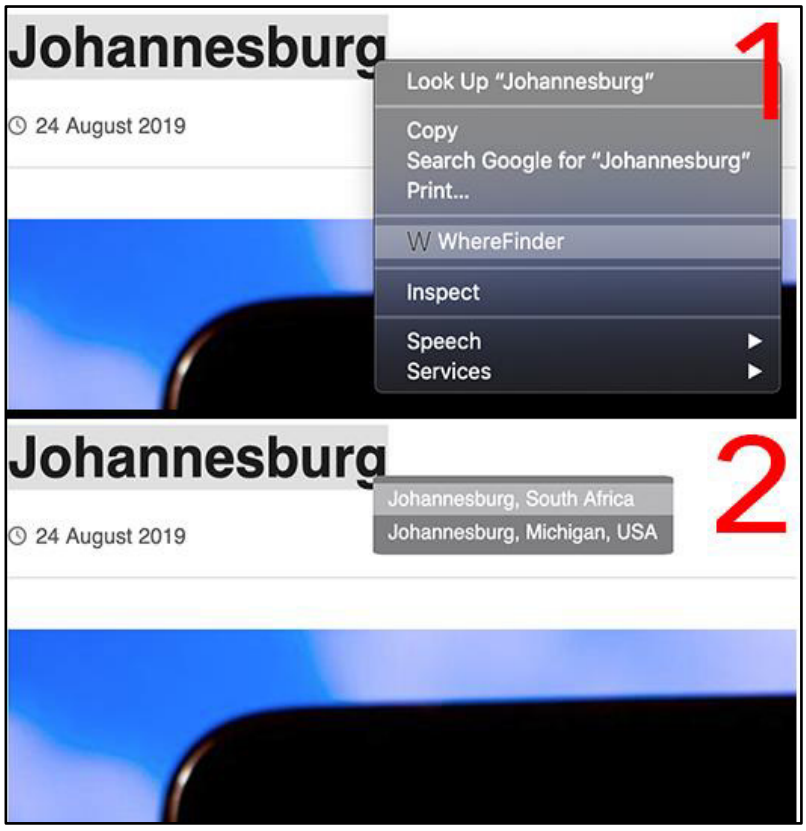


Figure 72: WhereFinder illustrations of use. 1: Highlighting text and clicking the 'WhereFinder' button in the context menu. 2: Clicking the specific location to retrieve information in the second context menu.

[71] For details on how the code that enables this functionality was written, see the sections, 'Process Query 1' (section 7.1.2.4), and 'Make Context Menu' (section 7.1.2.6).

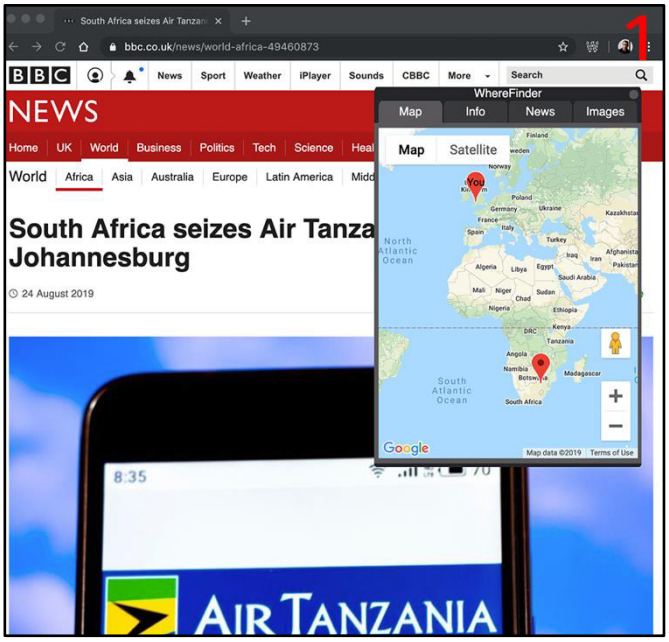


Figure 74: WhereFinder display within the Chrome web browser.

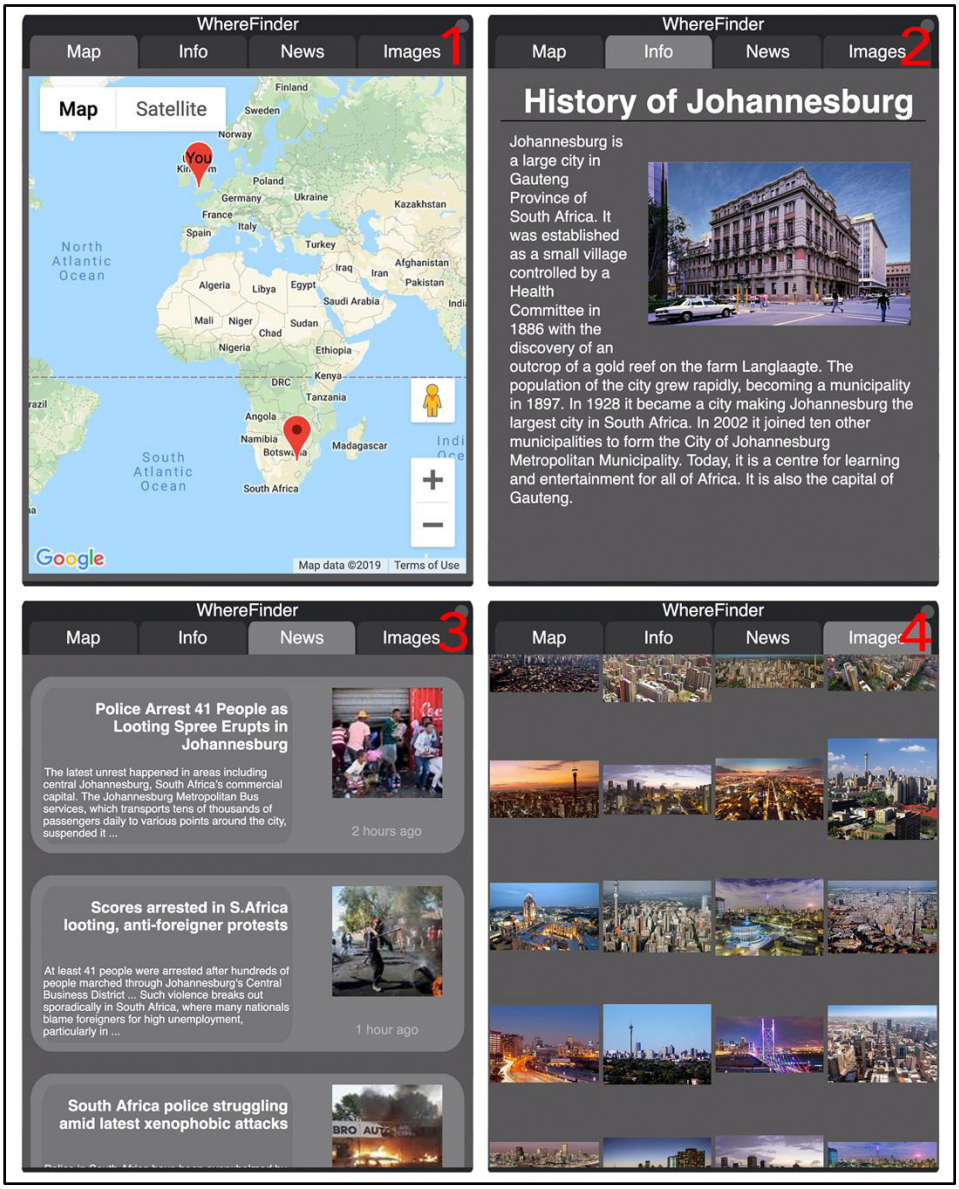


Figure 73: Displays of the 4 tabs within the WhereFinder extension. 1: Google Maps. 2: Wikipedia. 3: Bing News. 4: Bing Images.

