FYP Final Report

Monitoring User Preferences - An Application to Desktop Environments

Supervisor: Dr Jesús García Galán

Student: Michael Hallinan - 12134635

Table of Contents

|  |  |
| --- | --- |
| Section Title | Page |
| Table of Contents | *1* |
| [Acknowledgements](#_Project_Summary) | *2* |
| 1. Introduction and Objectives | *3* |
| 1.1 Introduction |  |
| 1.2 Motivation |  |
| 1.3 Objectives of Project |  |
| 2. Research |  |
| 2.1 Introduction |  |
| 2.2 Papers |  |
| 2.3 Existing Projects |  |
| 2.4 Other Sources |  |
| 3. DESIGN |  |
| 3.1 Introduction |  |
|  |  |
|  |  |
| 4. IMPLEMENTATION |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| 5. SYSTEM TESTING AND EVALUATION |  |
|  |  |
|  |  |
|  |  |
|  |  |
| 6. CONCLUSIONS AND RECOMMENDATIONS |  |
|  |  |
|  |  |
|  |  |

1: Introduction and Objectives

# 1.1 Introduction

Monitoring User Preferences - An Application to Desktop Environments

*A critical requirement of software systems is to satisfy the needs of the user.*

The focus of this project will be on desktop environments and will be aimed to identify strategies to capture user preferences and detect changes. This is intended to be done explicitly by requesting user feedback, as well as implicitly through mining of user behaviour or system logs.

Proper user preference specification is believed to be very important for maximum user enjoyment when using software. This project will focus upon acquiring these specifications through monitoring the user and to apply this information to create adaptive software tailored to the individual user.

Many users are not fully aware of the configuration capabilities of a system. Advanced configuration options can also intimidate novice users of a system. This project will seek to present a simple configuration but allow for more complicated configuration through its use of user-profiling. The chosen solution will be a “Web Assistant” Extension to the Google Chrome web browser that will monitor the users browsing habits and provide recommendations to the user based upon this.

The enormous growth in web content and software applications as well as the increased popularity of these products has led to increased demands for user accessibility. In recent years, there has been much effort put into developing user-adaptive web-sites and applications (Perkowitz & Etzoni, 2000).

Though many techniques exist, a commonly used method is based upon monitoring User Preferences. User preferences are frequently assessed in order to improve the user experience.

### 1.1.1 User Preferences

A user is described as being “any person who uses or operates something”. (Oxford Dictionary, 2016) User Preferences can be described as “Specifications or settings for features of a software program as defined by the user.”

Another term frequently used for the type of study is “User Profiling”. A user profile has been defined as the description of the user interests, characteristics, behaviours, and preferences. User profiling is the practice of gathering, organizing, and interpreting the user profile information. (Godoy & Amandi, 2005)

These are frequently options selected by users in order to simplify or enhance their experience while using a software program. Explicitly defined examples include simple alterations such as colour, font, or language as well as more advanced modifications such as key-binding or behavioural changes.

A common example of implicit information gathering would be the use of link personalisation to recommend items based on buying history or ratings by e-commerce applications. This is detailed further in (Rossi, et al., 2001)

### 1.1.2 Proposed Solution

This project is intended to study and monitor these preferences in order to allow for the optimal experience for users of all levels. This will be accomplished through the gathering of information from the user and modification of the desktop environment accordingly. This will be done explicitly through user feedback such as user-defined settings or implicitly through data logging and mining.

Many existing studies and papers have already dealt with similar topics and these will be detailed later in the **Research** section of the report. This project is intended to focus some of these studies into a single project.

Though numerous ideas and possible proposals were considered, the approach selected for this project was a scheduling and recommendation system for a web browser. The main part of the extension is the scheduler, which recommends to the user websites at certain times based upon their past browsing history. The second part of the extension is a recommender system that will show the user similar sites to those they are viewing. The final part of the extension is a profiling system that will create “snapshots” of the user’s patterns in order to properly complete a User Profile of the user and further recognise patterns in their browsing.

The goals for the planned extension are for it to contain various methods of retrieving data from the user through feedback such as buttons and option pages, as well as monitoring the browsing patterns of the user in order to further present recommendations and alter the interface accordingly.

This solution is intended to support the User’s web experience through the recommendation of both new sites and schedule entries. The scheduler will remind the User of certain websites at the appropriate times with a popup notification, and facilitate them by allowing the User to navigate to that page if the notification is clicked. This should allow for a more fluid and enjoyable experience from the user.

