

Creating an online adblocker

One Semester Individual Project

Final Year Project

Final Report

CM3203

Author: Oliver James Robert Storey-Young

Student Number: C1635943

Supervisor: Padraig Corcoran

Moderator: Federico Liberatore

15/05/2020

School of Computer Science and Informatics, Cardiff University.

Abstract

When browsing the internet, it is all too common to be bombarded by adverts that will hinder and alter their browsing experience. Over the past few years, the number of adverts and unwanted traffic loaded when you visit a website has increased to a ridiculous amount. Adverts can often be unsafe, collect personal information and have even been known to influence political elections.

This project aims to create a reliable and effective adblocker that will be able to detect and circumvent adverts that may be displayed on any given website. Since there are already adblockers available, time will spend time researching the techniques and algorithms that have previously been used by them to help identify adverts on a web page. As a result of that research and this project, an adblocking application shall be created.

The outcome of this project involved the creation of a Google Chrome extension that can be used to block and identify adverts while browsing the internet. The extension makes use of a concise handful of options and settings to accompany it. The finished product could later be slightly adapted to use and integrate a filter or block list published by third parties found online.

The following report will outline and evaluate thoroughly the processes that have been used to complete this project.

Acknowledgements

I would like to give a big thanks to all my friends and family who gave continued support and motivation throughout the duration of my project. I would also like to express my gratitude to Dr Padraig Corcoran for his continued guidance and help throughout the project.

Most of this project was completed during the lockdown brought on by the coronavirus outbreak of 2020. I would like to add further thanks to all the NHS staff, key and essential workers who have risked their lives to keep the country running during these times.

Contents

<i>Abstract</i>	1
<i>Acknowledgements</i>	2
<i>Table of Figures</i>	5
1 <i>Introduction</i>	7
1.1 Project Overview	7
1.2 Project Aims	8
1.2.1 First Aim	8
1.2.2 Second Aim	8
1.2.3 Third Aim	8
1.2.4 Fourth Aim	9
1.3 Project Scope	9
2 <i>Background</i>	10
2.1 Existing solutions	10
2.1.1 Adblock — best ad blocker [Fig. 2]	10
2.1.2 Adblock Plus - free ad blocker [Fig. 6]	11
2.1.3 uBlocker - Ad Block Tool for Chrome [Fig. 9]	11
2.2 Evaluation of the current solutions	11
2.2.1 The Balance of too many options and not enough options	11
2.2.2 Default adblocking effectiveness	12
2.3 Constraints and Limitations	13
3 <i>Approach</i>	14
3.1 Setup	14
3.2 Research	14
3.2.1 WebRequests	15
3.2.2 Storage	15
3.2.3 Adblock Plus	15
3.3 Development	15
4 <i>Specification and Design</i>	16
4.1 Popup	16
4.1.1 Users Perspective	16
4.1.2 System Behaviour	17
4.1.3 Architecture	18
4.1.4 Constraints	18
4.2 Storing Data	18
4.2.1 Users Perspective	18
4.2.2 System Behaviour	19
4.2.3 Architecture	19
4.2.4 Constraints	20
4.3 Selecting Elements	20
4.3.1 Users Perspective	20
4.3.2 System Behaviour	20
4.3.3 Architecture	21
4.3.4 Constraints	21
4.4 Blocking Elements	21
4.4.1 User Perspective	21
4.4.2 System Behaviour	21
4.4.3 Architecture	22
4.4.4 Constraints	22
4.5 Selecting Domains	22
4.5.1 User Perspective	22
4.5.2 System Behaviour	23

4.5.3	Architecture	23
4.5.4	Constraints.....	23
4.6	Blocking Domains	24
4.6.1	User Perspective	24
4.6.2	System Behaviour	24
4.6.3	Architecture.....	24
4.6.4	Constraints.....	24
5	Implementation	25
5.1	Popup	25
5.1.1	Interface	25
5.1.2	Data population.....	25
5.1.3	Events	26
5.1.4	Tables	26
5.2	Storing data	27
5.2.1	Data Structures.....	27
5.2.2	Setting Data.....	27
5.3	Selecting Elements	28
5.3.1	Highlighting elements.....	28
5.3.2	Preparing elements for storage.....	28
5.4	Blocking Elements	29
5.4.1	CSS injection	29
5.5	Selecting Domains	29
5.5.1	Context menu	29
5.5.2	XMLHttpRequest.....	30
5.6	Blocking Domains	31
5.6.1	Blocking domains from external list	31
5.6.2	Blocking domains from manual input.....	31
6	Results and Evaluation	33
6.1	The realisation of how ad-blockers work and what that means for this project	33
6.2	Comparison against other similar software	34
6.2.1	Blocking comparison.....	34
6.2.2	UX & UI comparison	34
6.3	Evaluation against requirements and Test Cases.....	36
6.3.1	Test Cases for First Aim	36
6.3.2	Test Cases for Second Aim	38
6.3.3	Test Cases for Third Aim	39
6.3.4	Test Cases for Fourth Aim.....	40
7	Future Work.....	42
7.1	Full validation	42
7.1.1	Domain Loading	42
7.1.2	Manual Domain Typing	42
7.2	Error handling	42
7.2.1	Error logging	43
7.2.2	Try Catch.....	43
7.3	Code optimisations	43
7.3.1	Core components.....	43
7.3.2	Improved data fetching	43
7.4	Integrating with EasyList	44
8	Conclusion.....	45
9	Reflection on Learning.....	46
10	References.....	47
11	Terminology.....	49
13	Figures.....	51

Table of Figures

[Fig. 1] A website covered in malicious adverts with the intent on tricking the user.	51
[Fig. 2] The AdBlock Chrome extension store listing.	51
[Fig. 3] The AdBlock extension popup.	51
[Fig. 4] The AdBlock settings page overview.	52
[Fig. 5] The AdBlock still showing some adverts on "reddit.com"	52
[Fig. 6] The Adblock Plus Chrome extension store listing.....	53
[Fig. 7] The Adblock Plus extension popup.....	53
[Fig. 8] The Adblock Plus settings page overview.	54
[Fig. 9] The uBlocker Chrome extension store listing	54
[Fig. 10] The uBlocker extension popup	55
[Fig. 11] The uBlocker settings overview	55
[Fig. 12] Overview of TortoiseGit.....	56
[Fig. 13] Screenshot of my Trello boards during development	56
[Fig. 14] Display of <hr/> tag used to divide sections in extensions popup	56
[Fig. 15] Display of <tables> used to display blocked elements and domains	57
[Fig. 16] Code showing HTML sections being used inside popup.html	57
[Fig. 17] Code showing HTML first section inside of popup.html.....	58
[Fig. 18] Code showing HTML second and third section inside of popup.html.....	58
[Fig. 19] Code showing HTML fourth section inside of popup.html.....	59
[Fig. 20] Code showing initially set values from storage inside popup.js.....	60
[Fig. 21] Code showing callback via document ready event inside popup.js	60
[Fig. 22] Code showing example of DOM selection and value setting inside popup.js.....	61
[Fig. 23] Code showing example of click and change handlers inside popup.js.....	61
[Fig. 24] Code showing message being sent and a value updated inside popup.js	61
[Fig. 25] Code showing table generation inside popup.js	62
[Fig. 26] Code showing initial data structure being generated on extension install inside background.js.....	62
[Fig. 27] Code showing element storage data structure being created inside SelectElement.js	62
[Fig. 28] Code showing Get Set technique to modify data inside popup.js	62
[Fig. 29] Code showing popup.js sending a message to the SelectElement.js	63
[Fig. 30] Code showing SelectElement.js Listening for messages	63
[Fig. 31] Code showing custom CSS being added to the head of the current web page from SelectElement.js.....	63
[Fig. 32] Code showing mousemove jQuery event in SelectElement.js	63
[Fig. 33] Code showing SelectElement.js click event	63
[Fig. 34] Code showing overview of SelectElement.js StoreElements() function	64
[Fig. 35] Code showing SelectElement.js variables being assigned	64
[Fig. 36] Code showing CSS.escape() in SelectElement.js.....	64
[Fig. 37] Code showing BlockElement.js	65
[Fig. 38] Code showing BlockElement.js DOMSubtreeModified listener	66
[Fig. 39] Code showing BlockElement.js injectCSS function	66

[Fig. 40] Code showing context menu creation in background.js	67
[Fig. 41] Code showing ResetManualDomainListeners() function in background.js.....	67
[Fig. 42] Code showing ResetWebblock() from background.js.....	68
[Fig. 43] Code showing request being made to external web page from background.js.....	68
[Fig. 44] Code showing onBeforeRequest in background.js	68
[Fig. 45] Code showing background.js function WebDomainBlock used by onBeforeRequest	68
[Fig. 46] Code showing ResetManualDomainListeners inside of background.js	69
[Fig. 47] Code showing ManualDomainBlock function used in onBeforeRequest for manual domain filtering in background.js	69
[Fig. 48] Code showing background.js onBeforeRequest for manual domain filtering	69
[Fig. 49] Code showing Blocked.js a file used to ship a default domain blocklist	70
[Fig. 50] AdBlocker verification example 1	70
[Fig. 51] AdBlocker verification example 2	70
[Fig. 52] Finished Interface of Chrome extension.....	71

1 Introduction

1.1 Project Overview

Websites have become littered with adverts. Adverts can come in many shapes and sizes, but all have one thing in common, to take something from you. As internet traffic has boomed over the past several years so have adverts. [1] Adverts have evolved from simple pictures to embedded frames that can track you, collect your information and can even have malicious intent. [2][Fig. 1] Some websites are so heavily plastered with adverts that load times can be reduced and website functionality drastically lowered.

The aim of this project is to allow the user to easily manipulate their own browsing experience by culling adverts they do not want to or should not be seeing. Although ad-blockers already exist online they are often hard to customize for everyday users. The project aims to create an application of some sort that can be used to block adverts. Users will be able to manually select adverts to be blocked or opt to use predefined ad blocking lists that can be provided or added later to the blocker.

The application will offer the user simple yet powerful tools that allow for the manipulation of the websites they are visiting. Adverts that the user wishes to be blocked will be synced to their Google account and therefore will persist through any of their signed-in instances of Google Chrome. The application will offer two main streams of blocking that are element blocking, and domain blocking. Simply put, element blocking will allow for specific items on a web page to be hidden, while domain blocking will allow for adverts to be blocked before they are even loaded. Combining these two approaches will allow for a flexible and effective blocking extension.

This project will follow the development of an extension for Google Chrome that can be used to block adverts on websites inside of their browser. The finished product will allow the user to manually select elements on the page to be blocked as well as offer a variety of address filtering methods for the user to block advert requests. The extension should be simple to use and easy to understand for the everyday web user. Although similar extensions already exist, the aim is to create a more user friendly and accessible version that will allow users to customise their filtering more simply.

My personal aims while completing this project are to increase my knowledge of JavaScript and to learn how to create and use the Google Chrome extension API¹. This project will allow me to greatly expand my knowledge in both through use of documentation, trial and error and questioning. By the end I should have a clear understanding of the components that make up a Google Chrome extension and the different functions and API endpoints I can use from Chrome.

¹ https://developer.Chrome.com/extensions/api_index

1.2 Project Aims

The project being undertaken consists of a handful of complex parts that will allow for the overall ad-blocking experience. Therefore, to effectively plan and understand the road ahead the project has been split into concise segments. Each segment holds a key role in the overall aim of this project therefore having detailed and well thought out acceptance criteria will help make sure they are met properly. As the project will be using the Agile² development process it will not be inconveniencing to make slight changes to the projects aims as needed whenever a major problem is encountered, or a new path discovered somewhere else.

1.2.1 First Aim

The extension allows for the manual selection, storage and blocking of elements from any website the user wishes to block elements.

Acceptance Criteria

- It is obvious to the user which element they are selecting.
 - A reference to any item the user selects is stored (across sessions).
 - When a website loads, stored references are checked and applied to the loaded website to hide elements.
 - The user can edit the list of blocked elements they have selected.
-

1.2.2 Second Aim

Adverts can be blocked via domain filtering selections made by the user.

Acceptance Criteria

- The user can select domains on an already loaded page to be blocked upon reload.
 - The user can manually type in domains which they wish to be blocked.
 - The user can edit the list of blocked elements they have selected.
-

1.2.3 Third Aim

Adverts can be blocked via domain filtering lists that the user can load or present to the extension themselves.

Acceptance Criteria

- The user will be able to specify some sort of path to an already created filter list that they wish to use in their blocker.

² <https://www.agilealliance.org/agile101/>

- The user can choose whether to enable this feature at any time.
- The provided list works without conflict towards any other filtering.

1.2.4 Fourth Aim

The user can easily and without confusion edit the settings of their application.

Acceptance Criteria

- All settings are loaded into the extension's popup window, not in additional html files or background pages.
- The settings have a good UX and will be understandable to users.
- All settings are saved seamlessly upon change

1.3 Project Scope

The original scope for the project involved creating an extension for Chrome, porting it over to other browsers like Internet Explorer and even looking at blocking domains through the routers firewall. However, as the project took foot, it was quickly discovered how hard it was going to be initially with learning another language and API. As the project ended the scope shrunk greatly. The scope of the project changed to being blocking adverts inside of the Google Chrome browser. Initially the project was too ambitious with what It set out to do and reducing the scope meant; less pressure, better code, more time learning less languages and overall being able to think about how the project could be improved. Overall, the ideas behind the adblocker have stayed the same, with the ideas for blocking elements being implemented, the domain filtering the project initially set for the firewall being implemented straight into the browser.

2 Background

2.1 Existing solutions

When searching the Chrome extension store it is easy to see that there is not a shortage of ad-blockers out there for you to download for free. But what makes them different from each other, why is one better than another one. This short evaluation of existing solutions aims to look at some of the features and ideas behind a few of the ad-blockers. Gathering and evaluation these will help me make better choices when creating my own adblocker.

This short evaluation will look at some of the options available with each of the blockers as well as carrying out some simple tests to see how well they block adverts across a couple of websites. Most of the ad-blockers appear to use the same underlying technologies so it will be interesting to see how they differ in action

2.1.1 AdBlock — best ad blocker³ [Fig. 2]

<http://getadblock.com/> | 10,000,000+ Users

Features: Offers a free or premium ad-blocking experience, allow non-intrusive ads, show statistics of adds blocked, acceptable ads, custom filters, anti-circumvention measures, cryptocurrency mining protection. Open source and has a good support suite. + many more options. [Fig. 4]

Pros, cons, and thoughts: Firstly, there are so many options for this adblocker, this is both good and bad. It boasts great customisation but can be a little overwhelming if you are not totally sure what you are doing. The popup is very simple to use and understand, also quite compact, however does not really have many options on it. [Fig. 3] The installation process on this one also takes a little longer than the others and does prompt you to donate once it has installed. Following on from the previous point the upgrade button is always displayed on the popup which can be a little annoying. With this adblocker when tested on *reddit.com* some ads were displayed, disappeared, and reappeared.[Fig. 5]

Blocked:

R= {Claimed: 5, Visibility: Poor}

S = {Claimed: 13, Visibility: Poor}

³ <https://Chrome.Google.com/webstore/detail/adblock-%E2%80%94-best-ad-blocker/gighmmpiobklfepjocnamgkkbiglidom>

2.1.2 Adblock Plus - free ad blocker⁴ [Fig. 6]

adblockplus.org | 10,000,000+ Users

Features: Whitelist website, whitelist page, basic statistics, different languages, tracking blocking, custom filters. Extensive help suite with lots of documentation.

Pros, cons, and thoughts: This adblocker have a good amount of options, not too many nor too few. The popup is very simple [Fig. 7] and therefore most options are hidden away in the settings tab. [Fig. 8] The UI for the adblocker is very clean and easy to understand. The adblocker installed fast and is totally free. When loading *reddit.com* no ads we are displayed to me. Overall, this adblocker is very simple to use and has a good amount of options that are easy to understand.

Blocked:

R = {Claimed: 3, Visibility: Poor}

S = {Claimed: 12, Visibility: Poor}

2.1.3 uBlocker - Ad Block Tool for Chrome⁵ [Fig. 9]

400,000+ Users

Features: Whitelist website, pause adblocker, statistics, allow non-intrusive add items through right click.

Pros, cons, and thoughts: This adblocker has a similar popup look to *AdBlock — best ad blocker*. It is very clean and shows you exactly what you need to see. [Fig. 10] Overall, the options available for this adblocker are very poor. However, it provides a simple adblocking tool for the user who does not want any customisation etc. The extension has a much smaller user base. Once again, the options panel is hidden away in another page and is not very eye catching but cuts straight to the point. [Fig. 11]

Blocked:

R = {Claimed: 4, Visibility: Excellent}

S = {Claimed: 8, Visibility: Good}

2.2 Evaluation of the current solutions

2.2.1 The Balance of too many options and not enough options

Options are important when choosing the adblocker for you, the user needs to be in control of the content that is being allowed and blocked from the websites they visit. Finding a balance between needed and extra options is also something to think about as you do not want users to be overwhelmed or confused by the options that are available to them.