Despite these positives, ultimately this project does not fulfil its aim. This is because the aim of this project was to 'improve the usability of web browsers', and despite WhereFinder's functionality, using the extension in its current iteration doesn't allow users to retrieve content more efficiently than traditional web browsing methods. This is because of the latency present when loading the WhereFinder main window[72], which adds enough time onto using the extension to warrant its use to be inefficient when compared to traditional browsing methods. Also, WhereFinder still has some defects that effect its performance on some web pages. The effect of these issues were shown in the results from the 'Usability Testing' (section 9.2) that was performed for this project. In this test, users generally had a more consistent experience using traditional web browser methods than using WhereFinder.

However, as the amount of content on the internet continues to grow, solutions to issues like excessive tab use could still benefit from new solutions, and as such, the notion has not changed – new solutions to navigating web content are needed could very well still be advantageous to users. The hurdles that remain in this project meeting its aim are merely technical challenges which are entirely surmountable. Moreover, these issues exist solely due to the short development schedule for this project, which was limited to eight weeks. The 'Revisions Needed Before Release' (section 10.2.1) of this extension can be found in a section below, in which the work that needs to be completed on the next iteration of this software is delineated. Additionally, this conclusion also contains a discussion on project 'Reflections' (section 10.1), as well as recommendations for 'Future Research' (section 10.2.2).

10.1 Reflections

This project represents an effective learning experience. The following section will reflect on the challenges and issues faced in this project. This includes discussions on; 'Requirements Reflections' (section 10.1.1),

[72] Details of this latency are explained in the sections 'NLP Reflections' (section 10.1.3.1), and 'Wikipedia Reflections' (section 10.1.3.2).

'Approach Reflections' (section 10.1.2), 'Software Development Reflections' (section 10.1.3), and 'Software Testing Reflections' (section 10.1.4).

10.1.1 Requirements Reflections

The 'Project Requirements' (section 4.1) helped inform other considerations made throughout this project. The tasks of the 'Project Timeline' (section 7), for instance, were created in reference to the requirements. In fact, each functional requirement was completed in the timeline in the exact order they were written in the requirements. Although this wasn't necessarily the optimal approach towards building the software, this nonetheless demonstrates the influence that the requirements had on creating the timeline for this project.

On reflection on the non-functional requirements (NFRs), however, there are some issues to be found, including:

- **Inaccurate requirements:** A few of the NFRs specify building a Python server, when ultimately Chrome's native messaging API was used to communicate with a local Python script.
- **Ineffective requirement**s: Some of the NFRs specify functionality that added unnecessary complexity to the project and thus could've eliminated[73].

These issues stem from the fact that the requirements were written at the very beginning of this project as part of the project's initial proposal. This was before much of the research, and learning, required to build this software had been undertaken. This meant that the NFRs were written without all of the necessary knowledge needed to write an optimal set of requirements.

If this project continues, a new set of NFRs would need to be written that are more appropriate for the functionality that the next iteration of this software should have.

[73] For example, on reflection, the implementation of NLP within this project added unnecessary complexity. This realization is discussed in the 'Software Development Reflections' section, 'NLP Reflections'.

10.1.2 Approach Reflections

The following section reflects on the 'Project Approach' (section 6).

10.1.2.1 Development Strategies Reflections

In the 'Development Strategies Considerations' (section 6.1), the rationale behind choosing the Waterfall method for this project's development was discussed. But on reflection, there are a number of reasons why this decision may have inadvertently harmed the project.

Firstly, the Waterfall method's inflexibility in "incorporating change" (Ivins, 2019b) to previously written code lead to some significant issues in this project. These issues arose because of the fact that occasionally during development unforeseen obstacles would arise that would add additional time and complexity to the software's development. This likely could have been avoided if an Agile approach to development had been taken, because an understanding of these issues would've been realised earlier in development, and thus the software could've been built with more awareness of what approaches to earlier development would work optimally with the software that was developed later in the project.

Secondly, another way that the Waterfall method harmed this project was the number of additional bugs that were created as a result of a linear development schedule. For example, when developing the communication between the extension and the text processor, many new bugs arose with the text processor, such as variables that were being improperly referenced at different points throughout the scripts. Finding all of these issues and correcting them added additional time to development that could've been avoided if the communication between this project's different scripts had been developed earlier in the development schedule, as would've been the case with an Agile approach.

10.1.2.2 Risks Reflections

Within the 'Business Considerations' (section 6.2), the 'Risks' (section 6.1.1) that were outlined provided useful warnings and advice for this project. However, in reality this information wasn't heeded as often as it could have been during the development of this project.

For instance, in one risk it was suggested that only APIs that are well-known or have good documentation should be used in this project because they're arguably less likely to cause issues. For certain phases of this software's development, this advice was ignored, and as a direct result of this, there is unnecessary latency in the loading of the WhereFinder extension that could've been avoided had the advice from this risk been heeded. This particular issue was caused by the library used to perform HTTP requests to Wikipedia (2019, PyPI)[74]. This issue within the software will be fixed in the next iteration of this extension, and the advice given in this risk will be heeded more carefully in the future.

Another risk that was partially ignored was the suggestion of keeping a remote repository continually up-to-date. A GitHub repository was made for this project (GitHub, 2019b), but it was not updated frequently enough to deem its use sufficient. A simple series of commands is all it takes to keep a remote repository up-to-date, and in the future, more effort should be made to properly maintain a backup of this and other projects.

10.1.3 Software Development Reflections

This section reflects on some of the software development decisions made throughout this report, including discussions on 'NLP Reflections' (section 10.1.3.1), and 'Wikipedia Reflections' (section 10.1.3.2).

[74]Details of this issue are explained in the section, 'Wikipedia Reflections' (section 10.1.3.2).

10.1.3.1 NLP Reflections

This project's use of NLP to extract and label geographical text from input strings may not have provided as much benefit to the extension as it did add complication. There are a couple of reasons for this:

1. Users tend to only highlight text which they intend to process. This was an observation noticed in the habits of the participants who took the usability test. Based on this observation, it would seem that users don't need NLP to extract the names of places from small segments of text for them. This makes the way this extension currently leverages NLP somewhat superfluous to actually accomplishing its aim.
2. Performing NLP on small amounts of text is difficult. Twitter messages, for instance, are limited 280 characters and processing them effectively is a "well-known hurdle for many tasks related to" NLP (GitHub Notes, 2015). The text processed by this extension is typically even smaller than Twitter messages. As a result of this issue, the NLP portion of this software occasionally mislabels the input text, which then causes the extension to fail to perform as it was intended to. Had there been more time, this problem could've been overcome by training SpaCy's statistical language model to label NEs as locations more frequently. SpaCy does include tools for training language models (as was briefly described in the 'Project Context' (section 3) discussion on 'NLP' (section 3.3)), but an NER that correctly labels NEs with 100% accuracy isn't currently possible, so this solution still wouldn't entirely solved this issue.
3. Loading a big language model is a time-consuming task. This latency was noticed using the Python 'logging' (fig. 75) module, which logged events as they occurred in the scripts. As can be seen, loading the large model requires over 10 seconds of time to load.

```
DEBUG 2019-09-02 13:02:04,478 :: beginning to load language model "en_core_web_lg"
DEBUG 2019-09-02 13:02:15,847 :: model loaded
DEBUG 2019-09-02 13:06:15,115 :: beginning to load language model "en_core_web_sm"
DEBUG 2019-09-02 13:06:15,847 :: model loaded
```

Figure 75: Excerpts of a log file written to when the 'langproFile.py' module is loaded. The highlighted text indicates the time at which the line of code of executed.