# 1.2 Motivating Factors

When I set out to select a subject for my Final Year Project, I wanted to pick something that was relevant but was also something that I could be interested in. Initial ideas for a self-submitted project entailed a voice activated user-access program similar to the existing digital assistants Siri (Apple Inc., 2011) and Cortana (Microsoft, 2015). This idea was grown from my interests in Machine Learning and Artificial Intelligence, as well as my interest in language parsing and syntactic analysis. Another reason for the decision to create a digital assistant was from my experiences on Co-op, where I worked as IT Help Desk in a bank in Luxembourg. While working, I saw the problems that many users had when using their computers. Many of these problems could be quite simple, but novice computer users would be intimidated. I had hoped to create an interface that could solve these problems for the users as they arose. This would have uses of increasing productivity in numerous organisations.

Though I had hoped to base my project upon general user interface assistance, upon beginning research into that initial project I began to realise the sheer magnitude of the task. I decided to focus the scope of the project into a more concise area because of this. One of the Final Year Project Proposals was based upon User Preferences, and after meeting with the project supervisor, and after further research into the subject, I decided that this project was very similar to my initial goals and felt that the experience of the supervisor would be a huge benefit. (REPHRASE?)

I began initially by reading papers by (Iglesias, et al., 2012) and (Holland, et al., 2003) that were provided by my supervisor. These papers mentioned the use of Machine Learning and heuristic algorithms in determining the user preferences, as well as the methods for collecting user information. This helped to give me an idea of where to begin my project. Through the use of Google Scholar, I was able to find more papers that dealt with these topics, and form ideas of how best to propose a solution to the project. Though the initial research and design was quite slow, once I began implementing my idea everything quickly began to come together.

# 1.3 Objectives of the Project

The objective of this project was to create a program capable of adapting to the user’s requirements. This was to be done through the monitoring of the user’s habits and preferences. These habits were to be processed and used to create an interface more tailored to the user. Extensive testing was to be done in order to ensure that the proposed solution would successfully tailor to the user’s needs as well as be readily and easily usable by everybody.

Prior to commencing the solution for this project, I felt it necessary to examine existing research as well as any similar projects or other documentation in this area. This existing research has been presented in **Chapter 2** along with a summary of the initial findings, while the efforts completed for the design of the implementation have been described in **Chapter 3**. Thedetails for the selected implementation have been given in **Chapter 4**, with the description and evaluation of testing conducted presented in **Chapter 5**.Finally the conclusion and recommendations for further projects are shown in **Chapter 6**.

2: Research

# 2.1 Introduction

An important part of the project was the research and reasoning behind any decisions made. Numerous studies and papers exist in similar fields and these were consulted during the initial stages of the project. A number of papers were supplied by the supervisor and through the use of Google Scholar, more papers were found. These papers were viewed using the University of Limerick Library’s services, and the books used for studying were also borrowed from the library.

A large portion of the first semester was spent researching the topic and forming possible ideas for the project. Below you will find the research papers that were read as well as any similar existing projects reviewed. You will also find the books and other sources that were consulted as part of the research.

# 2. 2 Research Papers

The initial papers consulted were *Creating Evolving User Behavior Profiles Automatically* (Iglesias, et al., 2012) and *Preference Mining: A Novel Approach on Mining User Preferences for Personalized Applications* (Holland, et al., 2003), among others. They detailed self-learning systems and were the first resources consulted upon initiation of the project. These papers were useful in presenting proper recording technique as well as showing previous studies in order to help form an idea for a possible solution for the project. The reports described the processes completed and their methods of creating a user profile from the user’s data.

References from these papers were consulted for further reading material. Google Scholar was also consulted to find more recent papers referencing these sources. These efforts led me to *Detecting innovative topics based on user-interest ontology* (Nakatsuji, et al., 2009) and *User Preference Mining Techniques for Personalized Applications* (Holland & Kießling, 2004)*.*

These papers spoke further about the processing of the information, such as mining algorithms and applications of the information. The subjects of both papers were based upon web-based applications which helped to cement the decision to work on a web-based application for the project. Nakatsuji, et al. based their report on recognising user-interest. This topic choice was major influence in where to base the proposed solution for this project, and led to the decision to implement the Recommendations system in the project.