⁴ <https://Chrome.Google.com/webstore/detail/adblock-plus-free-ad-bloc/cfhdojbkjhnlbpkdaibdcddilifddb>

⁵ <https://Chrome.Google.com/webstore/detail/ublocker-ad-block-tool-fo/lmknjkanfacinilblfiegkpaipcpjce>

Firstly, [2.1.1] had a huge range of options available which meant that the user was going to be able to fully customize their browsing and adblocking experience. However, it can be very overwhelming when looking at the options as it is not clear what a lot of them mean or what they could do for the user. Also having the premium and the options available with it constantly pushed in the face of the user is annoying. Overall, this adblocker had too many options available with it.

Secondly, [2.1.2] had a good amount of options, it felt like the options that were available had been carefully curated to keep the user informed and not overwhelmed. Just like the previous the options were tucked away in the settings page. In turn, this adblocker had a brilliant balance between options and simplicity.

Finally, [2.1.3] lacked a lot of options but seemed very simple and intuitive to use. This adblocker was missing a lot of features that I felt were needed such as being able to load in a custom filter list. This adblocker could really do with a few more options but if you are looking for something very simple and lightweight this may be a good option.

Overall, [2.1.2] had the best balance between customization and simplicity with its careful balance of options available. It could have seen more options available on the popup though as all three ad-blockers had them hidden away in the settings page.

2.2.2 Default adblocking effectiveness

An important factor to think about with these ad-blockers is how well they perform once they have just been installed, so no options being changed, the program in its default state.

A small test was carried out across two websites:

R = <https://www.reddit.com/>

S = <https://smallbusiness.chron.com/affiliate-marketing-tracking-tools-67773.html>

For each website records were taken of how many adverts are supposedly being blocked by the adblocker and then a personal view on how well it had done to remove the adverts from the page. To do this, both pages were viewed with and without the ad-blockers to see how well they did.

Firstly, [2.1.1] Boasted the highest numbers being blocked but failed to block the ads which we are visible to me on the page. This happened to both websites that were tested. Taking an educated guess this means tracking data in the background is being blocked but it has not recognised or filtered out the adverts that were seen.

Similarly, with the second, [2.1.2], both websites reported having a high number of ads blocked on the page, but all adverts were let through as usual. This is unfortunate for both bigger names ad-blockers to be letting through so many adverts I rated both websites as poor just like [2.1.1].

Finally, the smaller adblocker [2.1.3] pulled through and did a really good job at blocking the adverts. This adblocker reported much lower number of adverts being blocked but then went ahead and blocked all but one of the visible adverts that could be seen.

To conclude the results from the above test, it was discovered that the smaller of the blockers with much less options available had come out and blocked many more adds than it is bigger more popular competitors. There are many reasons this could be: Perhaps the other two need some setting up to work more effectively, Block may have a newer filter list.

2.3 Constraints and Limitations

One major constraint that will be faced when undertaking this task is being less informed with the technologies that this project will be using. To begin with my knowledge of JavaScript. This is a language that is unfamiliar to me and was only used previously at university for a short while. This language is used often by many people so perseverance and the guidance of documentation that can be found online will be key. Following this previous statement, the project will also be dabbling with jQuery⁶, a shorthand for JavaScript that should help make coding simpler. Similarly, a new environment to me will be used, Chrome. This is the first time developing a Chrome extension and working with the API, so it is unknown what is attainable and what is realistic. Again, for all these documentation and further reading will help lessen this constraint.

One constraint that has been encountered that has affected this project is the coronavirus. At the time of writing this report university has been closed for quite some time. [3] Universities have closed, society has shut down. The resources that are available when normally undertaking a university degree are not available. Computer labs, Libraries, and the ability to meet up with supervisors, peers and tutors have all been removed, potentially threatening the outcome of this project. As this project has progressed it has been noticeably difficult, being out of routine and stuck in a weird situation where no one is safe has impacted the ability to work properly will have had slight effects on this and many other projects.

The final constraint to this project is to do with filter lists that are used by ad-blockers. After doing research it is apparent that ad-blockers mainly use other people's lists as filters rather than their own. [4] Some of these lists are public [5] and can be used but are written in schemas and formats that are not fully explained and may therefore be hard to implement. [6] It is the aim of this project to at least allow for those filter lists to be loaded and used at least, as if they we are processing the filters correctly.

⁶ <https://jquery.com/>

3 Approach

The approach that was taken towards the project was Setup, Research and Develop. This approach allowed for all the core pieces of application to be put into place before cycling back and forth through research and development. As each week came by and work began on the aims of the project, research was made into each individual part and used to develop that piece. This approach worked well as it worked well with the flexibility and granularity of the Agile methodology. As the project move forward less and less research was needed, as familiarity grew, which left more time for development.

3.1 Setup

The first part to the approach was the setup. During this time, the project had all its core components created and setup which would allow for everything else to flow nicely. To begin with was the task of setting up version management. Git ⁷was chosen as it is easy to use and widely accessible. It was chosen that it would be hosted on a private repository on GitHub⁸. Having the project hosted here meant it could be worked at from any computer on any device and the files would never be lost. The client that was chosen to mediate this was TortoiseGit⁹ as it provides an easy and simple to use interface. [Fig. 12] Git played a big role in this project and allowed for rapid development through trial and error.

Next on the setup phase was to create the initial Chrome extension. I first had to learn what files would be needed and how they interact with each other. Over the course of a day the barebones application was created and an understanding of what each key individual file did was gained. Over the course of this project the skeleton evolved slightly with new discoveries and ideas coming into play. This played an important part as gaining a better understanding of how each component worked early allowed for progression with more pressing items.

The final Items that was part of the setup was a task board and notes page. Trello was used as an easy way to divide the project into tasks and subtasks that would allow for progress tracking. [Fig. 13] Trello¹⁰ is an online task board that works well with the Agile methodology. [7] Being online, like Git it is useful because it could be accessed from any device and keep development up to date at any point. [8] To keep notes a work in progress word document was created where anything from failed ideas to thoughts could be placed to help aid the writing of this report.

3.2 Research

Using a language and API that was not so familiar meant a lot of research was needed continually throughout the development. Research was carried out and then put into practice in the development stage before that small cycle started again with the next component. There were three main areas of research that were carried out and are listed below.

⁷ <https://git-scm.com/>

⁸ <https://github.com/about>

⁹ <https://tortoisegit.org/about/>

¹⁰ <https://trello.com/en-GB/about>

3.2.1 WebRequests

WebRequests is a crucial part of the Chrome API that would be needed in the development of the extension. WebRequests allows for user intervention at several locations during the life cycle of an incoming or outgoing request. [9] Using this would mean requests could be filtered and blocked to stop adverts from being loaded onto the webpage from external sources. On the API documentation page there are detailed and descriptive examples of how to use the code in a variety of scenarios. This and the overall Chrome API documentation was a big help and made development a lot simpler.

3.2.2 Storage

The Chrome API has an easy to use storage facility located in Storage.Sync. This storage facility allows for data to be stored inside of the browser, attached to your Google account. Using this meant that users settings, choices and block preferences could all be saved easily and loaded to anywhere they log in. [10] The storage API saved lots of time and effort as originally the plan was to save these things to documents. Storage.Sync has two simple functions, Get and Set. Get returns an object and set allows you to set an object. Combined these two powerful snippets helped create basic foundations of how the extension would solely function.

3.2.3 AdBlock Plus

AdBlock plus is one of the already popular ad-blockers that is available as a Chrome extension. They host lots of material on their website that talks about how their extension works, some of the algorithms and techniques behind the scenes. [11] [12] All their code is open source too. [13] Combining these elements meant there was a good starting point and reference location to help throughout. Although greatly useful some documentation and examples were missing which meant I was not always relying on this and constantly came up with my own solutions.

3.3 Development

The final part of the chosen approach was the actual development. Developing was straight forward and coupled with Git, allowed me to be flexible with my attempts as branches were there to fall back on. Development would differ per task but would normally consist of a front end to back end style approach where possible. Since this extension is focussing on having concise and available options to the user, there was lots of front-end options that needed hooking up. The aim was to create one front end object first, connect it to storage and then hook-up the background logic after. Creating each option like this meant that it would be simple to trouble shoot and each commit was related to just a single item. Finally, as each bigger stage would be completed an overhaul including refactoring, reordering, and minifying could be applied to make the code better. Throughout the process there would be many problems and challenges. Thanks to the approach set out and tools that were being used to help preserve the projects integrity the work could be done stress free and effectively code

the project. Overall, the approach was a good fit for the extension that was being created and worked nicely with the agile framework.

4 Specification and Design

The development process of this project is broken down into these key stages

- A popup window that allows the user to interact with the extension.
 - A storage system that will allow for user's settings to be stored and accessed across all sessions.
 - A script that will allow the user to interact with their current website to select items that need to be blocked
 - A script that will allow previously selected blocked items to be removed from the website
 - A script that will allow the user to select domains that need to be blocked
 - An importing script that will allow the user to load in a filter that can be used to block domains
-

4.1 Popup

4.1.1 Users Perspective

At the core of most Chrome extensions is the popup. The popup can be accessed by clicking on the extension's icon found just after the search bar inside of Chrome. The popup is primarily used to allow the user to interact with the back end of the extension. Often you can find settings and statistics inside this panel. Often the popup will link to a background page that allows the user to get further customisation.

With lots of extensions having their settings muddled between both the popup and background page and from my research, it was decided that all the options would be kept inside of the popup to make life simpler. This meant creating and mapping out what would be on this popup was one of the first steps. Once each of the options had been added, each of the scripts and code snippets that would run from and because of those options would be coded.

The popup would be split into three smaller sub sections that would group options and settings together. The groups are split with lines so that the distinction between them can be made. The group of options nearest the top of the popup that the user can see contains the options that allow whitelisting and a change of element hiding styles. The option to run on the page is located right at the top which means upon opening the popup can quickly be used to toggle the adblocker on and off. It uses a checkbox and is therefore easy for the user to understand. The next item is a set of radio buttons that allows the user to select whether blocked items are being collapsed or hidden. This small but concise group allows the user to quickly modify how items are being displayed on the website.

The following group contains most of the actual blocking interface. These options are primarily made up of buttons and allow the user to store items and domains that they wish

to block. Starting at the top of this group is a button that allows the user to start selecting items on the web page to be blocked. When this button is clicked the user will see a visual change on their website that allows them to select items. The next two options both buttons and allow for the user to reset in bulk items that they have previously selected for either the website they are on, or all websites they have data stored for. The next button sits next to a textbox and allows for the user to input a domain that they would like to be blocked. The user must first type in the textbox a domain string, before hitting the button to store it and refresh the page. The input textbox has a placeholder string that denotes the format of the string that needs to be entered. The penultimate option in this section allows for the user to load in an existing filter list via a URL. The option consists of two buttons and a textbox. The textbox can be used to enter the URL, the button on the right labelled Save, stores that URL in the storage and finally the button on the left enables and disabled this option. The save button and input textbox are both toggled on and off via the button to the left that enables and disables this option. The final option in the group is the default filter checkbox. This checkbox can simply be used to enable or disable the default domain filter list that is shipped with the extension.

The final group on the popup contains two dynamically created tables. Both tables look the same and have a header titling each one. The first of the two lists in rows all the elements that have manually been blocked on this page. The user can then click those elements to unblock them and return them to the website. The second table has the same functionality as the first but instead lists all the domains that have manually been blocked. Both tables display a count to the user, so they do not need to manually count how many items are in the lists. Each of the tables has a maximum height too so the tables will be scrolled more so than the popup itself.

Overall, I believe the design is simple and easy to follow for all users. The grouping has allowed for similar options to be put near each other in an obvious way. Coloured and bold text has also been used to emphasis certain parts of the UI and is even used as feedback in some places. All the options fit onto the popup as a scrollbar is used to catch the slight overflow and allow the user to scroll once some items have been added to the tables.

4.1.2 System Behaviour

The popup works primarily through JavaScript as its back end and uses a mixture of DOM events and the Chrome API to act. To begin with the HTML document is loaded with all the options set in their default states. JavaScript DOM events are used to listen for the page being fully loaded and then calls made to the Chrome API retrieve all the stored settings options. It is at this point that all the options are populated correctly – meaning the user will see what they have previously saved.

Similarly when the user interacts with the options on the HTML page their actions are listened to by JavaScript, which is monitoring for certain clicks on items, or changed events on checkboxes and will send data to the Chrome API to store these changes. When options are changed, the page the user is currently on will be reloaded so that these modifications can take effect. Some options will keep the popup open and therefor will actively modify the contents of the popup when they are changed to ensure the user gets feedback from the

interaction. An example of this is when a domain is removed, the popup stays open and the domain will be removed programmatically from the list, without the popup refreshing and repopulating

Another way in which the popup behaves is through messaging. Messages can be sent through Chrome to other script. These messages act as a way for the *popup.js* to interact with the *background.js* or the JavaScript of the page that is currently loaded. This can be seen through the reload function that will send a message from *popup.js* to the currently active page and tell it to refresh. Overall, the popup works by getting and setting stored data into Chrome storage and notifying other pages when something needs to happen.

4.1.3 Architecture

The architecture behind the extension is not too complicated and uses the traditional web languages as well as a couple of additional packages. HTML and CSS were both used in combination to create the basic outline for the application. Once this skeleton had been created JavaScript was added in its own file as well as exposing the interface to jQuery to make JavaScript coding more efficient.

JavaScript was heavily used in the popup as a means of loading in data from the storage, which will be talked about later. JavaScript was also used as a method of adding feedback to the user about their actions. An example being when the user enters an incorrect or accepted string into the domain blocking textbox.

Bootstrap was later added to add a cleaner style to the popup. Bootstrap is a premade CSS file that can be added to any project to style elements. The card class from bootstrap was used to house the tables located in the third group of options. Combining bootstrap with our own custom style sheet allowed for quick and simple customisation to the way the popup looked.

4.1.4 Constraints

As this project has a short deadline and the bulk of the work will be focussing on actual logic and code the UI is not quite as important of a task and therefore will be taking the back foot. This means that the options available may not be as intuitive or easy to understand as I would have liked but will still have the full functionality that I am intending. UI and UX design are primarily driven through user feedback and immense testing and this is not a priority.

4.2 Storing Data

4.2.1 Users Perspective

Storing data will be straightforward to the user. They will not have to worry about what data is currently stored behind the scenes. This data will be viewed by the user regularly but in its finalised format instead of raw items. This means the user will see settings inside of the settings window, checkboxes ticked etc, rather than interfacing directly with the data itself (seeing a stored true or false etc.) The user may store data only through the provided outlets, again, checkboxes, textboxes, and gestures such as clicks. This means the user is not able to compromise the data by accident or mess with the settings in a way which was not intended.

Data about stored domains and stored elements can be seen through the popup. Buttons are also available in the popup to help modify what is stored.

4.2.2 System Behaviour

The system will work by pushing and pulling data using the Chrome storage API. Data in Chrome can be stored using `Storage.Sync` which allows data to be synced to the Google account it came from, assuming extension syncing is enabled. The first method is the `get` method. When using the `get` method we can provide a string that we would like to look for. This means if the user clicks a button that needs one item, we do not have to get everything and can just retrieve the value they are looking for. All of this can be done using a call back function that will run once the request of data has been retrieved. This process is simple and easy to understand. If retrieving all data from the storage a JSON object will be retrieved, however if it is a specific item it will come in its stored format.

The other way in which the storage API will be used is to store data. Data can be stored by calling the `set` function and providing it with an object to store. An optional call back can be added to allow extra code execution to be done once the data has been fully set. It is important to note that most of this API is asynchronous and will therefor continue executing other code while it waits for its results. When setting data, we cannot just modify an item that is already stored. To modify an item that we have previously stored we will have to call the `get` function with a call back. Once this call back happens, we can call the `set` method and pass in a modified version of the data we received from the `get` function. This can be very annoying but is not very taxing overall.

4.2.3 Architecture

The architecture behind the storage of data is mainly how we format of the data we are storing. Data is stored inside of Chrome using JSON objects. JSON works seamlessly with JavaScript and will be easy to index and extract the items I need. The basic Idea is to store three main objects inside of the initial JSON object. These objects will be the blacklist, whitelist, and settings objects. Dividing the initial object up into these three sub objects should allow better data retrieval and a simple to use structure. It is important to note in JSON anything can be stored and nested, so objects in objects is fine.

The blacklist will be split up into two sections. The first section of the blacklist will denote which domains should not be allowed access from the browser and should be an array. The second item will be the elements that need to be blocked on a web page, another array. Storing them like this will mean it is simple to gather all the items which are “blacklisted” or bad and we do not want.

The second part is the whitelist. The whitelist will be used to store which websites and elements should be allowed to run on certain websites. This is important as we need to allow the user to select elements to be blocked but may need to be allowed on another website. It will be connected to the option that asks the user if they want the adblocker to run on this page.

The final part of the storage is the settings object. Quite simply this will be used to store the different settings data. These will primarily be true and false options that allow the user to switch things on and off. Such as using the default filter list or using a web filter list. Other options may use strings for storage or any form they see fit.