This loading happens every time the extension is first initiated (i.e. when a browser is loaded). A solution to this would be to just use smaller model, but this leads to incorrect NE labelling happening at a higher frequency. Additionally, a smaller amount of latency can also be seen when a the NLP is performed on the users input text – anywhere from roughly 1-3 seconds.

On reflection, rather than using NLP, a better solution would have been to simply use the input text to search a DB of locations for any matches; then, the DB search results could be used to determine whether or not the input text corresponds to any actual locations. This change would simplify the software and would eliminate some of the scenarios in which the software isn't performing as quickly as is ideal. This could improve the efficiency of the extension, which would help this project better meet its 'Project Aim' (section 2.1). In a future iteration of this software, this is a likely modification to be engineered.

Notwithstanding, there may be other ways that this extension could leverage NLP instead. One alternative use of NLP could be to rank order the matches returned by the locations DB. This could be done by using NLP to process a page's entire contents to create predictions on which location the web page is specially referring to. Then, the NLP predictions could be used to rank the potential matches returned by the location DB in order of their likelihood. This method of using NLP for 'key phrase extraction' from documents is a method for improving IR that was briefly described in the 'Project Context' (section 3) discussion on 'IR' (section 3.4). In the future, it would be advantageous to add functionality like this to the WhereFinder extension.

10.1.3.2 Wikipedia Reflections

There remain other opportunities to optimize the performance of this software, one of which is an improvement that could be made in how Wikipedia information is retrieved. Currently, the retrieval of such information occurs via a Python library called wikipedia (PyPI, 2019). However, this solution causes latency which becomes noticeable when the user is waiting for the WhereFinder display to load.

Figure 76: WhereFinder extension loading screen.

This latency was observed when using the Python logging module which was used to determine the flow and execution times of events throughout the Python scripts by writing information to a separate log file (fig. 77).

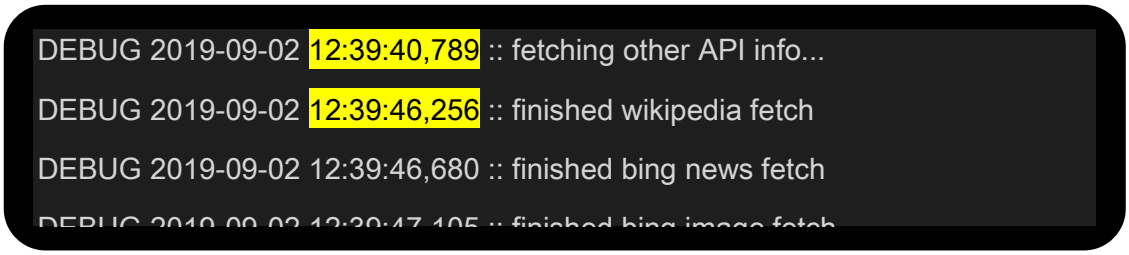


Figure 77: Excerpts from a log file: written when running the WhereFinder extension is processing text.

As seen above, it's the segment of the code that handles fetching content from Wikipedia that creates the largest latency. Based on this, it's believed that the 'wikipedia' library is to blame for the latency. In retrospect, a better solution for retrieving Wikipedia content would have been to work directly with the WikiMedia API (Wikimedia Foundation, 2019). This way, any slow or superfluous code within the wikipedia library could be avoided, and so the latency that's currently present in using the WhereFinder extension could've been avoided. In a future iteration of this software, this improvement would be a priority because it has the potential to make using this extension a quicker experience, which would improve its usability and help this project to better meet its aim.

10.1.4 Software Testing Reflections

Software testing isn't an area that was covered in detail on the Computing & IT Management course, but including it in this project was useful in ensuring that the software that was developed in this project was at least partially successful in fulfilling the 'Project Requirements' (section 4), and the 'Project Aim & Objectives' (section 2.1).

Firstly, throughout the development of this project it's worth noting how useful the Python library, 'logging' was. Discovering this library was instrumental in this project progressing past a bottleneck that

was experienced during the 'Connect Host (Native) Application' (section 7.1.2.3) phase of the development schedule. During this phase, the 'logging' module was used to debug the communications between the different sections of the extension – in specific, the 'background.js' script running on the Chrome browser, and the native application script ('langpro.py') running on the local machine[75].

This communication became an issue because when attempting to initialize a connection between these two scripts, the ambiguous error messages that were seen in the Chrome debugger provided very little detail on the exact nature of the issue was within the code (fig. 78):

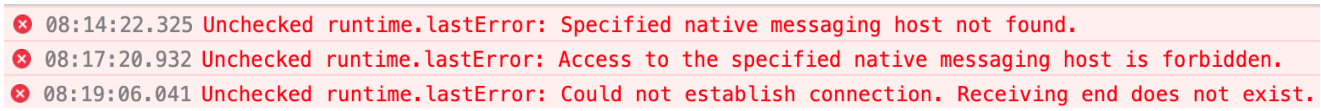


Figure 78: A few examples of native messaging errors seen when attempting to initialize a connect with the native application.

When looking up these errors in Chrome's documentation (Google, 2019dd) the troubleshooting help given in the documentation was found to be very general in nature and didn't provide particularly deep guidance on troubleshooting. In order to debug the issues at this stage of the project, the Python 'logging' module was used to understand the events that were occurring throughout the Python scripts as they were connected to and executed via the Chrome browser. This was achieved by logging information about the script's events within a separate log file. In using this log file, more specific details about the problems that were causing the connection between the two scripts to fail could be understood. This was a challenging section of the project that required a different approach to solving the issue. Using the Python 'logging' module didn't solve the problems faced, but it did provide a path through which the issues could then be better debugged and overcome. In the future, the techniques that were learned here will certainly be useful again.

[75] For more details on native messaging in this extension, see the section 'Connect Host (Native) Application' (section 7.1.2.3).

10.1.4.1 Integration Testing Reflections

Ideally, integration testing would've been conducted on more of the extension, beyond just the 'langproFile.py' module. However, performing white-box tests on browser extensions isn't a straightforward process due to the fact that the browser API's have to be accessible when running the tests, which can be difficult to simulate. There are strategies for simulating 'mock' browser APIs (for example, the JS tool 'Karma' can enable one to simulate executing code within a browser context for testing (Karma, 2019)), but due to the eight week development schedule of this project it wasn't possible to learn & develop sophisticated tests in time using such an approach. If more time was available, then more integration testing of the rest of the software would've been performed.

10.2 Recommendations

The follow sub sections discuss the future work to be done on this particular project ('Revisions Needed Before Release' (section 10.2.1)), and the future work that could be done within this area ('Future Research' (section 10.2.2)).

10.2.1 Revisions Needed Before Release

Before this product could see a commercial release, it's important that this software fulfil the 'Project Aim' (section 2.1). To accomplish this, a few changes would need to be made:

10.2.1.1 Revision #1

The first revision needed is a reduction in the latency present when loading the display of the WhereFinder extension. Currently, the two most time intensive steps in the software are; loading the NLP statistical language model[76], and retrieving information from Wikipedia[77]. Both of these are issues that have been

[76] This issue was discussed in the section, 'NLP Reflections' (section 10.1.3.1).
[77] This issue was discussed in the section, 'Wikipedia Reflections' (section 10.1.3.2).

highlighted during the 'Reflections' (section 10.1) and are technologies whose functionality should be modified in future iterations of this software.

10.2.1.2 Revision #2

The second revision needed is an improvement in the responsiveness of the main display across different web pages. Currently, the main display can sometimes perform inconsistently on certain web pages. This is because when the WhereFinder content is injected into a web page, the native HTML, CSS & JS on that page can interact with the WhereFinder content that's been injected. This can cause the injected content to perform differently than intended. More research is needed into finding a solution to this problem and more testing across many different websites is required in order to ensure that the injected WhereFinder content performs consistently.

One viable solution could be to only inject an Iframe into a web page rather than inject the majority of WhereFinder into the web page. Then, all of WhereFinder's content could instead be injected into a separate web page, which the Iframe that was injected into the parent web page would provide a view into. This would limit the amount of potential conflicts that a web page's native HTML, CSS & JS could have with WhereFinder because WhereFinder's content would instead exist on a separate web page, only accessed via a view provided by the Iframe element on the parent web page. A solution such as this would be ideal for the next iteration of this software.

10.2.1.3 Revision #3

A remote solution for the locations DB would need to be created as currently the locations DB is run locally. Two solutions for this are:

1. Using a cloud service provider, such as Amazon Web Services (AWS). AWS offer a Relational Database System (RDS) which could store the content of the locations DB (Amazon, 2019). Then, requests to this remote DB could be made as needed.

2. Purchasing access to a locations DB from a business. There are numerous businesses that offer such services[78], and in leveraging these DBs, the number of potential locations which the WhereFinder extension could match would see a significant increase.

Ultimately, both of these solutions would require financial spending, so in order to release this product the 'Project Constraints' (section 5) would have to be broken.

10.2.1.4 Revision #4

Expand the testing of this software. This includes adding unit testing, expanding integration testing to test all the JS scripts in the extension, and developing more test cases too. Also, another round of UT for the product should be conducted. After a second iteration of this extension is developed, performing UT again with a larger number of participants could provide results that have more statistical significance and therefore offer better proof that this extension is truly effective in meeting the 'Project Aim' (section 2.1).

10.2.2 Future Research

The largest contribution to the field that this project has made is identifying that there is an issue with how users currently navigate the internet and providing a potential solution to this problem. The number of users who use tabs excessively is a testimony to notion that the issue exists. On the internet, there are simply too many sources of content which users have to consume separately – navigating between different websites to consume their content individually.

It's likely that in the future, any systems that can reduce the number of divergent content sources that users have to retrieve information from manually (that is, navigating themselves to the source of the content) will become increasingly beneficial for users. These systems will likely automate the retrieval of

[78] Some businesses that offer such a service include SimpleMaps (SimpleMaps, 2019), and GeoDataSource (GeoDataSource, 2019).

one or multiple content sources and will allow users to leverage the content from different web pages together in interesting and dynamic ways.

More research should be conducted that investigates any such systems, because due to how ubiquitous the internet is, advancements in any such technologies have the potential to benefit a great number of people. Eventually, it would be ideal for browsers to build technologies that offer such solutions into browsers natively. Based on the perpetual development of web browsers, it's entirely possible that they will.

11 Acknowledgements

I would like to thank Dr Padraig Corcoran for the knowledge and guidance he provided me with in this project. The meetings we held were consistently helpful and enjoyable. Additionally, I'd like to thank my other professors over this last academic year whose modules have all informed and influenced different aspects of this project; Wendy Ivins, Martin Chorley, Matthias Treder, Irena Spasic; and Natasha Edwards – thank you all.

Lastly, I would also like to thank my parents, and my partner, who have all been a continued source of motivation throughout my academic endeavours.

I confirm that all the work in this project is my own, and that this project adheres with Cardiff University's academic regulations (Cardiff, 2019a). The word-count for this dissertation is 17,800.

12 Bibliography

Agile Alliance. 2001. *Principals behind the Agile Manifest*. Available at: http://agilemanifesto.org/principles.html [Accessed: 12th August 2019].

Alonso-Virgós, Lucía. et al. 2019. Analyzing compliance and application of usability guidelines and recommendations by web developers. *Computer Standards & Interfaces* 64. pp.117-132. doi: 10.1016/j.csi.2019.01.004.

Amazon. 2019. *Amazon Relational Database Service (RDS)*. Available at: https://aws.amazon.com/rds/ [Accessed: 1st September 2019].

Apple. 2019. *Building a Safari App Extension*. Available at: https://developer.apple.com/documentation/safariservices/safari_app_extensions/building_a_safari_app_extension [Accessed: 2nd August].

Azad, Hiteshwar Kumar. Deepak, Akshay. A new approach for query expansion using Wikipedia and WordNet. *Information Sciences* 492, pp.147-163. doi: 10.1016/j.ins.2019.04.019.

Barnum, Carol M. 2011. Establishing the Essentials. *Usability Testing Essentials*. pp. 9-26. doi: 10.1016/B978-0-12-375092-1.00001-5.

Black, Rex. 2003. *Critical Testing Processes: Plan, Prepare, Perform, Perfect*. (s.l): Addison-Wesley Professional.

Bradley, Gerald. 2006. *Benefit Realisation Management*. Hampshire, England: Gower.

BSi. 1997. *ISO 9241-1:1997 Ergonomic requirements for office work with visual display terminals (VDTs) — Part 1: General introduction*. Geneva: ISO.

BSi. 2008. *ISO 9241-151:2008 Ergonomics of human-system interaction — Part 151: Guidance on World Wide Web user interfaces*. Geneva: ISO.

BSi. 2018. *ISO 3100:2018: Risk management – Guidelines*. Geneva: ISO

Büttcher, Stefan. et. al. 2010. *Information Retrieval: Implementing and Evaluating Search* Engines. Massachusetts: MIT Press.

Cardiff University. 2019a. *Academic Regulations 2019/2020*. Available at: https://www.cardiff.ac.uk/__data/assets/pdf_file/0009/432666/Academic-Regulations-Handbook-2019-20-English.pdf [Accessed: 1st August 2019].

Christensen, Henrik. 2010. *Flexible, Reliable Software Using Patterns and Agile Development*. New York: CRC Press.

Cloud, Nicholas. Ambler, Tim. 2019. *JavaScript Frameworks for Modern Web Dev.* New York: Springer.

Dahab, Mohamed Yehia. et. al. 2017. A Tutorial on Information Retrieval Using Query Expansion. *Intelligent Natural Language Processing: Trends and Applications*, pp. 761-776. doi: 10.1007/978-3-319-67056-0.

Dale, Robert. 2019. NLP commercialisation in the last 25 years. *Natural Language Engineering* 25 (3), pp. 419-426. doi: 10.1017/S1351324919000135.

GeoDataSource. 2019. *GeoDataSource*. Available at: https://www.geodatasource.com/ [Accessed: 1st September 2019].

GitHub Notes. 2015. *The Twitter API and Python - Syntactic Sugar for Tweepy*. Available at: http://cmry.github.io/notes/twitter-python [Accessed: 29th August 2019].