4.2.4 Constraints

There are a few constraints when it comes to data storage. Firstly, is the amount of data I can store. The Chrome API boasts a few limitations as to how many items can be stored and how often. These limitations will shape the way in which filters are used by my application and may result in using external API's as a main form of input rather than saving items directly. Another constraint that arises from what was mentioned in the system behaviour is call backs. Call backs are slightly outdated, and a better option is promises which is unfortunately not what the Chrome API uses. [14] Call back hell exists and means watching how and where call backs are being used will be important to maintain the quality and readability of my code. [15]

4.3 Selecting Elements

4.3.1 Users Perspective

Selecting elements is a key part of the adblocking experience for the user. Selecting elements allows for the user to manually click an item on the webpage they are viewing to render it blocked. Creating and intuitive and simple interface for this was tricky at first but has turned out well.

For the user to be able to select and element to be blocked they must first open the popup. Once the popup is opened, they must select the setting that toggles element selection on, from the second group of options. Once this option has been selected the user will now see their current web page modified as they move their mouse around. As the user moves their cursor over the web page a border will appear around their currently hovered element. As the user moves their cursor around the border is programmatically removed and applied to whichever element they are not hovering over.

When the user is ready to block an element, they may click to add the classes and, or ids to the list of elements to be blocked on that website. During this time, the user may also want to block an element that is a link or has functionality attached to it. This is not a problem as when the user is in this selection mode links are blocked and therefore will not redirect the user. To further this the cursor will be the normal pointer rather than the hand which indicates a link. Once the selection has been made and the web page will reload, and the selection mode will be taken off revealing the normal website once again. This script is only designed to allow for the selection and storing of elements and blocking will be done by another component.

4.3.2 System Behaviour

When a web page is loaded the script Select Element is automatically loaded in as stated in the extensions manifest. This script is injected before any of the DOM has begun to load. This script sits idle with a listener waiting for a message telling it to begin. This message will be

sent when the user clicks the option in the popup. Once this message has been received the script will be activated.

The script will listen to movements of the user's mouse and identify which element they are currently hovering over. Whenever the mouse moves this function will be recalled and the domain object recalculated. Every time a new object is found it will be highlighted in an obvious manner for the user so they can see which element the script has calculated they are hovering over. The user is now stuck in this mode until the web page is refreshed which can be achieved by manually refreshing the page or continuing and selecting an element which will automatically trigger the refresh.

4.3.3 Architecture

The element selection consists of JavaScript and jQuery spread across two files. One is the *SelectElement.js* file which houses all the code that allows the user to select items. The other file is the *popup.js* file which tells the current page when to go into selection mode. *SelectElement.js* also interacts with the Chrome API to store items. CSS is also dynamically injected into the webpages to stop links from being clicked when in selection mode.

4.3.4 Constraints

One constraint for the current design means that elements without a class or id will not have the ability to be blocked. Currently the plan is to store a reference to the class and or id that the selected element has. This involved going up the DOM until the hovered item has either one. Using this strategy, it means selections bigger than the user's initial intent may have to be blocked.

4.4 Blocking Elements

4.4.1 User Perspective

The user perspective for blocking elements should be simple. The process should be done seamlessly upon items they have previously selected. This system should shy away from any extra user interaction and instead this should be handled by the previous, element selection. There should be two related options available to the user. The first is the whitelist option which will allow the user to toggle the adblocker on and off for the current web page they are viewing. And the second denotes how elements are hidden on the page. This second option will allow the user to choose whether the space of the adverts is to be left as it was, but blank, or for it to be collapsed. These options keep the process simple but offer two powerful choices for the user to make.

4.4.2 System Behaviour

As the script runs on every page it will always have the most up to date version of the stored elements that need to be blocked. This script will begin to run before the web page has even began loading properly. During this time, it will prepare all the CSS that needs to be injected. This should be done by creating one CSS rule that is applied to all the stored elements. Once This is ready and the page has begun loading it should be added straight to the page allowing it to block all the elements. As this should be done right at the start it should mean the adverts

do not get seen at all. Everything to do with this script is automatic and does not need any interaction from the user. A condition will wrap this code firstly telling it whether to run, this will be based on whether the website is on the whitelist or not. Secondly another code snippet will check the settings to either place the visibility or display property on it correctly. An important tag can also be applied add extra presidency onto the generated CSS statement.

4.4.3 Architecture

The architecture to this script is very simple. As previously mentioned, it loads at the start of the page. It will generate its own custom CSS rule that includes all the items it loaded from the storage. The script will contain a listener that waits for the document to begin loading properly, and a function that will be ran when this happens. This script is the easier part of blocking items and does exactly as you would expect.

4.4.4 Constraints

A constraint for this system will be to do with how elements were previously stored. As mentioned in the storing of elements, the class and id will be stored of the selected element, or first parent with one of these attributes. This could mean that only a child of the advert is being blocked and that the parent frame will have its own height and width. Although this parent may be blank it will not be collapsible in line with the setting to do so. Another constraint related to other elements with the same name or class. With the current design the blocker will do as it is told and block the ids and classes it has been given. However, this does not mean some elements on the page will not have the same class or id and get blocked by accident.

4.5 Selecting Domains

4.5.1 User Perspective

Selecting domains will allow the user to manually block entire websites they do not like. The user will be able to either type or select domains that they wish to stop traffic from loading from. There will be three ways in which the user is able to do this. Selecting, typing, and loading. Each of these three ways allows for a different interaction between the user and extension but ultimately end with the same goal of blocking domains. Having multiple ways is important as it will allow for a greater convenience top the user.

The first of the three ways that the user will be able to interface with domain selecting will be through a manual selection. The user should be able to identify an advert themselves and add it to the block list much like they can do with elements. With domains it is much easier for the user to see what they are selecting as Chrome displays the currently hovered URL against the bottom of the window. The user should then have the option to select the item and mark it as blocked, adding it to the storage.

The second way in which the user will be able to interface with domain selecting will be through typing it out and having it added to storage. Inside of the popup there is a textbox and button combo that allows the user to type in a domain they do not wish to receive traffic from and block it. Domains to be filtered require a specific format with the Chrome API and

there for a placeholder and entry feedback system will be in place to help the user type an acceptable string.

The third and final way the user will be able to select domains is through an external source. An option will be available to the user allowing them to save and load in a list of filters from a website. This option again will be displayed to the user inside the popup menu. The user will only need to supply a URL for this to work. The plan is that the user can use lists already created and used by other ad-blockers. Once the user hits save and ensures the toggle is enabled the loaded domains will be blocked.

4.5.2 System Behaviour

After any changes are made to the domain's lists, either than be a new entry is added or loaded the page will refresh so that the domain blockers may take effect on the updated list. Upon the user selecting a domain to be blocked manually the script will automatically convert the URL that was collected into the right format for the Chrome API and add it into storage. When the user selects to add their own URL to the block list and types it in the format will be verified and then added once again to the block list. When loading from a web page, the user will first have to enable the option before they enter the URL and hit save. Behind the scenes seamlessly the data will be collected and stored inside of the local session, to be used until the browser is closed. When the browser is reopened the data will be redownloaded from the saved site.

4.5.3 Architecture

The architecture behind domain selecting is varied per each method previously stated. With the method of manually selecting elements the Chrome API can be very handy. Using the Chrome API will allow for a custom context menu to be created and used. Using a custom context menu will allow for an extra option to be added to the right click menu. Clicking this item can run code which will allow the domain the click came from to be blocked. This is a simple process and is majorly handled by the Chrome API.

When it comes to typing out domains, we are instead restricted by the Chrome API. It only accepts domain strings in a specific matching format that it uses. Therefore, the user will have to type it correctly and Regex can be used to ensure this. With little space it is important to make it as obvious as possible what format the domains need to be in for the user.

The final method will use a form of built in request sending that is shipped with JavaScript. This will allow us send and receive data from other web addressed. The user should enter a URL that loads a list of domains they would like to filter against. The request can be used to load the URL and retrieve the contents of the page. Once the data has been retrieved it can be formatted correctly and then applied as a filter to the incoming requests.

4.5.4 Constraints

A major constraint lies with this part of the project and that is using external filter lists. Many lists of predefined domains already exist but can be hard to implement. To begin with as lists grow the time it takes for them to be filtered and compared will increase making the whole adblocker and browsing experience slower. Another constraint that goes with these lists is

implementing them. They all come in their own format with schemes etc and it may not be possible to parse the lists correctly if the documentation of how to do so does not already exist. A final constraint on selection could be making sure the user is selecting the right one.

4.6 Blocking Domains

4.6.1 User Perspective

The user is not going to see much when it comes to domain blocking, only really that the domains they have selected will not appear. There will be one related option inside of the popup that will allow the user to make changes to blocking. The User will be able to see a list of the domains they have manually chosen to block. This means domains they have typed in or entered through the context menu. The table these are in will allow the user to release domains they no longer wish to be blocked.

4.6.2 System Behaviour

When Chrome loads so does the background script. This is where the domain filter API will be attached to incoming requests. These listeners will take in a list of URLs to check against and block ones that match. These listeners take in a copy of the list so any changes will mean the listeners need to be refreshed. As the user makes changes to the settings, loads more URLs or removes some the background will update its listeners with updated versions of the lists. The blocking can be done in multiple ways. The blocking can either filter out entire domains or filter out domains that match certain character strings etc. This is something I will consider when I begin implementing.

4.6.3 Architecture

The architecture behind this part of the extension is primarily going to be focussed on the Chrome API. The Chrome API has a whole toolkit that allows you to work with incoming and outgoing web requests. These tools will be used to allow the filtering of domains being let in. the blocking will be done in the background script which is persistent throughout the duration of your session on Chrome.

4.6.4 Constraints

Blocking domains can come with a few constraints. To begin with there is an issue where the popup itself could become filtered, therefore rendering the whole extension useless until re-installed. As blocking domains can use pattern matching small parts of the URL could match and therefore make it filtered. Another constraint is just the overall war between ad-blockers and advert makers. Blocking domains on a basic level is not so tough but advert makers are making it harder to do so. There are many ways in which adverts are bypassing domain blocks and it will be hard to work against and combat them all. There is a lot of unknown and only working on this project will help reveal the effectiveness.

5 Implementation

5.1 Popup

5.1.1 Interface

The interface that has been created for the popup has been done so using traditional HTML and CSS with the bootstrap CSS style sheet also applied. The popup follows a very simplistic design and does not overly complicate what it is trying to display. [Fig. 52] The popup makes use of the following files which are all referenced for use in the head of the html file itself:

- *Interface/popup.html*
- *Interface/bootstrap.min.css*
- *Interface/popup.css*
- *Interface/popup.js*
- *Jquery.js*

The HTML document has been divided into easy to identify sections using the <section> tag. [Fig. 16] The section tag is part of HTML5 and is a block style element. Being a block type element means the sections will stack up nicely in the window. Each of the sections has a unique class identifier allowing styling to be applied directly to children of a given class. Between each of the four sections a <hr/> tag has been used to create a boundary. [Fig. 14] The first section simply used two tags to display information. [Fig. 17]

Between the following two sections unique ids are applied to all interactable elements so that additional processing may be done by the *popup.js* script later. The HTML makes use of <input> with the attribute type and values checkbox, radio, and text. <Button> is also used to allow a straightforward way to capture user clicks. [Fig. 18] The final section contains two <div> elements with the class card. [Fig. 19] This class comes from the bootstrap file and creates a nice-looking area. Inside each of the cards lies some text and the outline of a <table> element that will later be filled. [Fig. 15]

As mentioned previously the design also uses its own style sheet that has primarily been used to create a better layout for the elements. Padding has been applied to the body to keep all elements away from the edge and a Margin applied to all the buttons to keep them from touching each other vertically. Making good use of the correct elements, being block or inline has saved the need for extra formatting through CSS.

5.1.2 Data population

Data population of the popup and HTML interface is all handled by the *popup.js* file. This file as previously mentioned is referenced by the HTML file and therefore has full access to its DOM. When the user clicks onto the extension and the popup opens the options will all be set with their default values. The *popup.js* file will then go through and set the correct values after retrieving them from the Chrome storage API. Code that has been written in this file varies between vanilla JavaScript and jQuery. A mixture of both has been used as while jQuery provides some easy to use functions, I am not too familiar with all aspects.

To begin with everything inside the *popup.js* file is wrapped around a document ready event, with call back. [Fig. 21] This means the *popup.js* will sit and wait until the document, which is the HTML page has fully loaded before executing any of its code. This has been done so that values will only be set when the elements exist. There are five elements that will be set initially via the *popup.js* file.

The five items to be set are: Whether the blocker should run on this page, What type of blocking should be done(collapse / hide), Whether to use an external URL filter, What the external URL filter is and finally whether to use the default block list. These five parts are all wrapped inside the call back of a call to the Chrome storage API requesting the information needed. [Fig. 20] Four of these options are toggle style options and use IF statements to switch their state depending on the retrieved value. IF statements have been used as they are clean and simple to use although ternary operators could be used to save lines. Using JavaScript DOM selectors on the ID of each element is then used to set its values. [Fig. 22] All of this should happen without the user ever seeing the initial default values.

5.1.3 Events

Events are used inside of the popup to run code when something happens. The events that have been used inside of *popup.js* are click and change. [Fig. 23] Using these events allows us to have code executed when the user performs an action. Each of the options that are located inside of the options menu has a unique ID that can be used within an events listener. Event listeners can be applied directly to an element so an event like a click can be captured from just a specific element. The listeners execute a call back function when the event happens.

The contents of each event function differ greatly depending on what element called it. Some of the events will post a message through the Chrome extension API to the background script, telling it to refresh a piece of data (this needed as it's a persistent script), while others will check what the current stored value is and make a change to it. [Fig. 24] Example of this are when a setting is changed from true to false or when an item is being removed from the filter list. Events are the backbone of all user interaction on this page.

5.1.4 Tables

The tables at the bottom of the user interface are loaded in as empty tables to begin with. Once the document has loaded code will be executed to populate them. To begin with data is collected from storage. Once this data is retrieved the call back function starts. Inside of the call back function, button elements are created and stored in JavaScript. The name of the item is added as the text of the button and an on click event is added that removes the item from storage and the table when clicked. Once the button is made, a <td> element is made, and the button added. Following this a <tr> element made and the <td> added to this. Now the id of the table is located and added as a child. [Fig. 25]

5.2 Storing data

5.2.1 Data Structures

Data storage is used all over this extension. Data is constantly being stored and retrieved from different script and it is important it is stored in a manner that is easy to work with. The data structure that Chrome works with and is used in the Chrome storage API is JSON. Throughout the development of this project the storage structure was changed many times and now uses a mixture of arrays and objects. The data structure is first created when the application is installed. [Fig. 26] This means that the data will always be accessible when called and will not need generating on each first call.

The overall object that stored data contains three objects itself. The first object named Websites stores all the classes and id's the user has previously selected to block. The data structure inside of this object is KvP where the key is the domain to store data for and the value another object. This object then stores two KvP. In this the key is either ids or classes and the value both arrays of the data. The structure looks like this {"Google.com" : {ids : [], classes : []}, "reddit.com" : {ids : [], classes : []}}; This structure allows for data to be classified as its stored. It makes retrieval of specific items much easier. [Fig. 27]

The next object is named Settings. This object contains five KvP. Two are strings, two are Booleans and the final are an array. [Fig. 26] Keeping the settings together in this way means the settings can be retrieved without pulling tonnes of extra data in the form of user selected items for blocking. The final object named Blocked is another KvP that stores an array. The array is used to store user selected domains. The array stores these domains as strings. Arrays are ideal and allow for lots of built in functionality for searching and sorting.

5.2.2 Setting Data

Default values for data are initially set when the program is first installed. This all takes place in the *background.js* file. [Fig. 26] As previously mentioned, this makes setting data later easier as the basic data structure will already be in place. Data is set in many places of the extension and done so in different ways. The Chrome storage API is very powerful and allows for quick and easy data manipulation in the storage. To begin with when setting data, we must ensure we are modifying the data that already exists. When we can `Chrome.Storage.sync.set(variable)` variable will overwrite everything that is already being stored.

Inside the extension whenever we want to make modifications to data, we first call the `Chrome.storage.sync.get(data, call back)` method. This will return us all the data that is currently being stored. Now we have this data we can make modifications. Some parts of the application will index to a specific item inside the data structure and change its value whereas others will use the delete keyword to remove whole sets of data from within. For other bits of data like arrays, Push and pop and splice have been used to manipulate the array with new values etc. [Fig. 28] once the data has been manipulated a call is once again made to the storage API to set the data. An optional call back is used throughout the application to communicate these changes to other parts of the extension.

5.3 Selecting Elements

5.3.1 Highlighting elements

The process of highlighting and selecting an element was not too tricky. This mode can be activated by the user clicking on the Select an element option from the menu. A message is then sent through the come API to the script which has a listener waiting. Once the message is received code will begin executing. [Fig. 29][Fig. 30] Firstly, a custom style is appended to the head of the document that stops links from being clicked. [Fig. 31] This allows the user to click any element on the page without worrying about a redirect.