GitHub. 2019a. *en_vectors_web_lg-2.1.0*. Available at: https://github.com/explosion/spacy-models/releases//tag/en_vectors_web_lg-2.1.0 [Accessed: 3rd August 2019].

GitHub. 2019b. *Dissertation-Project.* Available at: https://github.com/DrakeSterlingDeaton/Dissertation-Project [Accessed: 28th August 2019].

GLUE. 2019. *Leaderboard.* Available at: https://gluebenchmark.com/leaderboard/ [Accessed: 3rd August 2019].

Google. 2009. Extensions Status: On the Runway, Getting Ready for Take-Off. *Chromium Blog.* 9th September. Available at: https://blog.chromium.org/2009/09/extensions-status-on-runway-getting.html [Accessed: 2nd August].

Google. 2012. *Zeitgeist 2012*. Available at: https://archive.google.com/zeitgeist/2012/#the-world [Accessed: 4th August 2019].

Google. 2018. Trustworthy Chrome Extensions, by default. *Chromium Blog.* 1st October. Available at: https://blog.chromium.org/2018/10/trustworthy-chrome-extensions-by-default.html [Accessed: 1st August 2019].

Google. 2019a. *AdBlock.* Available at: https://chrome.google.com/webstore/detail/adblock/gighmmpiobklfepjocnamgkkbiglidom [Accessed: 1st August 2019].

Google. 2019aa. *Places API Usage and Billing.* Available at: https://developers.google.com/places/web-service/usage-and-billing [Accessed: 20th August 2019].

Google. 2019b. *Adobe Acrobat*. Available at: https://chrome.google.com/webstore/detail/adobe-acrobat/efaidnbmnnnibpcajpcglclefindmkaj?hl=en [Accessed: 1st August 2019].

Google. 2019bb. *Content Security Policy (CSP)*. Available at: https://developer.chrome.com/extensions/contentSecurityPolicy [Accessed: 29th August 2019].

Google. 2019c. *Avast SafePrice*. Available at:https://chrome.google.com/webstore/detail/avast-safeprice-compariso/eofcbnmajmjmplflapaojjnihcjkigck [Accessed: 1st August 2019].

Google. 2019cc. Dev Channel Update for Desktop. *Chrome Releases.* 30th August 2019. Available at: https://chromereleases.googleblog.com/ [Accessed: 1st September 2019].

Google. 2019d. *Google Maps select and search.* Available at: https://chrome.google.com/webstore/detail/google-maps-select-and-se/iobjmgojenedagiebkecldbpgimlchje [Accessed: 1st August 2019].

Google. 2019dd. *Debugging Native Messaging.* Available at: https://developer.chrome.com/apps/nativeMessaging#native-messaging-debugging [Accessed: 4th September 2019].

Google. 2019e. *Open in Google Maps.* Available at: https://chrome.google.com/webstore/detail/open-in-google-maps/cinncpjcejdnoaffllljjmgibkkgkigag?hl=en [Accessed: 1st August 2019].

Google. 2019f. *Send to Google Maps*. Available at: https://chrome.google.com/webstore/detail/send-to-google-maps/bhggankplfegmjjngfmhfajedmiikolo/related [Accessed: 1st August 2019].

Google. 2019g. *Quick Maps.* Available at: https://chrome.google.com/webstore/detail/quick-maps/bgbojmobaekecckmomemopckmeipecij?hl=en [Accessed: 1st August 2019].

Google. 2019h. *Chrome APIs.* Available at: https://developer.chrome.com/extensions/api_index [Accessed: 2nd August 2019].

Google. 2019i. *Content Scripts.* Available at: https://developer.chrome.com/extensions/content_scripts [Accessed: 2nd August].

Google. 2019j. *Manifest File Format.* Available at: https://developers.chrome.com/extensions/manifest [Accessed: 2nd August 2019].

Google. 2019k. *chrome.extension*. Available at: https://developer.chrome.com/extensions/extension [Accessed: 3rd August].

Google. 2019l. *chome.runtime*. Available at: https://developer.chrome.com/apps/runtime [Accessed: 3rd August 2019].

Google. 2019l. *chrome tabs*. Available at: https://developer.chrome.com/extensions/tabs [Accessed: 3rd August].

Google. 2019m. *Google Translate.* Available at: https://chrome.google.com/webstore/detail/google-translate/aapbdbdomjkkjkaonfhkkikfgjllcleb?hl=en [Accessed: 3rd August 2019].

Google. 2019n. *Read Aloud: A Text to Speech Voice Reader.* Available at: https://chrome.google.com/webstore/detail/read-aloud-a-text-to-spee/hdhinadidafjejdhmfkjgnolgimiaplp [Accessed: 3rd August 2019].

Google. 2019o. *Google Maps Platform Documentation*. Available at: https://developers.google.com/maps/documentation/ [Accessed: 5th August 2019].

Google. 2019p. *TooManyTabs for Chrome*. Available at: https://chrome.google.com/webstore/detail/toomanytabs-for-chrome/amigcgbheognjmfkaieeeadojiibgbdp [Accessed: 7th August 2019].

Google. 2019r. *TDFB*. Available at: https://chrome.google.com/webstore/detail/tdfb/pffbbpegcmjjkgkefanlcbklhgkmlfoj?hl=en [Accessed: 7th August 2019].

Google. 2019s. *Overview.* Available at: https://developers.google.com/maps/documentation/javascript/tutorial [Accessed 7th August 2019].

Google. 2019t. *Place IDs.* Available at: https://developers.google.com/places/place-id [Accessed: 7th August 2019].

Google. 2019u. *Libraries.* Available at: https://developers.google.com/maps/documentation/javascript/libraries [Accessed: 8th August 2019].

Google. 2019v. *chrome.contextMenus.* Available at: https://developer.chrome.com/apps/contextMenus [Accessed: 10th August 2019].

Google. 2019w. *chrome.tabs*. Available at: https://developer.chrome.com/extensions/tabs [Accessed: 10th August 2019].

Google. 2019x. *chrome.runtime*. Available at: https://developer.chrome.com/extensions/runtime [Accessed: 11th August 2019].

Google. 2019y. *Google Maps JavaScript API V3 Reference*. Available at: https://developers.google.com/maps/documentation/javascript/reference/ [Accessed: 11th August 2019].

Google. 2019z. *worlddb.* Available at: https://code.google.com/archive/p/worlddb/downloads#! [Accessed: 15th August 2019].

Google. 2109q. *Enhancer for YouTube*. Available at: https://chrome.google.com/webstore/detail/enhancer-for-youtube/ponfpcnoihfmfllpaingbgckeeldkhle [Accessed: 7th August 2019].

Grammarly. 2018. How We Use AI to Enhance Your Writing | Grammarly Spotlight. *Grammarly Blog.* August 14th. Available at: https://www.grammarly.com/blog/how-grammarly-uses-ai/ [Accessed: 3rd August 2019].

Gupta, Swati. Patel, D. 2019. \[\hbox {NE}^2\]: named event extraction engine. *Knowledge and Information Systems* 59 (2), pp. 311-335. doi: 10.1007/s10115-018-1208-8.

Hart, Sandra G. 1986. *NASA Task Load Index (TLX) Volume 1.0; Paper and Pencil Package*. CA, US: NASA Ames Research Centre.

Hasibi, Faegheh. et. al. 2017. Entity Linking in Queries: Efficiency vs Effectiveness. In: Joemon M., J. ed. *European Conference on Information Retrieval,* Aberdeen, UK, April 8-13, 2017. (s.l): Springer, pp. 40-53. doi: 10.1007/978-3-319-56608-5_4.

Helms, James W. et. al. 2006. A field of the Wheel – a usability engineering process model. *Journal of Systems and Software* 76 (6). pp. 841-858. doi: 0.1016/j.jss.2005.08.023.

Hendy, K. et. al. 1993. Measuring subjective workload: When is one scale better than many? *Human Factors* 35 (4), pp. 579-601. doi: 10.1177/001872089303500401.

Hermawati, S. Lawson, G. A user-centric methodology to establish usability heuristics for specific domains. In: *Proceedings of the International Conference on Ergonomics & Human Factors*, Northamptonshire, UK, 13-16 April 2015. pp. 80-85.

Hubert, Hofmann. Franz, Lenhner. 2001. *IEEE Software* 18 (4), pp. 58-66. doi: 10.1109/MS.2001.936219.

IEEE. 1990. *IEEE Standard Glossary of Software Engineering Terminology.* New York: The Institute of Electrical and Electronics Engineers Inc.

Inostroza, R. et. al. 2015. Developing SMASH: a set of SMArtphone uSability Heuristics. *Computer Standards & Interfaces* 43. pp. 40-52. doi: 10.1016/j.csi.2015.08.007.

Ivins, Wendy. 2019a. Session 11 – Risk Management. *CMT301: Business and IT Management.* Cardiff University. Available at: https://learningcentral.cf.ac.uk/webapps/blackboard/content/listContent.jsp?course_id=_387583_1&content_id=_4936508_1 [Accessed: 9th August 2019].

Ivins, Wendy. 2019b. Session 4 - Software Development and Requirements. *CMT301: Business and IT Management.* Cardiff University. Available at: https://learningcentral.cf.ac.uk/webapps/blackboard/content/listContent.jsp?course_id=_387583_1&content_id=_4849198_1 [Accessed: 11th August 2019].

Jinho D. Choi. et. al. It Depends: Dependency Parser Comparison Using a Web-based Evaluation Tool. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics*, Beijing, China, July 26-31 2015. (s.l): ACL, pp.387-396. doi: 10.3115/v1/P15-1038.

Karma. 2019. *Karma*. Available at: https://karma-runner.github.io/latest/index.html [Accessed: 5th September 2019].

LDC. 2013. *OntoNotes Release 5.0.* Available at: https://catalog.ldc.upenn.edu/LDC2013T19 [Accessed: 3rd August 2019].

Liu, B. 2011. Sentiment Analysis and Opinion Mining. In: *Association for the Advancement of Artificial Intelligence – 2011*, CA, USA: Morgan and Claypool Publishers.

Liu, Y. L. Wickens, C. D. 1994. Mental workload and cognitive task automaticity - An evaluation of subjective and time-estimation metrics. *Ergonomics* 37 (11). pp. 1843- 1854. doi: 10.1080/00140139408964953.

Mayhew D.J. 1992. *Principals and Guidelines in Software User Interface Design*. New Jersey: Prentice Hall.

Mccullough, Brian. 2018. *How the Internet Happened: From Netscape to the iPhone*. New York: Liveright Publishing Cooperation.

Microsoft 1999. *Microsoft Delivers World's Fastest Modern Browser Available Today.* Available at: https://web.archive.org/web/20071213175948/http://www.microsoft.com/presspass/press/1999/mar99/ieavlpr.mspx [Accessed: 2nd August].

Microsoft. 2017. *Internet Explorer Browser Extensions*. Available at: https://docs.microsoft.com/en-us/previous-versions/windows/internet-explorer/ie-developer/platform-apis/aa753587(v=vs.85) [Accessed: 2nd August].

Microsoft. 2019a. *Bing News Search.* Available at: https://azure.microsoft.com/en-us/services/cognitive-services/bing-news-search-api/ [Accessed: 5th August 2019].

Microsoft. 2019b. *Bing Image Search.* Available at: https://azure.microsoft.com/en-gb/services/cognitive-services/bing-image-search-api/ [Accessed: 5th August 2019].

Microsoft. 2019c. *NewsSearchAPI class.* Available at: https://docs.microsoft.com/en-us/python/api/azure-cognitiveservices-search-newssearch/azure.cognitiveservices.search.newssearch.newssearchapi?view=azure-python [Accessed: 11th August 2019].

Microsoft. 2019d. *ImageSearchAPI class.* Available at: https://docs.microsoft.com/en-us/python/api/azure-cognitiveservices-search-imagesearch/azure.cognitiveservices.search.imagesearch.imagesearchapi?view=azure-python [Accessed: 11th August 2019].

Microsoft. 2019e. *NewsOperations class.* Available at: https://docs.microsoft.com/en-us/python/api/azure-cognitiveservices-search-newssearch/azure.cognitiveservices.search.newssearch.operations.newsoperations?view=azure-python [Accessed: 11th August 2019].

Microsoft. 2019f. *ImagesOperations class.* Available at: https://docs.microsoft.com/en-us/python/api/azure-cognitiveservices-search-imagesearch/azure.cognitiveservices.search.imagesearch.operations.imagesoperations?view=azure-python [Accessed: 11th August 2019].

Microsoft. 2019g. *Microsoft Azure.* Available at: https://azure.microsoft.com/en-us/free/ [Accessed: 20th August 2019].

Mozilla. 2017. *Context Graph Data Analysis*. Available at: https://medium.com/firefox-context-graph/context-graph-data-analysis-1be000ef8341 [Accessed: 1st August 2019].

Mozilla. 2019a. *Statistics for Web Developer.* Available at: https://addons.mozilla.org/en-us/firefox/addon/web-developer/statistics/usage/?last=30 [Accessed: 1st August 2019].

Mozilla. 2019b. *Search on Google Maps*. Available at: https://addons.mozilla.org/en-GB/firefox/addon/search-on-gmaps/?src=search [Accessed: 1st August 2019].

Mozilla. 2019c. *What are extensions?* Available at: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/What_are_WebExtensions [Accessed: 2nd August].

Needham, Kev. 2015. The Future of Developing Firefox Add-ons. *Mozilla Add-ons Blog.* 21st August. Available at: https://blog.mozilla.org/addons/2015/08/21/the-future-of-developing-firefox-add-ons [Accessed: 2nd August 2019].

Nielsen, Jakob. 1993a. Chapter 2 – What is Usability? *Usability Engineering*. pp. 23-48. doi: 10.1016/B978-0-08-052029-2.50005-X.

Nielsen, Jakob. 1993b. Chapter 3 – Generations of User Interfaces. *Usability Engineering*. pp. 49-70. doi: 10.1016/B978-0-08-052029-2.50006-1.

Nielsen, Jakob. 1993c. Chapter 4 – The Usability Engineering Lifecycle. *Usability Engineering.* pp. 71-114. doi: 10.1016/B978-0-08-052029-2.50007-3.

Nielsen, Jakob. 1993d. Chapter 5 - Usability Heuristics. *Usability Engineering*. pp. 115-163. doi: 10.1016/B978-0-08-052029-2.50008-5.

Norouzi, Yousef. Hakimpour, Farshad. 2019. A Spatiotemporal Semantic Search Engine For Cultural Events. In: *2019 5th International Conference on Web Research (ICWR)*, Tehran, Iran. April 24-25 2019. doi: 10.1109/ICWR.2019.8765287.