A mousemove event is attached through jQuery that will listen for and execute every time the user's cursor moves. Firstly, the current style is removed. This just takes off any inline styles the element may have. This is done to remove the border from the element if it was the previous selected one. A variable is set to the element the mouse is currently hovering over. At this point since we are blocking elements by either a class or id we loop continuously calling the parent of the element until we reach an element either a class or id or ready the body, which is the top level element. This loop ensures we have an element with either a class or ID. At this point an inline style is applied directly to the element that the user's mouse is over. [Fig. 32]

Another event is also attached when the selection mode begins. This event is placed on the document and listens for a click. Since this click is on the document level it will be caught wherever it is on the page. When the click happens both itself and the mouseover event will be unbound to ensure the click is captured correctly and another event cannot fire. Next the red border is removed the data sent to a function to be formatted and stored before reloading the page. [Fig. 33] Overall, the process of highlighting an element was not too difficult and works nicely.

5.3.2 Preparing elements for storage

Preparing elements for storage took a long time to figure out. Initial attempts involved trying to store entire elements in storage and eventually worked somewhat. After going through the pain of making this work, extra research was done into how the other ad-blockers did this and discovered they simply blocked via ID and classes. With this new idea in mind a quick re-work to the existing code was made and resulted in this much cleaner solution. [Fig. 34]

Whenever the selection is made by the user of the element they would like to block a function called StoreElements() is run. This function will firstly create two variables and attempt to store the value from the class and id in them, respectively. If the element they are taking from does not have one of them then the variable is set to "". [Fig. 35] One problem that occurred was illegal characters in CSS names. Initially it was believed that the code worked fine until there was an element unable to be blocked. After investigating with other ad-blockers it became apparent they had escaped certain characters. This led to the discovery of CSS.escape(). [Fig. 36] This library that is built into JavaScript allows you to parse CSS strings easily and automatically without having to worry about illegal characters. [16]

Next a call will be made to the storage API asking for the currently stored data. Once this has been retrieved and the call back execute the new ids and classes are appended into the array of currently stored items. Now the set API will be called and add the modified gotten value back into storage.

5.4 Blocking Elements

5.4.1 CSS injection

The implementation of blocking elements in this extension is quite simple. All the logic for this resides in the file named *BlockElement.js* [Fig. 37] This file is loaded before the web page begins. The script first gets the whitelist from within Settings > Whitelist in storage and checks to see whether the domain is contained in the array. If the domain is not stored in the whitelist, then a listener is stored that checks for changes to the web page. This listener is set to `DOMSubtreeModified`. [Fig. 38] This event is fired anytime that something is changed on the web page. Because this script is being loaded before the web page, we will get an event for every small change as the page begins to load.

The function `injectCSS()` is ran whenever the event is fired. This function firstly retrieved the stored arrays for classes and ids of elements to be blocked related to the domain of the current page. Once these have been retrieved a style element is created in memory. At this point a string variable that will hold the blocking statement begins to be formed. Each item that was retrieved from the storage is added to the string with a comma and space after it. In CSS terms this will mean for each of these items. Once all the items have been added another check is made against the stored settings to see whether the user has opted for collapsing or hiding of elements. The correct CSS statement is then added at the end of the string. [Fig. 39]

Now we have our CSS statement stored we can place it into the web page. Wrapped around all this logic is a check to see which element changed on the page when the `DOMSubtreeModified` event was fired. If the change was the head of the document loading, then all this logic was executed. At this point the script knows the head has been loaded into the web page. With this knowledge the script will now append the style into the header blocking all the elements in the string in the manner the user selected. The string to collapse the elements is `display: none`, and the string to hide the elements is `visibility: hidden`. The string `!important` is placed right after the previous statement to add extra precedence to it. Once this tag has been injected into the head the scripts job is done and the listener detached to stop this code from executing again.

5.5 Selecting Domains

5.5.1 Context menu

The selection of domains to block needed to be as simple and convenient and possible for the user therefore the Context menu seemed like a perfect option. The context menu is part of the Chrome API and therefore requires little work to implement. [17] Located in the *background.js* script the context menu is added programmatically whenever the browser is opened. The API location is `Chrome.contextMenus.create({title: , contexts: , onclick: })`. An

object is taken in as its only argument. The object contains a title, context and onclick function. The title is “Block this domain” and is what is displayed inside the context menu. The contexts are an array of strings that denotes when this option should be visible in the context menu. For this implementation, the string “Links” so the option is only displayed when right clicking links (to note most adverts have a surrounding <a> tag). [Fig. 40]

The onclick function is used to run code whenever the option is used from the context menu. Firstly, the data from the context menu is processed, converting the URL into a domain, and formatting it into the Chrome pattern matching format. Once this has been done a get request is sent to storage and a check is done to make sure the value does not already exist in storage. Assuming it does not the program will then push it into the stored array and set the value back to storage. Once it’s stored the call back will trigger and `ResetManualDomainListeners()` [Fig. 41] will be called before the page reloads with `ReloadPage()`.

Once `ResetManualDomainListeners()` has been called the program will once again make a call to storage and get the Domain List to ensure it’s up to date. The block list will then be made by optionally adding in the default blocklist if that option is enabled. To finish this process the original `WebRequest` `onBeforeRequest` listener will be released and a new one attached with the new array of domains. [Fig. 41] This is important as when the listener is attached it takes a copy of the array not a reference, therefore the only way to update the array being used is to reattach the listener.

5.5.2 XMLHttpRequest

The user has been given the option to import a filter from an external address. This will save them time by not having to find or enter each one manually. Initially the idea was to implement something like EasyList. After spending time on this and not managing to get it working, instead it was opted to just load in any text file. EasyList contains a set of domains and search terms that use special characters to denote how they should be interpreted. For now, the current implementation just allows for any URL to be loaded in. It is intended that this URL will contain simple text stored line by line. A good example of this the raw view of a text file on pastebin. The current implementation uses the include filter to check whether the URL contains at all any items from the filter. I.e. `Reddit.com/ads` contains `.com/ads`, `Re`, etc...

The method `ResetWebBlock` [Fig. 42] handles the creation and blocking of domains loaded from an external URL. The method is called once when the browser launcher and then again whenever the option is toggled on or off or the URL to load from is changed. Firstly, a call is made to retrieve the settings, the settings are checked to know whether to use the filter. If the settings are true, then an object of type `XMLHttpRequest` is created and stored under the variable `req`. [Fig. 43] A call is then made on `req.open()` that named a get request to the URL we had retrieved from storage. A function is then applied on `req.onload()` that denotes what is to be returned. We tell the request that an array should be returned and use `req.responseText` to get all of the text from the page. The `split` function is then used on the string at `\r\n`. Splitting it on these characters will split the string whenever a new line is found. This will store straight into the array with each line being a new array item.

5.6 Blocking Domains

5.6.1 Blocking domains from external list

Following on from Domain Selecting Domain Blocking from an external list is also achieved through the method `ResetWebBlock`. [Fig. 42] Once the previous `XMLHttpRequest` has finished and the filter array has been stored a call is first made to remove any previous listeners. This call is made because this function is also called later to reset the listeners when a new URL is chosen, or the toggle of this setting is hit.

A call is next made to `Chrome.webRequest.onBeforeRequest`. [Fig. 44] The listener runs before any request is made by the web page. This means right before every request is about to be sent out our call back function will run. To begin we need to supply three items. A call back function, an array of URLs to apply this to and finally the context. For this case context will be ["Blocking"] as that is what we intent to do with these requests. Our array of URLs will simply be ["<all_urls>"] as we would like to run this filter on all traffic. Lastly, our call back function is where we are going to check the URL against our filter list. The call back function named `WebDomainBlock()` takes in the URL that was passed and then checks over all items in the filter list to see if they're located inside the URL. If one is found, then an object of {cancel: true} is returned signalling for this request to be blocked. The using `Array.find(e => url.includes(e) !== undefined)` will let us know whether any of the filters were matched within the URL. [Fig. 45]

The function `ResetWebBlock()` also contains an else if check to stop any loading or blocking from the external list if the settings are not enabled. This function is also called from the *popup.js* at two different locations. If either the setting to enable is changed or the URL itself is changed a message is sent via the Chrome API to the background script where a listener will intercept it and run the `ResetWebBlock()` method again.

5.6.2 Blocking domains from manual input

The blocking of manually selected domains and domains that are pre-loaded with the default domains list take place inside of the `ResetManualDomainListeners()` [Fig. 46] function. As mentioned before here the list of domains is created by loading from storage and concatenating with the default domain list if set to do so. Just like the function from web domain blocking we first detach any listener as this function is used to refresh the list later. `Chrome.webRequest.onBeforeRequest` [Fig. 48] is used again to attach another function when a call is about to go out from the page.

The big difference this time is instead of applying this function to every domain we are just going to apply it to the domains we have already stored. So, in this instance the context is still ["blocking"] but the array of filters is instead set to the variable `manualFilterList`. The method we use to execute the check is called `ManualDomainBlock`. [Fig. 47] This method will just evaluate whether or not the filter list has values in it and then return the object {cancel: true/false} accordingly. This evaluation is done as if the domain list is empty every request from any domain will be blocked including loading the actual page itself.

ResetManualDomainBlock() is also called from the *popup.js* file. Like previous a message is sent through the Chrome API to the background page where this function is located telling the function to run again (a value has changed, and the listeners need an update). The function is called whenever the user: Changes their selection of whether or not to use the default list, they add an additional domain through the context menu, add a domain by typing it out or whenever the user removes a domain from the list through the table interface in the *popup.html* window.

6 Results and Evaluation

This project was carried out and completed using the agile development methodology. Using this method means that each individual component was tested as working before the next component was ever worked on. This type of development meant that functionality testing was always happening. As previously mentioned, work was divided up and assigned to sprints using Trello, an online task board management system. Trello allowed me to keep track of my progress and prepare for future sprints. When each component was completed it was tested, as well as testing the previously added components to ensure that no changes disturbed the already working parts.

The manual functionality testing that was carried out on the application as each component added was great and relatively easy to do on this smaller-scale project. However, one way in which this whole process could have been improved would be using unit tests. Unit tests can be used to test a program programmatically. These tests can either be run manually by the user or run when code is pushed to the version management server. In previous projects I have used tests and believe it would be a good addition to the project. However, working with JavaScript meant that I was not so familiar with the route of action to achieve this. Furthering this entire point converting the entire development method to Test Driven Development would have allowed for an even more concise and accurate development cycle.

In the continuation of this section I will begin to look at how well my application compares to other similar products. I will also be looking at how my overall outcome has changed from my initial ideas and explore the main reasons behind this. Finally, I will look at how the original aims of this project held out and test to see how well they were met in the finished application.

6.1 The realisation of how ad-blockers work and what that means for this project

Throughout the development of this project I actively explored and developed using information I had, and continually gathered from existing ad-blocking extensions. Many of my initial attempts and approaches to blocking ads fell through after long coding sessions that ultimately led to lacklustre results. As the development of my project grew and I began to implement some of the later features such as domain blocking. It was at this stage of the project that the real challenge and deeper research came into play and shed light on how ad-blockers really work.

The previous statement is slightly exaggerated but conveys the point greatly. Of course, I had previously researched and knew a filter list of some kind was used, I knew of a few and I knew how to at least load them into my project. Ad-blockers, to be blunt are just systems that can stop adverts from being displayed on a website. This can be done in a variety of different ways but ultimately that is all they can do. Ad-blockers do not automatically detect, they do not guess, they are told precisely what to block.

Implementing an already existing blocklist became the biggest challenge for me and ultimately did not pull through. The most used list is EasyList. EasyList which has previously

been mentioned in this report uses a unique schema in its list that allows for precise and complex pattern matching to take place. After spending a good amount of time reading through the documentation and making many attempts to convert their schema into RegEx, I had to make the decision to not implement the block list. However, I do not believe this counts as a defeat. The adblocker that has been created still blocks adverts, by domain, by element just like other ad-blockers. Without their third-party domain list, they would be in the same situation as me.

I still believe at the end of this project I have been able to make an ad-blocker that is equipped with a handful of useful options. While I have not implemented the block list, I have however made a short but precise list of powerful options available to the user. I believe my success lies with the more concise and prompter list of options that are available all through the extension window. When previously reviewing other ad-blockers one of the most apparent things was that the options were too few, too many and always overly complicated. Furthering these points, the ad-blockers I tested did not even perform greatly when tasked with removing adverts from the webpage. Therefore, I believe that giving the user the power to load their own blocklist will allow for a more powerful blocking experience.

Lastly to note the project does ship with a file named *Blocked.js*. This file is intended to contain a default blocklist that would have already been curated for the user. This file however has not been curated and instead with the finalisation of the project contains a few test URLs that can be used to prove the functionality of this feature. [Fig. 49]

6.2 Comparison against other similar software

6.2.1 Blocking comparison

As previously just mentioned the project I have created does not directly ship with a full ad-blocking list. Therefore, I believe it would not be right to make a proper comparison about how many ads are blocked. Really a blocking comparison would not make sense as each ad-blocker's score would be based upon their list and not the actual extension.

One way in which we could measure the effectiveness of the ad-blockers would be to measure the speeds in a website takes to load with the adblocker turned on. Again, this test would potentially be affected by the lists behind them. This however could work by disabling all pre-entered lists and adding a baseline of rules. I will not be conducting an evaluation on the speeds as after using all three ad-blockers and the one that has been created it is easy to see visually there is not a difference. There is not an extra hang time, or any noticeable lag created by any of the ad-blockers. It is hard to make a real comparison against each of the competitors. Instead of speeds it is proposed by this project that functionality and ease of use is instead the better measurement. In the next section I will look at how ad-blockers compare in terms of clicks needed to perform an action from the options available.

6.2.2 UX & UI comparison

With adblocker it is important that the user can quickly and efficiently navigate through the options menu to whatever they are looking for. This for example could be the user trying to

disable a setting or even block an ad they have spotted. I will therefore carry out and showcase the results of tests that relate to these UX features.

Pre-Conditions:					
<ul style="list-style-type: none"> • Extension has just been installed. • Restart the browser. • Navigate to “Stackoverflow.com”. 					
Ad-blockers	Clicks to do:				
	Block element	Unblock an element	Whitelist website	Block domain	Add/Change filter list
<i>Project</i>	3	2	2	2	4
<i>AdblockPlus</i>	4	5	3	5+ ?	5
<i>Adblock-Best</i>	5 really 3	3 really 2	2	6	5
<i>uBlocker</i>	5 really 3	3 really 2	2	N/A	N/A

With the application I have created all the options are as close and convenient to you as they could be. To begin with for the user to block an element they must click the popup, click the element blocking option and then simply click the element on the page. For the user to unblock an element its even easier and just requires a click to open the popup and a click on the item they wish to bring back, noting a downward scroll could be required. Next the ability to whitelist a website once again involves a click on the popup and a click on the checkbox that disables it for the current page. Blocking a domain does not use the popup and instead requires a right click and then a left click on the opened context menu, simple. Lastly is changing the external filter list. This takes slightly longer as it requires the user to open the popup, enable the option, click the textbox to begin entry and then finalising their entry with the save button. [Mean – 2.6, Median - 2, Mode - 2].

AdBlock Plus hosts a few useful options in the popup window and then a mixture of other options in the settings panel. Firstly, when blocking an element, you have 3 initial clicks, open the popup, block and element and then selecting the element before clicking the refresh button manually. To unblock an element, you must open the popup, click the settings button then navigate to the advanced page and remove it from the table. This is only a few extra clicks, but it is hidden away quite well. Similarly, to blocking an element the whitelist is an initial 2 clicks, popup and the option followed by a refresh. Blocking a domain was not obvious to me and I was not actually able to do so. I managed to navigate to the page I believed it would be in 5 clicks but ultimately did not manage to accomplish it. Finally, to change a filter it was five clicks again, the option hidden away in the back settings. [Mean – 4.4, Median - 5, Mode - 5]

AdBlock – Best AdBlock has the most options available to it. One major thing to note with this addon is the amount of verification. I have recorded clicks with two numbers, the first is how many it took, and the second is without pressing, yes, confirm etc. Firstly, to block an add took 5/3, the option can be done from the popup or by right clicking, there are then two verification menus to confirm which item you selected. [Fig. 50][Fig. 51] Next for unblocking it is 3/2, simply open the popup and then press the option, again one verification clicks.

Whitelisting is a low 2, just open the popup and select not to run on this page. Blocking a domain is hard to find in the options menu in the background and boasts a larger 6 clicks. Some useful information is provided to help make the entry in their required format. Finally, to add an external link is 5 clicks once again located in the back. [Mean – 4.2, Median - 5, Mode - 5]

uBlocker is the last adblocker that was tested. This addon is very similar to Adblock Plus and takes the same number of clicks for all three of the first tests. However, this addon's settings page is much smaller and does not offer an option to either block a specific domain or change the filter list. This will not be scored as its results would skew the others.

After having time to use and evaluate each of the ad-blockers mentioned above I believe the product I have created stands up well in terms of having key options available and the ease of use. Other ad-blockers seem to have confusing options that are jumbled up in the settings window. Their options are often overwhelming and confusing. The application I have developed is concise and straight to the point. I believe some extra settings could be added and some of the existing ones fine-tuned for a better experience. With the right implementation of the EasyList added this adblocker could be just as good as some of the top ones available on the Chrome store.

6.3 Evaluation against requirements and Test Cases

When I began planning this project there were four main major requirements that were set to ensure the worthiness of the application. The four requirements would allow the user to: Selected and block elements on a web page, Selected domains to be blocked, Load domains to be blocked and for there to be an easy to access and use settings menu available to the user. I will now detail some test cases that can be used to show how well, if it all, these aims have been met by my finished project. The aims will be broken down further to show each smaller part works correctly. The tests will be correct as of May 2020 and cannot guarantee their results as permanent.

All the acceptance criteria for each aim was passed except for the criteria "The settings have a good UX and will be understandable to users." At this point it is easy to see some work can be done to increase the clarity of the popup to help users better understand how each option works and what its role is. Find below the Test cases.

6.3.1 Test Cases for First Aim

Test #: 1	Aim: The extension allows for the manual selection, storage and blocking of elements from any website the user wishes to block elements.
	Acceptance Criteria(s): It is obvious to the user which element they are selecting.
Pre-Conditions: <ul style="list-style-type: none"> • Extension has just been installed. • Restart the browser. • Navigate to a website "stackoverflow.com". 	
Steps: <ul style="list-style-type: none"> • Open the popup for the extension (click the icon for it) 	

<ul style="list-style-type: none"> • Press the button labelled “Block an Element” • Hover the mouse over the stack overflow logo • Hover the mouse over the search bar 		
Expected: Both elements should have a red border around them when hovered over respectively.	Actual: A red dashed border is displayed around the element.	PASS

Test #: 2	Aim: The extension allows for the manual selection, storage and blocking of elements from any website the user wishes to block elements.	
	Acceptance Criteria(s): A reference to any item the user selects are stored (across sessions). AND When a website loads, stored references are checked and applied to the loaded website to hide elements.	
Pre-Conditions: <ul style="list-style-type: none">• Extension has just been installed.• Restart the browser.• Navigate to a website “stackoverflow.com”.		
Steps: <ul style="list-style-type: none">• Open the popup for the extension (click the icon for it)• Press the button labelled “Block an Element”• Hover the mouse over the stack overflow logo• Click the stack overflow logo• Close the browser fully• Navigate to “stackoverflow.com”		
Expected: The stackoverflow logo should be missing from the web page	Actual: The logos web page is no longer displayed	PASS

Test #: 3	Aim: The extension allows for the manual selection, storage and blocking of elements from any website the user wishes to block elements.
	Acceptance Criteria(s): The user can edit the list of blocked elements they have selected.
Pre-Conditions: <ul style="list-style-type: none">• Extension has just been installed.• Restart the browser.• Navigate to a website “stackoverflow.com”.	
Steps: <ul style="list-style-type: none">• Open the popup for the extension (click the icon for it)• Press the button labelled “Block an Element”• Hover the mouse over the stack overflow logo• Click the stack overflow logo• Open the popup for the extension (click the icon for it)• Click all buttons displayed in the “Manually blocked elements” table.	

Expected: The buttons should disappear and the page load to display the now un-stored element	Actual: The page refreshes as expected, and the elements are no longer stored.	PASS
--	---	-------------

6.3.2 Test Cases for Second Aim

Test #: 4	Aim: Adverts can be blocked via domain filtering selections made by the user.	
	Acceptance Criteria(s): The user can select domains on an already loaded page to be blocked upon reload.	
Pre-Conditions: <ul style="list-style-type: none">• Extension has just been installed.• Restart the browser.• Navigate to a website “stackoverflow.com”.		
Steps: <ul style="list-style-type: none">• Scroll to the bottom of the page• Right click on the text “Facebook”, observer the url is “facebook.com” (Shown in the bottom left of the browser in a small alt text”• Select “Block this domain” on the context menu.• When the page reloads -> scroll to the bottom again• Click facebook		
Expected: The page should try and load but instead display a blocked message	Actual: a screen loads notifying you that facebook is blocked.	PASS

Test #: 5	Aim: Adverts can be blocked via domain filtering selections made by the user.	
	Acceptance Criteria(s): The user can manually type in domains which they wish to be blocked.	
Pre-Conditions: <ul style="list-style-type: none">• Extension has just been installed.• Restart the browser.• Navigate to a website “stackoverflow.com”.		
Steps: <ul style="list-style-type: none">• Open the popup for the extension (click the icon for it)• Inside the textbox with a placeholder text type “*://*.Google.com/*”• Press the button to the left “Block Domain”• Navigate to the web page “Google.com”		
Expected: The page should try and load but instead display a blocked message	Actual: a screen loads notifying you that Google is blocked.	PASS

Test #: 6	Aim: Adverts can be blocked via domain filtering selections made by the user.		
	Acceptance Criteria(s): The user can edit the list of blocked elements they have selected.		
Pre-Conditions:			
<ul style="list-style-type: none">Extension has just been installed.Restart the browser.Navigate to a website “stackoverflow.com”.Test #5 has just been completed			
Steps:			
<ul style="list-style-type: none">Open the popup for the extension (click the icon for it)Scroll to the bottom of the popupClick all buttons displayed in the “Manually blocked domains” table.Refresh the url in the browser			
Expected: Google.com should display and load correctly.		Actual: Google loads.	PASS

6.3.3 Test Cases for Third Aim

Test #: 7	Aim: Adverts can be blocked via domain filtering lists that the user can load or present to the extension themselves.		
	Acceptance Criteria(s): The user will be able to specify some sort of path to an already created filter list that they wish to use in their blocker.		
Pre-Conditions: <ul style="list-style-type: none">Extension has just been installed.Restart the browser.			
Steps: <ul style="list-style-type: none">Open the popup for the extension (click the icon for it)Click the button labelled "Enabled Ext"Enter the text into textbox to the right "https://pastebin.com/raw/8JHHkDZa"Hit the save buttonNavigate to "stackoverflow.com"			
Expected: The page should try and load but instead display a blocked message		Actual: the page does not load and displays the block message	PASS

Test #: 8	Aim: Adverts can be blocked via domain filtering lists that the user can load or present to the extension themselves.
	Acceptance Criteria(s): The user can choose whether to enable this feature at any time.
Pre-Conditions: <ul style="list-style-type: none">• Extension has just been installed.• Restart the browser.• Test #7 has just been completed	

Steps: <ul style="list-style-type: none"> • Open the popup for the extension (click the icon for it) • Click the button labelled "Disable Ext" • Reload the web page 		
Expected: stackoverflow should load correctly.	Actual: stackoverflow does load correctly.	PASS

Test #: 9	Aim: Adverts can be blocked via domain filtering lists that the user can load or present to the extension themselves.	
	Acceptance Criteria(s): The provided list works without conflict towards any other filtering.	
Pre-Conditions: <ul style="list-style-type: none">• Extension has just been installed.• Restart the browser.		
Steps: <ul style="list-style-type: none">• Open the popup for the extension (click the icon for it)• Click the button labelled "Enabled Ext"• Enter the text into textbox to the right "https://pastebin.com/raw/8JHHkDZa"• Hit the save button• Enter in the textbox with the placeholder text "*//*.Google.com/*"• Hit button labelled "Block Domain"• Navigate to Google.com -> observe• Navigate to stackoverflow.com -> observe		
Expected: Both websites should fail to load and display the block message	Actual: both pages fail to load	PASS

6.3.4 Test Cases for Fourth Aim

Test #: 10	Aim: The user can easily and without confusion edit the settings of their application.		
	Acceptance Criteria(s): All settings are loaded into the extension’s popup window, not in additional html files or background pages.		
Pre-Conditions: <ul style="list-style-type: none">• Extension has just been installed.• Restart the browser.			
Steps: <ul style="list-style-type: none">• Open the popup for the extension (click the icon for it)• View settings -> observe			
Expected: All settings should be shown. No links to other pages etc	Actual: All settings are indeed in the popup window.	PASS	

Test #: 11	Aim: The user can easily and without confusion edit the settings of their application.		
	Acceptance Criteria(s): The settings have a good UX and will be understandable to users.		
Pre-Conditions: <ul style="list-style-type: none">Extension has just been installed.Restart the browser.			
Steps: <ul style="list-style-type: none">Open the popup for the extension (click the icon for it)View settings -> observe			
Expected: Settings should be clear and easy to use		Actual: The settings are slightly confusing and could use additional information to help understand them	FAIL

Test #: 12	Aim: The user can easily and without confusion edit the settings of their application.		
	Acceptance Criteria(s): All settings are saved seamlessly upon change		
Pre-Conditions: <ul style="list-style-type: none">Extension has just been installed.Restart the browser.			
Steps: <ul style="list-style-type: none">Open the popup for the extension (click the icon for it)Click the radio button labelled "Collapse Items"Click the checkbox labelled "Run on this page"Click the Button labelled "Enable Ext"Close the popup window by clicking the pageOpen the popup for the extension (click the icon for it)			
Expected: The following settings should be visible: Run on this page: unticked Collapse Items: checked Button Disable Ext : shown		Actual: These settings are shown and the textbox and save button next to disable ext are also active.	PASS

7 Future Work

As the project has progressed code has become messy, comments lost, and bits of code duplicated. Some features did not make it in because of time restrictions and difficulty to implement them so this section will look at some of the things that were missed out on and would benefit being added. The program currently works well and does not have any really pressing issues. As my knowledge of JavaScript and the Chrome API as the project has progressed earlier bits of code still lie around and could easily be improved.

7.1 Full validation

Validation is something that was only touched on slightly during the project. Currently there is only validation when a user enters a domain string to be added to the domain block list. This validation works well and gives decent feedback to the user in the form of a green or red outline to explain the result of their entry. There is also some validation behind the scenes to stop duplicate items being added to the blocklists but not under all circumstances. Validation like this would be useful in all scenarios and help the user understand how their actions are impacting the extension. Luckily, this application does not have too many points of entry for the user so there are not tonnes of things that need validating.

7.1.1 Domain Loading

The option to load domains straight from a URL is very important and useful in the extension. However, this feature can be quite dangerous and easily break the application if not used correctly. It would be a good start and important point to add perhaps a limit to which sites the URLs can be from. Perhaps limit it to only pastebin files that are raw (so it is just plain text) or as some other ad-blockers do, provide a drop down of premade lists. If the option to allow any sort of URL was kept then ensuring that all the contents of the file are acceptable would be a good form of validation. This would stop errors happening during parsing.

7.1.2 Manual Domain Typing

Currently the user must type in a domain following the schema that is displayed in the placeholder text. This is then validated once entered. It would be better if instead the user could enter any domain and that be validated and then converted into the correct format. Although the format is obvious as it is shown, it would be easier and be less error prone if the user just needed to type a normal domain. The current validation could also be updated to reflect the acceptability of the string on each keypress.

7.2 Error handling

Currently the extension does not have any error handling in it. There have not been any ground-breaking or application stopping bugs yet. As we are primarily storing items and retrieving them any data that breaks the application while running could probably be solved by a browser restart.

7.2.1 Error logging

Error logging could add to the application to allow the user to see what errors they are having. These errors could be displayed in a custom debugger window and will allow for advanced users to fix problems themselves. Error logging could also be used to send important information about problems straight to the developer if the user has opted in for their data to be shared. Having error logs can benefit everyone. Currently the application does not record or alert the user of any problems and this can be confusing and could be frustrating if it breaks entirely.

7.2.2 Try Catch

Currently most of the code runs as normal in the flow it has been created for. It is assumed that the data going in will be right but not every entry point or condition has been thought out. Try catch allows for code to be executed without breaking a program. The try statement allows for code to be ran and if nothing is wrong it continues as normal. The catch can be used to alert an error or fix an issue if the try has failed. JavaScript does not rely on user typed data types and therefore works well to cast and convert datatypes where it can but can sometimes fail, this is a scenario where try and catch is useful. Another place could be when connecting to a supplied URL if it is inaccessible, to try again or alert the user.

7.3 Code optimisations

As this project progressed old code and new code have collided and been made to work. Existing structures that were put into place have remained but could be done better. Some components could do with a complete rewrite and may be better off with one. Code however has been refactored where possible and therefore is efficient still. It is believed that further optimisation would help increase the effectiveness of this adblocker. As many parts of this are dealing with requests and run before pages are loaded every millisecond of improvement will help make the extension better.

7.3.1 Core components

As stressed previously this project has been a giant learning curve and many things have been built upon the initial code. Currently many parts of the code are being duplicated and are jumbled up. Much like an object-oriented language this project could benefit from having its core and reusable components stripped out and added to their own JavaScript files. JavaScript is very robust and allows for code to be split up very easily. Doing this would help make code more readable and the overall structure of files in the extension more concise. Having less duplicated code would also mean a smaller file size of the overall extension.

7.3.2 Improved data fetching

Currently in the project data is being get and set all over the place. Most methods pull data whenever they are called. While this works well it means that code is being duplicated constantly and fetching data that is likely already being stored in the extension's current memory. It would be much more efficient to pull the data in once and pass it to every script that needs to use it. Then updating it at any location and setting the original getter to update its values. Finding a way to implement data fetching with promises would allow for a better

code structure too. Currently there have been race conditions and problems with call backs that have made other parts of the code structure less clear. Finding a better solution to these race conditions and call back problems would help the tidiness and efficiency of the code.

7.4 Integrating with EasyList

EasyList is one of many free to use lists that contains bad domains and elements for websites. Currently the list is around 2MB and contains hundreds of thousands of entries. During my project I tried to implement this into the program but was not able to after having troubles understanding the full schema it uses. EasyList has its own format for URLs to be written in which allows it to filter domains precisely, with specific attributes, letter patterns, extensions etc. This list would allow domain blocking that is much more on par with the current solutions that implement this. Implementing with EasyList would however mean adding major optimisation onto how each filter is checked. With tens of thousands of potential matches, and this happening before every request goes out it would need to have no major impact to the performance of the web page.

8 Conclusion

After the completion of this project and writing this report during the second semester of university, it can be concluded that the task of creating an Adblocker has been a success. The Chrome extension that has been created allows the user to easily select and block adverts through either an in page of elements or through the user selecting domains. As this project progressed and the realisation of how ad-blocking really works was made the focus of this project changed slightly. When the project started identifying adverts on a webpage was the focus I had in mind. Towards the end of the project the focus instead switched towards having a concise and useful selection of options available to the user.

The ad-blocker has a long way to go before it would be Chrome-store ready and could benefit from some re-structing and refactoring. Overall, the code that has been written manages to do the tasks it was created for without having any negative impacts on the users browsing experience. With the addition of a big-name filter list such as EasyList the application would soon be able to compete with other well-known ad-blockers that are already available on Chrome.

After the extension that was created from this project was tested against other extensions available online it was easy to see that this extension was much easier to use. The UX allowed for users to perform actions more easily when compared to other extensions. Other extensions did however still have good UX, but their options were more overwhelming, sometimes more confusing, and always locked away behind more clicks. To better improve the UX of the finished extension, some final front end work could help make the interface even easier to understand.

Overall, this project has been a success and has completed the task it set out to do.

9 Reflection on Learning

Throughout the duration of this project I have been able to further develop and familiarise myself with JavaScript and the Chrome extension API. JavaScript, a language that I previously had little knowledge of, began with a steep learning curve. Being used to object oriented programming and strict typing JavaScript proved very different but overtime became more inherent and its usefulness increased. Using the Chrome API was also a very tough start as I had previously never touched it. Thanks to hard work and good research skills I was able to make good use of the documentation provided online by Google and gain knowledge from other people's questions online. Coming out of this project I would indeed say that my programming knowledge and diversity has increased in many ways. Some notable gains of knowledge include but is not limited to, Using JSON effectively, Manipulating the DOM, using call back functions and making requests to websites.

While on this project I have been able to make use of skills I already had, reinforcing them, and furthering my affirmation of knowledge. To begin with using GIT has made the project much easier to handle and has allowed me to rekindle knowledge I do not use so often. Working with HTML and CSS is something I enjoy and has been a fun part of the project. Having not worked with them this year it was an enjoyable experience to play around the styles and see what I could create. Alternatively, while I still used bootstrap for some of the style, I was exposed to new bootstrap classes I had not previously touched.

It is important to note that many of my general and soft skills have been improved through this project. To begin with I have had to do lots of research from start to finish. Previously I had not undertaken much research or research on this scale but now feel much more confident in working alone on a big project. My time management has been tested and improved. Overall, this project has been a good opportunity for me to develop skills by myself and see a project through from start to finish. I am glad I have had this opportunity and will make use of all lessons learnt along the way.

My final thoughts and advice for anyone who is about to undertake their final year project is to plan and be proactive. You have a long period of time in which you're able to do this project and the quicker you work out a schedule, the less pressure you will have and the better your end result will be.