OpenSSL. 2019. *OpenSSL: Cryptography and SSL/TLS Toolkit.* Available at: https://www.openssl.org/ [Accessed: 5th September 2019].

Oxford. 2016. *A Dictionary of Computer Science*. 7th ed. Oxford: Oxford University Press.

PyPI. 2019. *Wikipedia Documentation*. Available at: https://wikipedia.readthedocs.io/en/latest/code.html [Accessed: 11th August 2019].

Quiñones, Daniela. Rusu, Cristian. 2017. How to develop usability heuristics: A systematic literature review. *Computer Standards & Interfaces* 53. pp. 89-122. doi: 10.1016/j.csi.2017.03.009.

Ray, Jeffery, et. al. The Rise of Big Data Science: A Survey of Techniques, Methods and Approaches in the Field of Natural Language Processing and Network Theory. *Big Data and Cognitive Computing* 2 (3). doi: 10.3390/bdcc2030022.

Rico, David F. et. al. 2009. *The Business Value of Agile Software Methods*. (s.l): J. Ross Publishing.

Rossum, G. 1995. *Python Reference Manual*. Amsterdam: CWI.

SimpleMaps. 2019. *World Cities Database*. Available at: https://simplemaps.com/data/world-cities [Accessed: 1st September 2019].

Simske, Steven J. Vans, Marie. 2019. Functional Applications of Text Analytics Systems. In: Archiving Conference, *Archiving 2019 Final Program and Proceedings* (4), May 14-17 2019, pp 114-119. doi: https://doi.org/10.2352/issn.2168-3204.2019.1.0.27.

SpaCy. 2019a. *SpaCy*. Available at: https://spacy.io/ [Accessed: 3rd August 2019].

SpaCy. 2019b. *Training SpaCy's Statistical Models.* Available at: https://spacy.io/usage/training [Accessed: 3rd August 2019].

SpaCy. 2019c. *Language Models: English.* Available at: https://spacy.io/models/en#en_vectors_web_lg [Accessed: 3rd August 2019].

SpaCy. 2019d. *Annotation Specifics.* Available at: https://spacy.io/api/annotation#named-entities [Accessed: 1st September 2019].

Stack Overflow. 2017. *What is the difference between functional and non functional requirements?* Available at: https://stackoverflow.com/questions/16475979/what-is-the-difference-between-functional-and-non-functional-requirement [Accessed: 6th September 2019].

Stanford. 2019. *GloVe: Global Vectors for Word Representation.* Available at: https://nlp.stanford.edu/projects/glove/ [Accessed: 7th September 2019].

StatCounter. 2019a. *Global market share held by leading desktop internet browsers from January 2015 to June 2019*. Available at: https://www.statista.com/statistics/544400/market-share-of-internet-browsers-desktop/ [Accessed: Accessed: 8th August 2019].

StatCounter. 2019b. *Number of internet users worldwide from 2005 to 2018 (in millions)*. Available at: https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/ [Accessed: 4th August].

Van Greunen, D. et. al. 2011. A three-phase process to develop heuristics. In: *Proceedings of the 13th ZA-WWW Conference*, Johannesburg, South Africa, 14-16 September 2011.

W3C. 2002. *Usability – ISO 9241 Definition*. Available at: https://www.w3.org/2002/Talks/0104-usabilityprocess/slide3-0.html [Accessed: 1st August 2019].

W3C. 2019a. Available at: https://browserext.github.io/charter/ [Accessed: 2nd August 2019].

W3C. 2019b. Available at: https://browserext.github.io/browserext/#overview [Accessed: 2nd August 2019].

Wang, Alex. et. al. 2019. *Glue: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding.* [Preprint] Available at: https://arxiv.org/abs/1804.07461 [Accessed: 4th August 2019].

Wikimedia Foundation. 2019. *API: Main Page.* Available at: https://www.mediawiki.org/wiki/API:Main_page [Accessed: 5th August 2019].

Yadav, Vikas. Bethard, Steven. 2019. A Survey on Recent Advances in Named Entity Recognition from Deep Learning models. In: *Proceedings of the 27th International Conference on Computational Linguistics*, Santa Fe, New Mexico, USA, August 20-26 2018, pp 2145–2158.

Zhao, Lin. et. al. Event-Oriented Keyphrase Extraction Based on Bi-clustering Model. In: Rodrigues, João M. F. et. al. eds. *Computational Science – ICCS 2019*, Faro, Portugal, June 12–14 2019. pp. 207-220. doi: 10.1007/978-3-030-22750-0_16.

13 Appendix

13.1 JSON IR Object

This is an example of all the data gathered based from a user's input query (fig. 79). A JSON object similar to this is used to generate the context of the WhereFinder main display window.

```
{
  "textObj":
    {
    "tabId":INT, // Id of browser tab the user currently has open
    "text: STR, // Initial input text from the user
    "mouseX":INT, // Location of initial context menu event
    "mouseY":INT, // Location of initial context menu event
    "type": STR, // Type of message being sent to/from the text processor
    "processedText":  // Initial processing of user input text
      {
      "GCS":[], // Lat/lng coordinates (if there was any to extract)
      "label":STR, // Type of NE extracted from user input text
      "input":STR, // NE extracted from input text
      "locBase":STR, // Confirmed type of NE (checked by location DB)
      "runQueryU":BOOL, // Determines if the user needs to be queried
      "DBSearchResults": [ … ], // List of locations DB search results
      "iso":STR, // Part of full location name
      "localName":STR, // Part of full location name
      "geoType":STR, // Type of location (i.e. region, city…)
      "inputFull":STR, // Full location name
      "params":STR, // Query string of the URL to Google's Places API
      "placeReqURL":STR, // Full URL to Google's Place's API
      "googleKey":STR, // Unique Google developer ID key
      "placeID":STR, // Id for the location of interest from user query
      "wiki": // Data gathered on loc-of-interest from wikipedia
```

```
        {
            "newsNum":0, // Article number
            "url:STR, // URL to article page
            "title":STR, // Title of article
            "desc":STR, // Short description of article
            "date":DATE, // Date of publication
            "time":TIME, // Time of publication
            "img":URL, // URL to display image of the article
            "imgWidth":INT, // Width of the display image
            "imgHeight":INT // Height of the display image
        },
        {…}, // Up to N articles can be retrieved
    ],
"images": // Images gathered from Bing Image Search
    [
        {
            "imgNum":INT, // Image number
            "url":STR, // URL to image
            "host_url":STR, // URL of web page hosting the image
            "imgWidth":INT, // Width of the image
            "imgHeight":INT // Height of the image
        },
        {…}, // Up to N images can be collected
    ],
"msgTypeIn":STR, // Type of msg sent to/from the text processor
```

Figure 79: Example JSON object creating during the IR processes of the WhereFinder Extension.

13.2 Host Application Manifest

Below is an example manifest for a host application (fig. 80). The mangiest of the native Python application that handles the text processing for this project uses a manifest very similar to this.

```
{
  "name" STR, // Name of Host Application
  "description": STR, // Short description of Host Application
  "path":PATH, // Local absolute path to the host application
  "type": STR, // Type of communication to be used (Chrome only supports “stdio”)
  ”allowed_origins": [ … ] // List of STRs of extension ID’s that can use this app
}
```

Figure 80: Example JSON object host application manifest.