10 References

- [1] C. Elliott, "Yes, There Are Too Many Ads Online. Yes, You Can Stop Them. Here's How.," 09 02 2017. [Online]. Available: https://www.huffpost.com/entry/yes-there-are-too-many-ads-online-yes-you-can-stop_b_589b888de4b02bbb1816c297. [Accessed 05 04 2020].
- [2] "Malvertising: What It Is and How to Protect Yourself," 17 06 2017. [Online]. Available: <https://www.pandasecurity.com/mediacenter/malware/malvertising/>. [Accessed 05 04 2020].
- [3] "Coronavirus (COVID-19)," [Online]. Available: <https://www.cardiff.ac.uk/coronavirus>. [Accessed 13 05 2020].
- [4] R. Cassidy, "How to use custom filters," 08 02 2019. [Online]. Available: <https://help.getadblock.com/support/solutions/articles/6000161377-how-to-use-custom-filters>. [Accessed 21 04 2020].
- [5] "About," [Online]. Available: <https://easylis.to/pages/about.html>. [Accessed 20 04 2020].
- [6] "Adblock Plus filters explained," [Online]. Available: <https://adblockplus.org/filter-cheatsheet>. [Accessed 25 04 2020].
- [7] L. Moon, "An Agile Workflow That Keeps Tasks Flexible In Trello," 16 10 2017. [Online]. Available: <https://blog.trello.com/an-agile-trello-workflow-that-keeps-tasks-flexible>. [Accessed 25 04 2020].
- [8] "About remote repositories," [Online]. Available: <https://help.github.com/en/github/using-git/about-remote-repositories>. [Accessed 25 04 2020].
- [9] "webRequest," [Online]. Available: <https://developer.chrome.com/extensions/webRequest>. [Accessed 02 05 2020].
- [10] "chrome.storage," [Online]. Available: <https://developer.chrome.com/extensions/storage>. [Accessed 02 05 2020].
- [11] W. Palant, "Adblock Plus and (a little) more," 22 08 2006. [Online]. Available: <https://adblockplus.org/blog/investigating-filter-matching-algorithms>. [Accessed 05 05 2020].
- [12] "How to write filters," [Online]. Available: <https://help.eyeo.com/en/adblockplus/how-to-write-filters>. [Accessed 03 05 2020].

- [13] "Adblock plus," [Online]. Available: <https://github.com/adblockplus>. [Accessed 28 04 2020].
- [14] A. Rai, "Callbacks Vs Promises and basics of JS," 03 07 2017. [Online]. Available: <https://medium.com/@theflyingmantis/callbacks-vs-promises-and-basics-of-js-80d3d1515e81#a99f>. [Accessed 07 05 2020].
- [15] "Callback Hell," [Online]. Available: <http://callbackhell.com/>. [Accessed 13 05 2020].
- [16] "CSS," [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/API/CSS>. [Accessed 02 05 2020].
- [17] "chrome.contextMenus," [Online]. Available: <https://developer.chrome.com/apps/contextMenus>. [Accessed 01 05 2020].

11 Terminology

[Term. 1] Advert – an advertisement added onto a web page. This could come in a variety of multimedia forms.

[Term. 2] Ab-blocker – an application of some sort that stops adverts from being displayed.

[Term. 3] Google Chrome – A browser that can be used to access the internet, created by Google

[Term. 4] Google Chrome Extension – An additional piece of software that can be installed into the Google Chrome browser.

[Term. 5] API – Application Programming Interface, allows programming accessing features or data of another service.

[Term. 6] Agile – A short cycled methodology that is used for software development. This methodology allows frequent changes to the sprint (calendar)

[Term. 7] Whitelist – A list of allowed items

[Term. 8] Blacklist – A list of unallowed items

[Term. 9] JavaScript – A programming language primarily used in web development.

[Term. 10] jQuery – A superset of the JavaScript language that allows for easier development.

[Term. 11] Git – A version management software that allows collaboration and version control.

[Term. 12] GitHub – A website that allows you to host Git repositories (files).

[Term. 13] TortoiseGit – A graphical interface for Git.

[Term. 14] Trello – An online task board used to plan out agile workloads.

[Term. 15] WebRequest – A sub section of the Chrome API that allows viewing and manipulation of requests from a web page.

[Term. 16] Storage.Sync – A sub section of the Chrome API that allows storing data across google accounts.

[Term. 17] URL – The web address of a website.

[Term. 18] Domain – The web address of a website without anything else. google.com etc.

[Term. 19] HTML – Hyper Text Markup Language is the language in which web pages are written.

[Term. 20] DOM – Document Object Model is an interpretation of the HTML webpage that can be interacted with via JavaScript.

[Term. 21] CSS – Cascading Style Sheet can be used to style a web page visually.

[Term. 22] UI – User interface.

[Term. 23] UX – User experience, how easy it is to use.

[Term. 24] JSON – JavaScript Object Notation a format in which data can be stored in and interacted with via JavaScript.

[Term. 25] function – A defined piece of code.

[Term. 26] Filter – A list of items that can be used to compare to an item and either accept or reject.

[Term. 27] Element – An item on a web page.

[Term. 28] Bootstrap – A CSS stylesheet that has already been created and can be imported to the project to have styles already available.

[Term. 29] Class – An attribute that can be added to an Element to group it together.

[Term. 30] ID – An attribute that can be added to an Element to give it a distinct identifier.

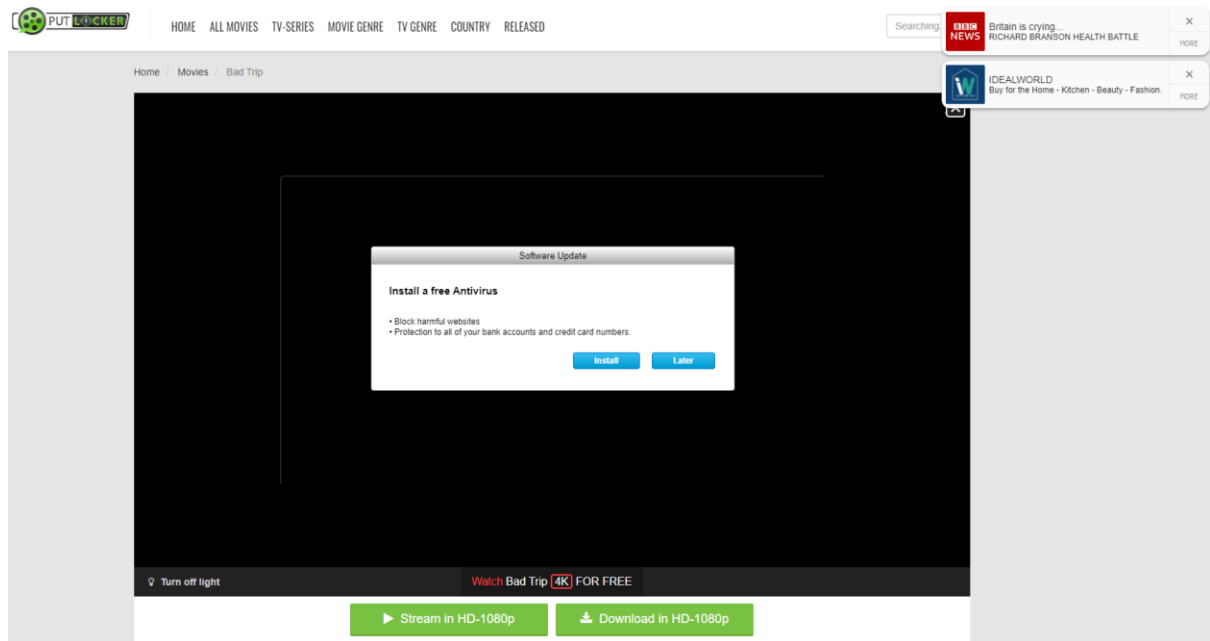
[Term. 31] <div><section> etc – examples of HTML elements.

[Term. 32] popup – The inbuilt extension interface that developers can customise.

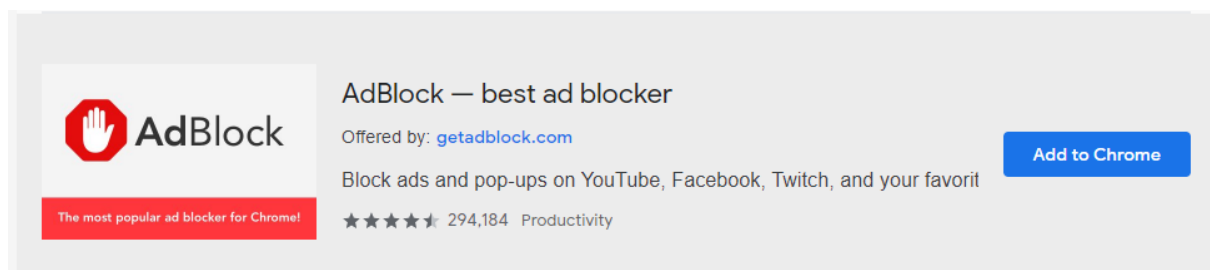
[Term. 33] EasyList – A list of predefined rules that can be used freely as a filter list.

[Term. 34] uBlocker, AdBlock and AdBlocker – Other ad-blocking extensions.

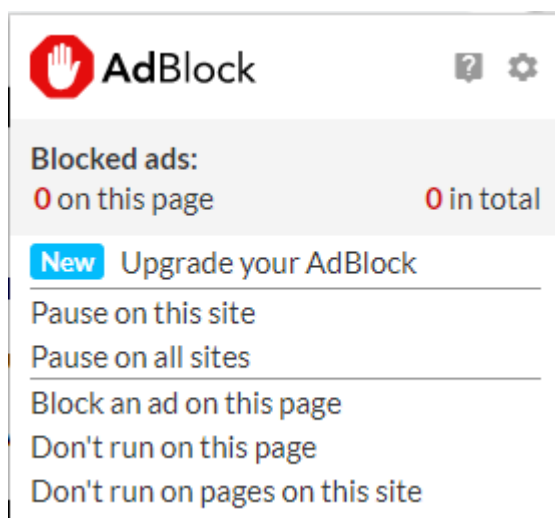
13 Figures



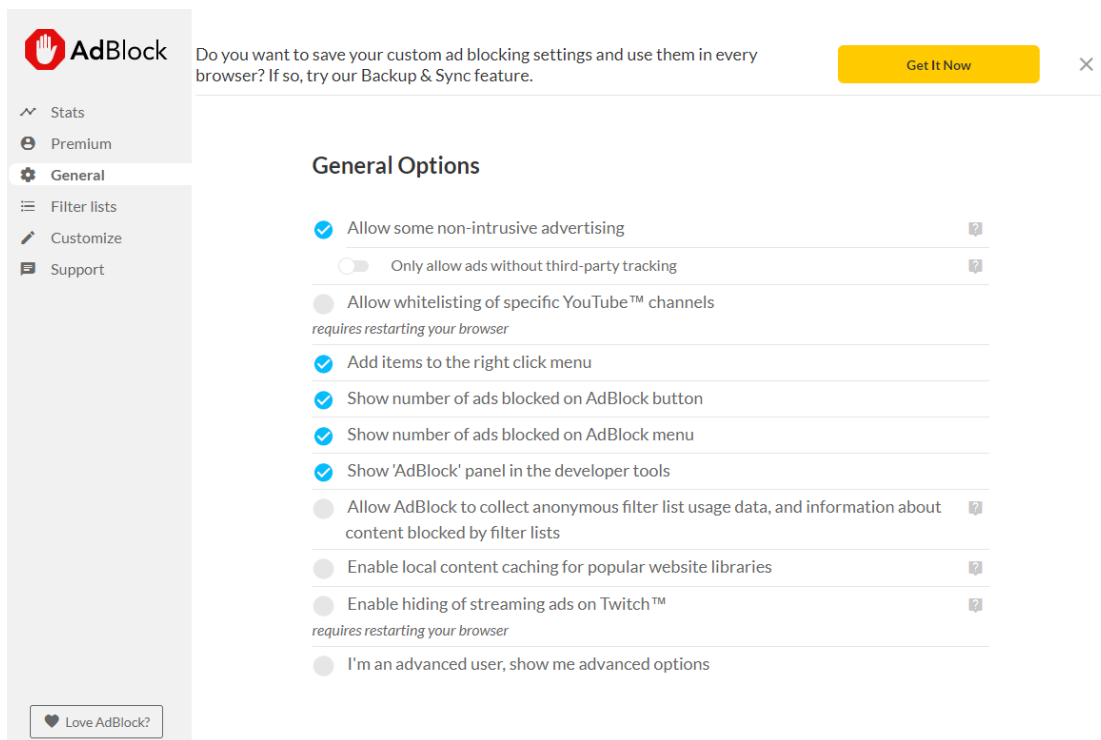
[Fig. 1] A website covered in malicious adverts with the intent on tricking the user.



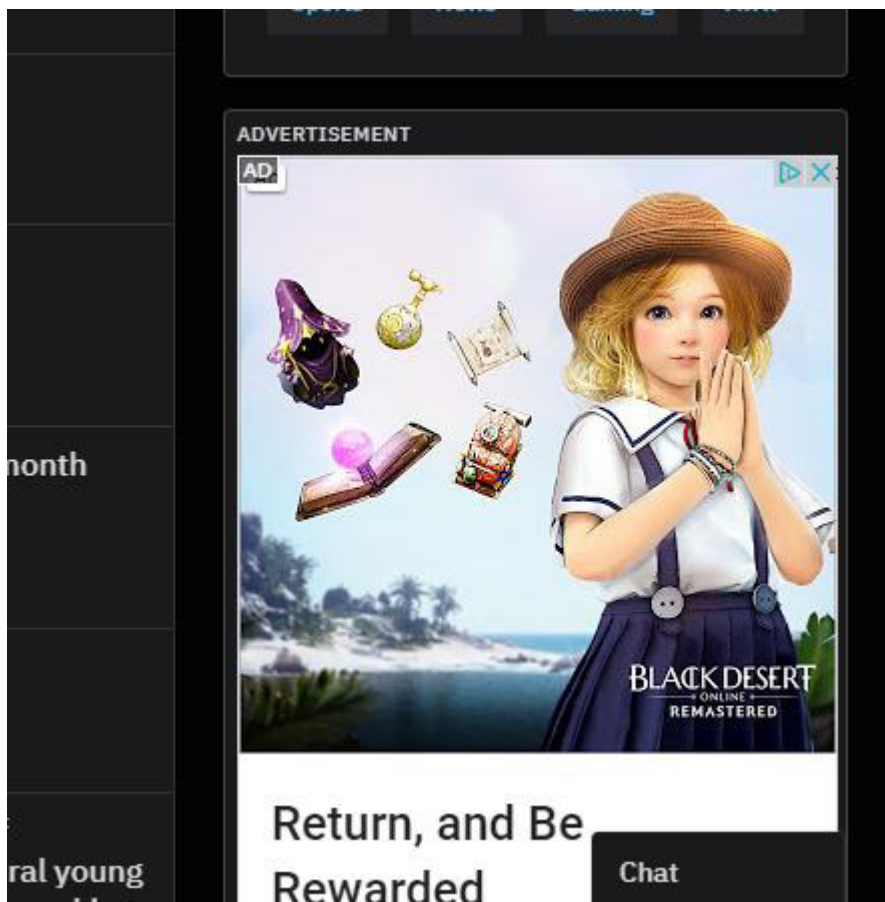
[Fig. 2] The AdBlock Chrome extension store listing.



[Fig. 3] The AdBlock extension popup.



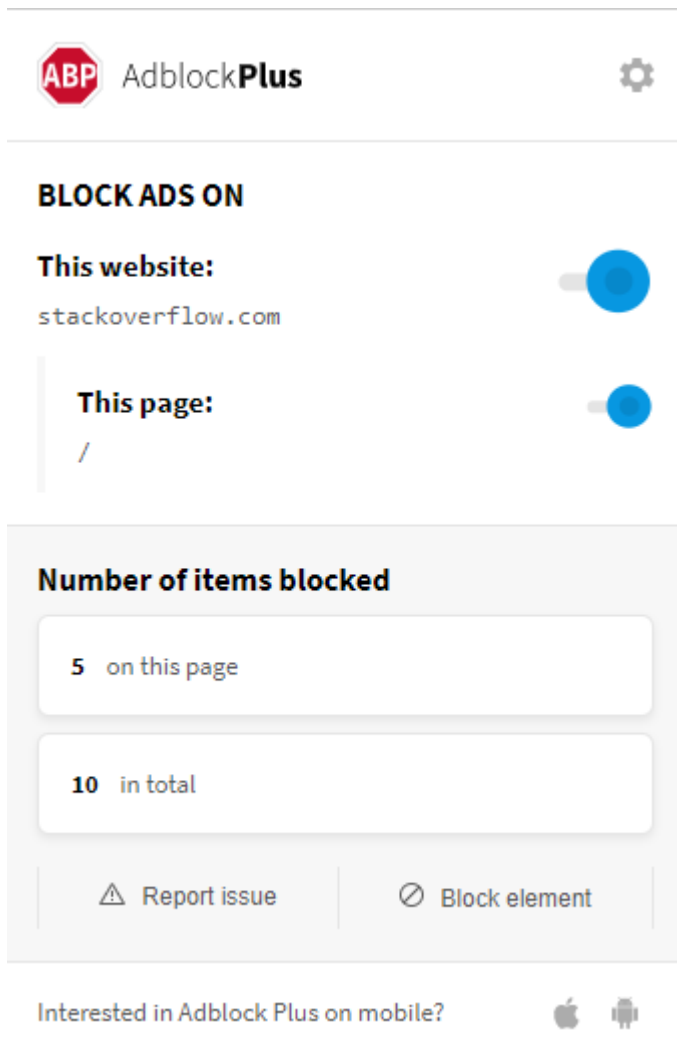
[Fig. 4] The AdBlock settings page overview.



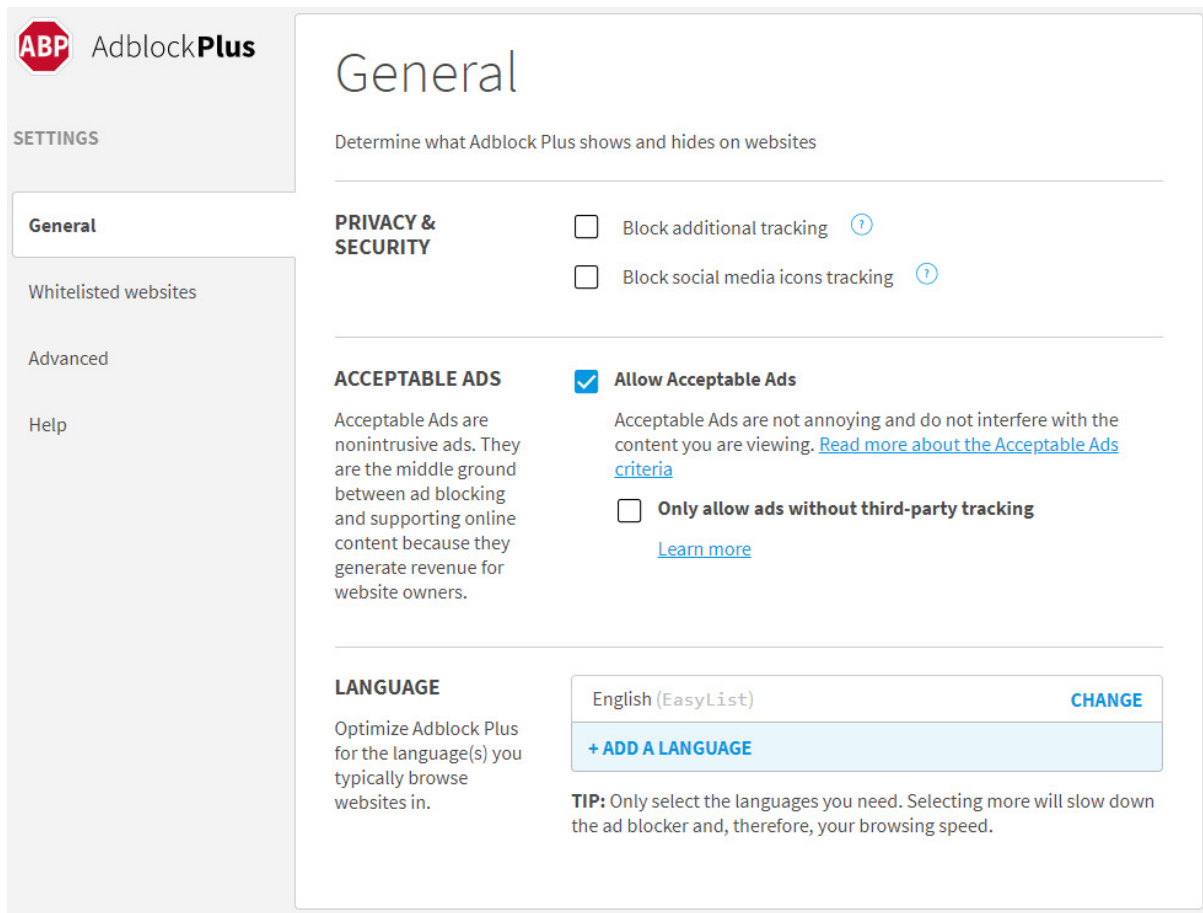
[Fig. 5] The AdBlock still showing some adverts on "reddit.com"



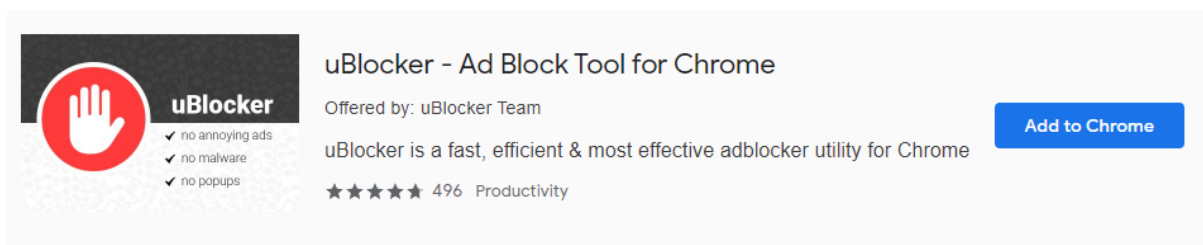
[Fig. 6] The Adblock Plus Chrome extension store listing



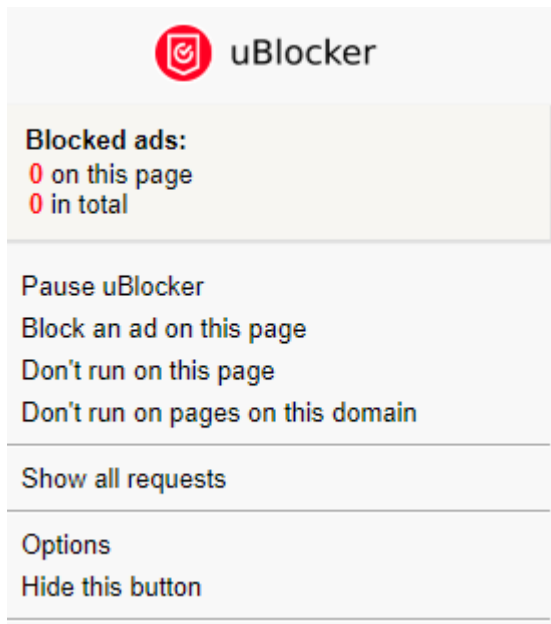
[Fig. 7] The Adblock Plus extension popup



[Fig. 8] The Adblock Plus settings page overview.



[Fig. 9] The uBlocker Chrome extension store listing

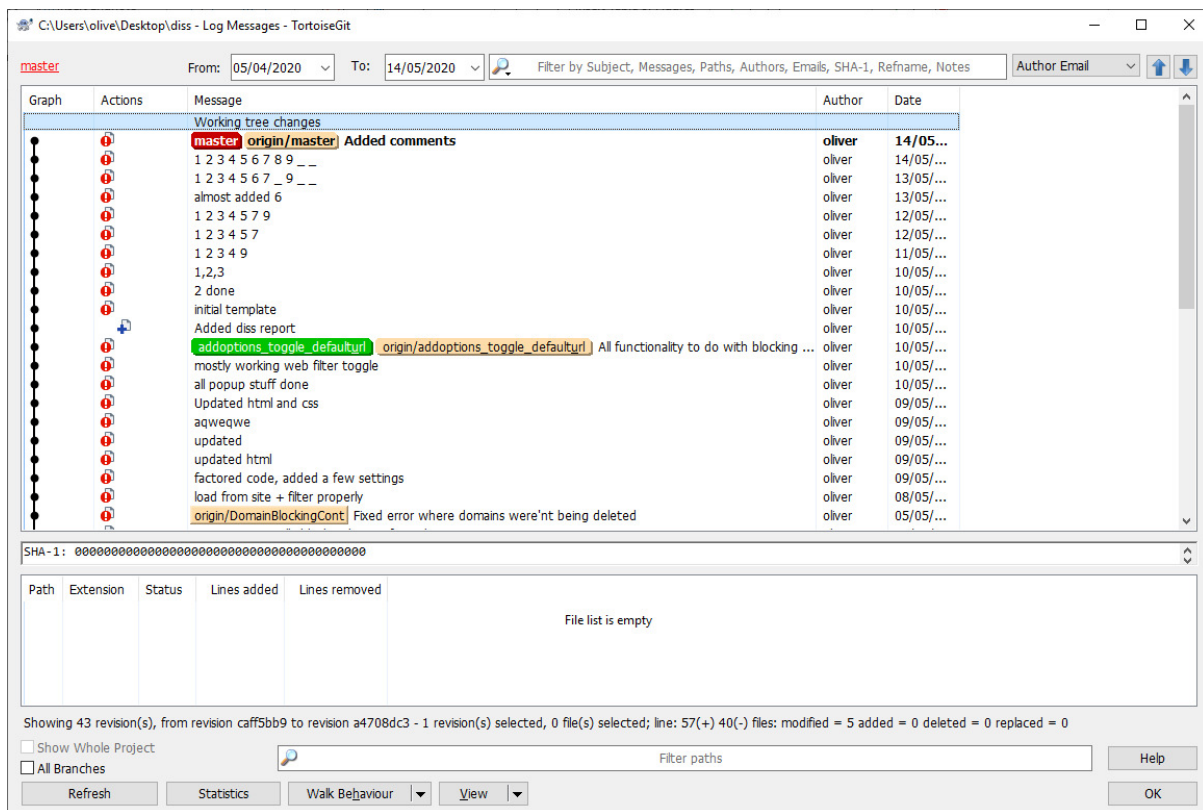


[Fig. 10] The uBlocker extension popup

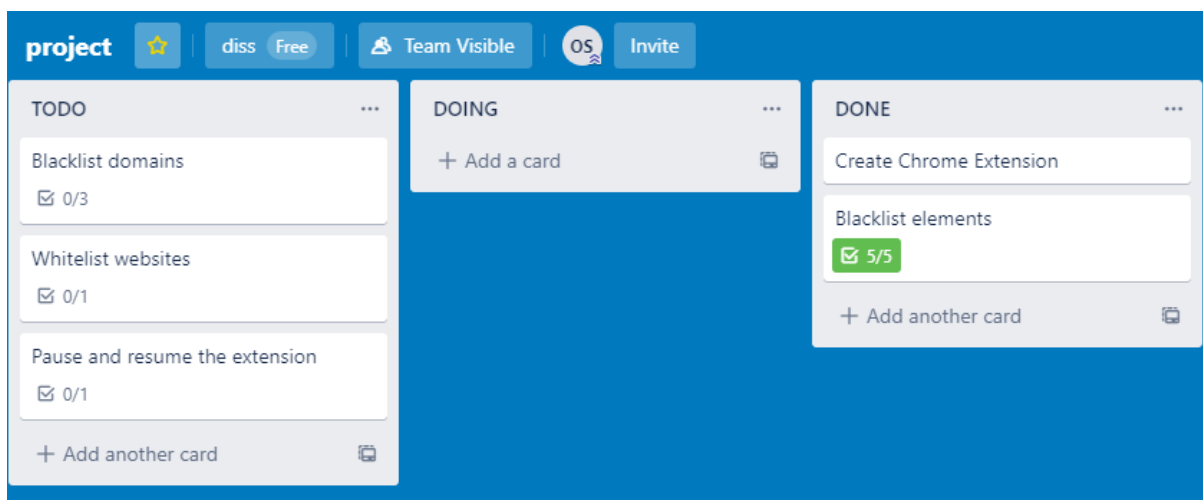
Options

- ☐ Allow some non-intrusive advertising
- ☒ Add items to the right click menu
- ☒ Show number of ads blocked on uBlocker button
- ☒ Show number of ads blocked on uBlocker menu
- ☒ Show 'uBlocker' panel in the developer tools
- ☐ Allow uBlocker to collect anonymous filter list usage data, and information about content blocked by filter lists
- ☐ I'm an advanced user, show me advanced options

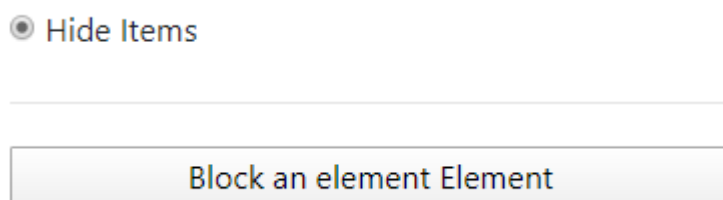
[Fig. 11] The uBlocker settings overview



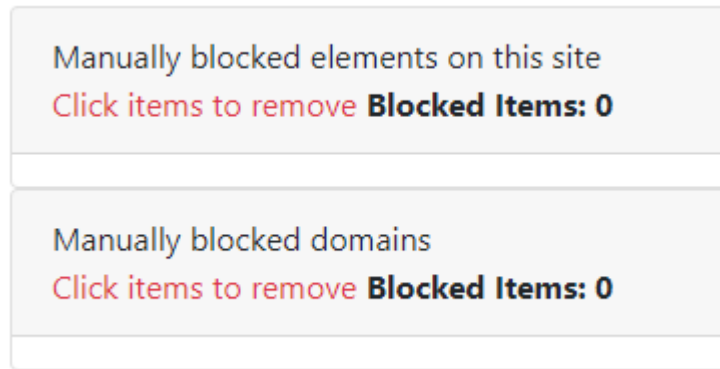
[Fig. 12] Overview of TortoiseGit



[Fig. 13] Screenshot of my Trello boards during development



[Fig. 14] Display of <hr/> tag used to divide sections in extensions popup



[Fig. 15] Display of <tables> used to display blocked elements and domains

```
<html>
<head>
  <link rel="stylesheet" href="popup.css"/>
  <link rel="stylesheet" href="bootstrap.min.css"/>

  <script src="\jquery.js"></script>
  <script src="popup.js"></script>
</head>

<body>
  <!--Extension Header-->
  <section class="title">...
</section>

  <hr/>

  <!--Options: Element Blocking-->
  <section class="toggles">...
</section>

  <hr/>

  <!--Actions-->
  <section class="Blocking">...
</section>

  <hr/>

  <!--Info Display-->
  <section class="cards">...
</section>
</body>
</html>
```

[Fig. 16] Code showing HTML sections being used inside popup.html

```

<!--Extension Header-->
<section class="title">
    <span>Oliver SY - C1635943</span>
    </span>Ad-Blocking Extension</span>
</section>
<hr/>

```

[Fig. 17] Code showing HTML first section inside of popup.html

```

<!--Options: Element Blocking-->
<section class="toggles">
    <input type="checkbox" id="runOnPage"/>
    <label for="runOnPage">Run on this page</label><br/>
    <div class="radio-buttons">
        <input type="radio" name="BlockType" id="BlockDisplay">
        <label for="BlockDisplay">Collapse Items</label>
        <br/>
        <input type="radio" name="BlockType" id="BlockVisibility">
        <label for="BlockVisibility">Hide Items</label>
    </div>
</section>

<hr/>

<!--Actions-->
<section class="Blocking">
    <button id="SelectElement">Block an element Element</button>
    <button id="ResetBlockedSite">Reset Blocked : Current Site</button>
    <button id="ResetBlockedAll">Reset blocked : All Sites</button>
    <button type="button" id="btn_BlockDomain">Block Domain</button>
    <input type="text" id="txt_DomainStr" placeholder="*://*.www.google.com/*"/>
    <button type="button" id="btn_ToggleExt">Enable Ext</button>
    <input type="text" id="txt_BlockExt"/>
    <button type="button" id="btn_SaveExt">Save</button>
    <input type="checkbox" id="chk_DefaultFilter"/>
    <label for="chk_DefaultFilter">Use Default filter</label><br/>
</section>

```

[Fig. 18] Code showing HTML second and third section inside of popup.html

```

<!--Info Display-->
<section class="cards">
  <div class="card">
    <div class="card-header">
      Manually blocked elements on this site
      <br/>
      <span class="text-danger">Click items to remove</span>
      <b><span id="blockedCount">Blocked Items: 0</span></b>
    </div>
    <table id="manuallyblocked" class="table">

  </table>
</div>

  <div class="card">
    <div class="card-header">
      Manually blocked domains
      <br/>
      <span class="text-danger">Click items to remove</span>
      <b><span id="blockedDomainCount">Blocked Domains: 0</span></b>
    </div>
    <table id="manuallyblockeddomains" class="table">

  </table>
</div>
</section>