13.3 API Functions

For each API, the following functions were used (table. 11):

API	Function	Description
chrome.	chrome.extension.getURL	Converts a relative file path within an extension install directory to a fully-qualified URL. (Chrome, 2019k). Used to add URLs to HTML src tags and to fetch HTML from URL paths.
	chrome.contextMenus.create	Create a new item in Chrome's right-click context menu (Google, 2019v)
	chrome.contextMenus.remove	Removes item from Chrome's right-click context menu (Google, 2019v).
	chrome.contextMenus.onClicked.addListener	Adds listener to click events that occur with the context menu (Google, 2019v). Used to trigger specific events when the user interacts with the extension via the context menu.
	chrome.tabs.query	Retrieves specific information about the different tabs the user currently has open (Google, 2019w). Used to find which tab the user current had opened.
	chrome.tabs.executeScript	Used to Inject JavaScript code into a web page (Google, 2019w).
	chrome.tabs.sendMessage	Used to send messages from browser context scripts to the content scripts within specified tabs (Google, 2019w) and used to handle responses from those sent messages (callbacks).
	chrome.runtime.onMessage.addListener	Adds listener to 'message sent' events that occur across the extension when either 'runtime.sendMessage' or 'tabs.sendMessage' is used (Google, 2019x).

	chrome.runtime.onStartup.addListener	Adds listener to when the browser first starts. Used to add the extension to the context menu, and to initiate a connection with the Python scripts.
	chrome.runtime.connectNative	Initiates a connection with a native application on the user's host machine (Google, 2019x).
	chrome.runtime.sendMessage	Used to send messages from content scripts to browser context scripts (Google, 2019x).
google.	google.maps.Map	Used to create an instance of the 'Map' class. (Google, 2019y)
	google.maps.LatLngBounds	Create a latitude and longitude restriction that can be applied to a Map, which that map's viewport will not exceed (Google, 2019y).
	google.maps.places.PlacesService	Creates an instance of the 'PlacesService' class (Google, 2019y). Used to retrieve details on specific places (i.e. latitude and longitude).
	google.maps.places.PlacesService.getDetails	Retrieves details about the place identified by the given placeId (Google, 2019y).
	google.maps.places.PlacesServiceStatus.Ok	Variable from a PlacesService request which confirms the status return of the search.
	google.maps.LatLngBounds.extend	Add a new point to the latitude and longitude bounds which extends it (Google, 2019y).

	google.maps.LatLngBounds.fitBounds	Resets the viewport to contain the current given bounds (Google, 2019y).
	google.maps.Marker	Creates an instance of the 'Marker' class (Google, 2019y). Used to create markers for the map.
wikipedia.	wikipedia.search	Searches Wikiedia using a given query (PyPI, 2019).
	wikipedia.page	Fetches a specific wikipedia page. Used after a Wikipedia search to inspect specific pages from the search. The information gathered from each page was; its URL, its images, its title, and the text content of the page.
msrest.	msrest.authentication.CognitiveServiceCredentials	Creates an instance of the "CognitiveServiceCredentials" class. Passed into NewsSearch API and ImageSearch API objects to identify a unique Azure subscription so that using each API is authorized.
azure.	azure.cognitiveservices.search.NewsSearchAPI	Creates an instance of the 'NewsSearchAPI' class (Microsoft, 2019c).
	azure.cognitiveservices.search.ImageSearchAPI	Creates an instance of the 'ImageSearchAPI' class (Microsoft, 2019d).
	azure.cognitiveservices.search.NewsSearchAPI.news.search	Used to request relevant news articles from Bing (Microsoft 2019e). From each article returned, the URL, article name, description, date published, thumbnail URL, thumbnail height, and thumbnail width, were extracted.
	azure.cognitiveservices.search.ImageSearchAPI.news.search	Used to send a query to Bing and get back a list of relevant images (Microsoft, 2019f). From each result returned, the following data was extracted: image URL, host page URL, image height, and image width.

Table 2: Functions used from remote APIs

13.4 RTLX Test

Geolocation Information Retrieval Test

Subjects:

10 participants took this test between Wednesday August 29th 2019 – Wednesday August 5th 2019. The average age of a participant was 29. All of those who took part were familiar with web browsers and understood the prompt clearly.

Prompt:

The participant is given a web browser with 10 opened tabs, each one with a different news article in it pertaining to a different location around the world.

For each article in each tab, the participant must find two fragments of information pertaining to the location that the article is about:

1. A map showing the location.
2. Some additional information about the location; either from Wikipedia, or another news article.

Tests:

There will be two tests which use this prompt:

1. The first will require the user to complete the prompt with only a web browser.
2. The second will require the user to complete the prompt by leveraging the WhereFinder extension within the web browser.

After both tests, the participant will complete two questionnaires – one for each test taken.

Figure 81: RTLX usability test guide.

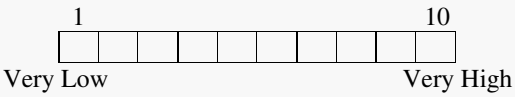
NASA Task Load Index Assessment Tool

Activity: Geolocation Information Retrieval Test – Without WhereFinder

Test Subject Num:

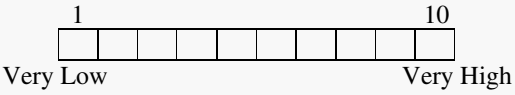
Mental Demand:

How much mental and perceptual activity was required (e.g. thinking, deciding, calculating, remembering, looking, searching, etc)? Was the task easy or demanding, simple or complex, exacting or forgiving?



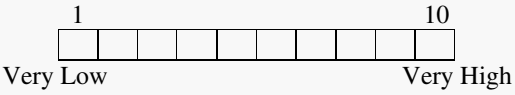
Performance:

How successful do you think you were in accomplishing the goals of the task set by the experimenter? How satisfied were you with your performance in accomplishing these goals?



Effort:

How hard did you have to work to accomplish your level of performance?



Frustration:

How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, replaced and complacent did you feel during the task?

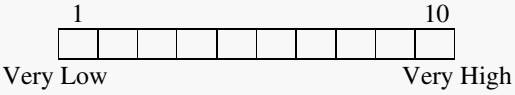


Figure 82: RTLX usability test A.

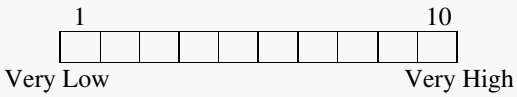
NASA Task Load Index Assessment Tool

Activity: Geolocation Information Retrieval Test – With WhereFinder

Test Subject Num:

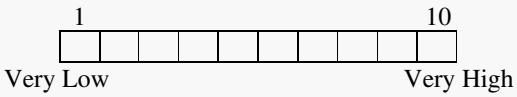
Mental Demand:

How much mental and perceptual activity was required (e.g. thinking, deciding, calculating, remembering, looking, searching, etc)? Was the task easy or demanding, simple or complex, exacting or forgiving?



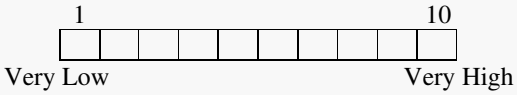
Performance:

How successful do you think you were in accomplishing the goals of the task set by the experimenter? How satisfied were you with your performance in accomplishing these goals?



Effort:

How hard did you have to work to accomplish your level of performance?



Frustration:

How insecure, discouraged, irritated, stressed and annoyed versus secure, gratified, content, replaced and complacent did you feel during the task?

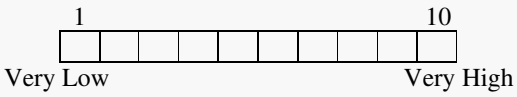


Figure 83: RTLX usability test B.