```

[Fig. 19] Code showing HTML fourth section inside of popup.html

```

//Initialise settings
chrome.storage.sync.get(["Settings"], function(result){

    //Init Display Type
    var BlockType = result["Settings"]["BlockType"];
    if(BlockType == "Display"){
        document.getElementById("BlockDisplay").checked = true;
    }else{
        document.getElementById("BlockVisibility").checked = true;
    }

    //Init Whitelist
    var Whitelist = result["Settings"]["Whitelist"];
    chrome.tabs.query({active: true, currentWindow: true}, function(tabs) {
        var CurrentDomain = new URL(tabs[0].url).hostname;
        if(Whitelist.includes(CurrentDomain)){
            document.getElementById("runOnPage").checked = false;
        }
        else{
            document.getElementById("runOnPage").checked = true;
        }
    });

    //Init Default filter
    var UseDefaultFilter = result["Settings"]["useDefaultBlock"];
    if(UseDefaultFilter == false){
        document.getElementById("chk_DefaultFilter").checked = false;
    }
    else{
        document.getElementById("chk_DefaultFilter").checked = true;
    }

    //Init Web filter
    SwapWebFilterStatus(result);

    //Init Web filter url
    document.getElementById("txt_BlockExt").value = result["Settings"]["webBlockURL"];
});

```

[Fig. 20] Code showing initially set values from storage inside popup.js

```

1  $(document).ready(function(){
2
3      //Open a connection that can be used to send messages between scripts.
4  >   var port = chrome.extension.connect({ ...
6      });
7
8      //Sends a message telling the current tab to be reloaded
9  >   function ReloadPage(){ ...
11      }

```

[Fig. 21] Code showing callback via document ready event inside popup.js

```
document.getElementById("BlockVisibility").checked = true;
```

[Fig. 22] Code showing example of DOM selection and value setting inside popup.js

```
124 //OnChange handler for #runOnPage
125 //Sets whether the element blocking should happen on the current page
126 > $('#runOnPage').change(function(){...
127 });
128
129 //OnClick handler for #BlockDisplay
130 //Sets the block type of elements to "Display"
131 > $('#BlockDisplay').click(function(){...
132 });
```

[Fig. 23] Code showing example of click and change handlers inside popup.js

```
//OnClick handler for #btn_ToggleExt
//Changes the value that denotes if an external list should be used.
$('#btn_ToggleExt').click(function(){
    chrome.storage.sync.get(["Settings"],function(result){
        var curr = result["Settings"]["useWebBlockURL"];
        result["Settings"]["useWebBlockURL"] = (curr == true ? false : true);
        chrome.storage.sync.set(result, function(){
            port.postMessage("WebFilterUpdate");
            ReloadPage();
        });
        SwapWebFilterStatus(result);
    });
});
```

[Fig. 24] Code showing message being sent and a value updated inside popup.js

```

96 //Initialise blocked domains table
97 chrome.storage.sync.get(["Blocked"], function(result){
98   document.getElementById("blockedDomainCount").innerHTML = "Blocked Items: " + (result["Blocked"].length);
99   //Create a row for each domain in storage
100   result["Blocked"].forEach(element => {
101     var tr = document.createElement('tr');
102     var td = document.createElement('td');
103     var btn = document.createElement('button');
104     btn.innerHTML = element;
105     //OnClick function to remove domain
106     btn.onclick = function () {
107       var index = result["Blocked"].indexOf(element);
108       result["Blocked"].splice(index,1);
109       var currRow = this.parentElement.parentElement;
110       this.parentElement.parentElement.parentElement.removeChild(currRow);
111       //Refresh listeners on background page
112       chrome.storage.sync.set(result, function(){
113         port.postMessage({Reset : result["Blocked"]});
114       });
115       document.getElementById("blockedDomainCount").innerHTML = "Blocked Items: " + (result["Blocked"].length);
116     }
117     td.appendChild(btn);
118     tr.appendChild(td);
119     document.getElementById("manuallyblockeddomains").appendChild(tr);
120   });
121 });
122

```

[Fig. 25] Code showing table generation inside popup.js

```

//Set up storage.sync when the extension is installed.
chrome.runtime.onInstalled.addListener(function(){
  var storage = {};
  storage["Websites"] = {};
  storage["Settings"] = {};
  storage["Settings"]["BlockType"] = "Visibility";
  storage["Settings"]["webBlockURL"] = "";
  storage["Settings"]["useWebBlockURL"] = false;
  storage["Settings"]["useDefaultBlock"] = false;
  storage["Settings"]["Whitelist"] = [];
  storage["Blocked"] = [];
  chrome.storage.sync.set(storage);
});

```

[Fig. 26] Code showing initial data structure being generated on extension install inside background.js

```

result["Websites"][domain] = {classes:[], ids:[]}

```

[Fig. 27] Code showing element storage data structure being created inside SelectElement.js

```

chrome.storage.sync.get(["Settings"],function(result){
  result["Settings"]["BlockType"] = "Display";
  chrome.storage.sync.set(result, function(){
    ReloadPage();
  });
});

```

[Fig. 28] Code showing Get Set technique to modify data inside popup.js

```
//OnClick handler for #SelectElement
//Sends a message to the current tab telling it to begin element selection.
$('#SelectElement').click(function(){
    chrome.tabs.query({active: true, currentWindow: true}, function(tabs) {
        chrome.tabs.sendMessage(tabs[0].id, {beginSelection: true});
    });
});
```

[Fig. 29] Code showing popup.js sending a message to the SelectElement.js

```
chrome.runtime.onMessage.addListener(
function(request, sender, sendResponse) {
    if (request.beginSelection == true){
```

[Fig. 30] Code showing SelectElement.js Listening for messages

```
$("<style type='text/css'> a{pointer-events:none;} </style>").appendTo("head");
```

[Fig. 31] Code showing custom CSS being added to the head of the current web page from SelectElement.js

```
//Start selecting elements
$(document).mousemove(function(mouse){
    $(currentElement).removeAttr("style");
    var currentmouseover = mouse.target;
    while(currentmouseover.nodeName != "BODY" && (currentmouseover.id == "" && typeof $(currentmouseover).attr("class") === 'undefined' )){
        currentmouseover = currentmouseover.parentNode;
    }
    currentElement = currentmouseover;
    $(currentElement).css("border", "2px dotted tomato");
});
```

[Fig. 32] Code showing mousemove jQuery event in SelectElement.js

```
//Stop selecting
$(document).click(function(event){
    //Reset things
    $(document).off("mousemove");
    $(document).off("click");
    $(currentElement).removeAttr("style");

    StoreElements();
    location.reload();
})
```

[Fig. 33] Code showing SelectElement.js click event


```

function StoreElements(){
  //storage prep
  var classesToBlock = "";
  var idToBlock = "";
  if($(currentElement).attr("class") !== 'undefined'){
    $(currentElement).prop("classList").forEach(element => {
      classesToBlock += "." + CSS.escape(element);
    });
  }
  idToBlock = (currentElement.id == "") ? "" : ("#" + CSS.escape(currentElement.id));

  //storage
  chrome.storage.sync.get(["Websites"],function(result){
    //if nothing stored, make new object else edit old
    if(typeof result["Websites"][domain] === 'undefined'){
      result["Websites"][domain] = {classes:[], ids:[]}
      if(classesToBlock !== ""){
        result["Websites"][domain].classes.push(classesToBlock);
      }
      if(idToBlock !== ""){
        result["Websites"][domain].ids.push(idToBlock);
      }
      chrome.storage.sync.set(result);
    }else{
      if(classesToBlock !== "" && result["Websites"][domain].classes.includes(classesToBlock) == false){
        result["Websites"][domain].classes.push(classesToBlock);
      }
      if(idToBlock !== "" && result["Websites"][domain].ids.includes(idToBlock) == false){
        result["Websites"][domain].ids.push(idToBlock);
      }
      chrome.storage.sync.set(result);
    }
  });
}

```

[Fig. 34] Code showing overview of SelectElement.js StoreElements() function

```

//storage prep
var classesToBlock = "";
var idToBlock = "";
if($(currentElement).attr("class") !== 'undefined'){
  $(currentElement).prop("classList").forEach(element => {
    classesToBlock += "." + CSS.escape(element);
  });
}
idToBlock = (currentElement.id == "") ? "" : ("#" + CSS.escape(currentElement.id));

```

[Fig. 35] Code showing SelectElement.js variables being assigned

```

CSS.escape(currentElement.id));

```

[Fig. 36] Code showing CSS.escape() in SelectElement.js

```

var currentDomain = new URL(window.location.href).hostname;

//Add listener to changes in the DOM
chrome.storage.sync.get(["Settings"], function(whitelistCheck){
    if(!whitelistCheck["Settings"]["Whitelist"].includes(currentDomain)){
        document.addEventListener('DOMSubtreeModified', injectCSS);
    }
});

//Reloads the page when told to via message.
chrome.runtime.onMessage.addListener(function(request, sender, sendResponse) {
    if (request.reload == true)
        location.reload();
});

//Adds custom css style to the head of the page to block elements.
function injectCSS(){
    if(document.head){
        document.removeEventListener('DOMSubtreeModified', injectCSS);
        chrome.storage.sync.get(["Websites"], function(result){
            if(!jQuery.isEmptyObject(result)){
                if(typeof result["Websites"][domain] !== 'undefined'){
                    var cssString = "";
                    var style = document.createElement("style");
                    for(var key in result["Websites"][domain]){
                        result["Websites"][domain][key].forEach(item => {
                            cssString += item + ", ";
                        });
                    }
                    chrome.storage.sync.get(["Settings"], function(result){
                        cssString = cssString.slice(0, -2);

                        //Check if the settings are set to hide or collapse;
                        if(result["Settings"]["BlockType"] == "Display"){
                            cssString += " { display: none !important }";
                        }else{
                            cssString += " { visibility: hidden !important }";
                        }
                        style.innerHTML = cssString;
                        document.head.appendChild(style);
                    });
                }
            }
        });
    }
};

```

[Fig. 37] Code showing BlockElement.js

```
chrome.storage.sync.get(["Settings"], function(whitelistCheck){
    if(!whitelistCheck["Settings"]["Whitelist"].includes(currentDomain)){
        document.addEventListener('DOMSubtreeModified', injectCSS);
    }
});
```

[Fig. 38] Code showing BlockElement.js DOMSubtreeModified listener

```
//Adds custom css style to the head of the page to block elements.
function injectCSS(){
    if(document.head){
        document.removeEventListener('DOMSubtreeModified', injectCSS);
        chrome.storage.sync.get(["Websites"], function(result){
            if(!jQuery.isEmptyObject(result)){
                if(typeof result["Websites"][domain] !== 'undefined'){
                    var cssString = "";
                    var style = document.createElement("style");
                    for(var key in result["Websites"][domain]){
                        result["Websites"][domain][key].forEach(item => {
                            cssString += item + ", ";
                        });
                    }
                    chrome.storage.sync.get(["Settings"], function(result){
                        cssString = cssString.slice(0, -2);

                        //Check if the settings are set to hide or collapse;
                        if(result["Settings"]["BlockType"] == "Display"){
                            cssString += " { display: none !important }";
                        }else{
                            cssString += " { visibility: hidden !important }";
                        }
                        style.innerHTML = cssString;
                        document.head.appendChild(style);
                    });
                }
            }
        });
    }
};
```

[Fig. 39] Code showing BlockElement.js injectCSS function

```

//Context menu for right click
chrome.contextMenus.create({
  title: "Block this domain",
  contexts:["link"],
  onclick: function(clickedLink){
    //Create url format
    var link = new String(new URL(clickedLink.linkUrl).hostname);
    link = "*/**." + link + "/*";
    chrome.storage.sync.get(["Blocked"], function(result){
      if(result["Blocked"].includes(link) == false){
        result["Blocked"].push(link);
        chrome.storage.sync.set(result, function(){
          //Reset listeners and update list
          ResetManualDomainListeners(result);
          ReloadPage();
        });
      }
    });
  }
});

```

[Fig. 40] Code showing context menu creation in background.js

```

//Creates the blocklist for manual domains blocking
//Resets the listener for manual domain blocking
function ResetManualDomainListeners(obj){
  chrome.storage.sync.get(null, function(res){
    manualFilterList = [];
    manualFilterList = res["Blocked"];
    if(res["Settings"]["useDefaultBlock"] === true){
      manualFilterList = manualFilterList.concat(blocked);
    }

    chrome.webRequest.onBeforeRequest.removeListener(ManualDomainBlock);
    chrome.webRequest.onBeforeRequest.addListener(
      ManualDomainBlock,
      {urls: manualFilterList},
      ["blocking"]
    );
  })
}

```

[Fig. 41] Code showing ResetManualDomainListeners() function in background.js

```

//Creates listeners used to block requests related to the externally loaded block list.
function ResetWebBlock(){
    chrome.storage.sync.get(["Settings"], function(res){
        if(res["Settings"] !== undefined && res["Settings"]["useWebBlockURL"] === true && res["Settings"]["webBlockURL"] !== ""){
            //Send a request and recieve an array of strings
            var req = new XMLHttpRequest();
            req.open('GET', res["Settings"]["webBlockURL"]);
            req.onload = function() {
                webFilterList = [];
                webFilterList = req.responseText.split("\r\n");
            };
            req.send();
            //Reset listeners
            chrome.webRequest.onBeforeRequest.removeListener(WebDomainBlock);
            chrome.webRequest.onBeforeRequest.addListener(
                WebDomainBlock,
                {urls: ["<all_urls>"]},
                ["blocking"]
            );
        }
        else if(res["Settings"] !== undefined && res["Settings"]["useWebBlockURL"] === false){
            chrome.webRequest.onBeforeRequest.removeListener(WebDomainBlock);
        }
    })
}

```

[Fig. 42] Code showing ResetWebblock() from background.js

```

//Send a request and recieve an array of strings
var req = new XMLHttpRequest();
req.open('GET', res["Settings"]["webBlockURL"]);
req.onload = function() {
    webFilterList = [];
    webFilterList = req.responseText.split("\r\n");
};
req.send();

```

[Fig. 43] Code showing request being made to external web page from background.js

```

chrome.webRequest.onBeforeRequest.addListener(
    WebDomainBlock,
    {urls: ["<all_urls>"]},
    ["blocking"]
);

```

[Fig. 44] Code showing onBeforeRequest in background.js

```

//returns whether the url should be blocked
//If the url contains a string loaded from the external block list.
var WebDomainBlock = function(details) {
    return {cancel: webFilterList.find(element => details.url.includes(element)) !== undefined}
}

```

[Fig. 45] Code showing background.js function WebDomainBlock used by onBeforeRequest

```

//Creates the blocklist for manual domains blocking
//Resets the listener for manual domain blocking
function ResetManualDomainListeners(obj){
    chrome.storage.sync.get(null, function(res){
        manualFilterList = [];
        manualFilterList = res["Blocked"];
        if(res["Settings"]["useDefaultBlock"] === true){
            manualFilterList = manualFilterList.concat(blocked);
        }

        chrome.webRequest.onBeforeRequest.removeListener(ManualDomainBlock);
        chrome.webRequest.onBeforeRequest.addListener(
            ManualDomainBlock,
            {urls: manualFilterList},
            ["blocking"]
        );
    })
}

```

[Fig. 46] Code showing ResetManualDomainListeners inside of background.js

```

//returns whether to block url from manual domain blocking
function ManualDomainBlock(){
    return {cancel: (manualFilterList.length > 0)};
}

```

[Fig. 47] Code showing ManualDomainBlock function used in onBeforeRequest for manual domain filtering in background.js

```

chrome.webRequest.onBeforeRequest.addListener(
    ManualDomainBlock,
    {urls: manualFilterList},
    ["blocking"]
);

```


[Fig. 48] Code showing background.js onBeforeRequest for manual domain filtering

```


JS Blocked.js > ...
1 //This file can be used to store a list of domains to be shipped with the extension.
2 //The current values are here to show this functionality works.
3 var blocked = [
4   "*/**/.0001-cab8-4c8c-43de.reporo.net/*",
5   "*/**/.002-slq-470.mktoresp.com/*",
6   "*/**/.004-btr-463.mktoresp.com/*",
7   "*/**/.005.free-counters.co.uk/*",
8   "*/**/.006.free-counters.co.uk/*",
9 ];

```

[Fig. 49] Code showing Blocked.js a file used to ship a default domain blacklist


Step 1: Figure out what to block

Slide the slider until the ad is blocked correctly on the page, and the blocked element looks useful.



Blocked element:

```


< DIV
  id="test_control_body"
  class="not-delayed" >

```

Looks good

Cancel

[Fig. 50] Adblocker verification example 1


Last step: What makes this an ad?

What do you think will be true about this ad every time you visit this page?

- ☒ **Type** will be *DIV*
- ☒ **id** will be *test_control_body*
- ☒ **class** will be *not-delayed*

That matches 1 item on this page.

Not sure? Just press 'Block it!' below.
The filter, which can be changed on the Options page:

```
DIV[id="test_control_body"][class="not-delayed"]
```

Block it!

Back

Cancel

[Fig. 51] Adblocker verification example 2

Oliver SY - C1635943 Ad-Blocking Extension

☒ Run on this page

☐ Collapse Items

☒ Hide Items

Block an element Element

Reset Blocked : Current Site

Reset blocked : All Sites

Block Domain

Enable Ext Save

☐ Use Default filter

Manually blocked elements on this site

Click items to remove **Blocked Items: 0**

Manually blocked domains

[Fig. 52] Finished Interface of Chrome extension