The papers spoke at length about the algorithms used and processing methods but were brief with the description of the methods of information gathering that were used. This helped to decide how to process the information but it was necessary to look elsewhere in order to find other possible methods of gathering the information for processing.

Papers were researched dealing with search engine queries such as (Silverstein, et al., 1999) which dealt with the analysis of the search requests made to the AltaVista Search engine over a period of six weeks. This documented the analysis of the queries and detailed the session information and correlation between them. The paper researched most similar to the project solution was about a system incorporated into Internet Explorer that would profile a user’s interest based upon their recent browsing history (Grčar, et al., 2005). This system would perform clustering algorithms upon word-vectors formed from the text content of pages and identify the user’s interests. This was very similar to the eventual project solution and helped to influence the final decision.

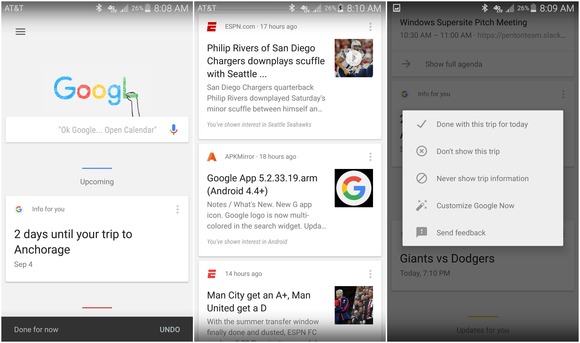
# 2.3 Existing Projects

### 2.3.1 Google Now

The most prevalent example found of proper monitoring of User Preferences is Google Now by Google. This application gathers information from users’ search history in order to display possibly useful material related to fields the user may be interested in. The application can then gather explicit information from the user such as feedback to this material or selection of certain fields such as “Home” location. This clearly displays both implicit (material) and explicit (feedback) information gathering. It also processes this information to display information relative to the user, as is intended by our application.

The application may also gather location information using Wi-Fi or GPS. This information may be used to automatically determine a “Home” location and a “Work” location, along with promoting local news sources. These locations will then be used to alert the user to weather and traffic conditions around these locations. This information is processed primarily by recording location versus time, with night time locations being evaluated higher as being home locations. Daytime locations, particularly weekdays, would also be evaluated more highly as possible work locations. Another capability of the application is contextual assistance such as displaying local attractions if the user is in a location far from their “Home” location.

Research has been presented recently investigating the abilities of Google Now as a Personal Assistant (Guha, et al., 2015). The research details the personalisation functionalities of the application as well as the contextual assistance provided. This paper detailed the current algorithms and functionalities of Google Now, such as context identification and task assistance that are currently in use, as well as the shortfalls and possible growth areas for the application.

  
Figure 2.1: Sample display for Google Now.   
This shows user calendar information on the left, relevant news articles in the centre, and user feedback on the right.  
Source: http://www.greenbot.com/article/2979664/google-apps/google-now-gets-visual-overhaul-with-new-logo-and-bursts-of-animation.html

### 2.3.2 Flix Plus by Lifehacker

Originally entertaining the idea of a Netflix extension for Chrome, the many existing products were reviewed. Though these applications were less focused upon adaptive interfaces and information gathering, they did perform actions similar to those desired by the project application. A primary example of a similar extension was *“Flix Plus”* by Life Hacker. This extension performed many of the originally conceived ideas for the Netflix implementation. These included interface overlays and improved customisation options for the interface such as hiding elements or highlighting certain aspects.

However, this extension used only explicit user information gathering and failed to display any signs of an automatically adapting interface or user profiling. Most of the features contained were limited to improved user settings and primarily covered visually improving the user interface. Examples include blurring of episode information and dynamic removal of show categories.

### 2.3.3 Google Analytics

Gathering the information to be processed is an important part of the project. A popular method used among the papers and other projects researched was Web Analytics. Web Analytics is defined as *“the measurement, collection, analysis, and reporting of web data for purposes of understanding and optimising web usage”* (Burby, et al., 2007). The most widely used Web Analytics service is another product by Google, Google Analytics. Web analytics services track and report data such as website traffic. This can be utilised by an organisation in order to garner information about the visitors to this site.

This information can be used for more accurate advertising and other services tailored to the user. Advertising agencies can use this information to create more focused targeted ads. As advertising is frequently based upon a Pay-per-click (PPC) model, this allows for greater ad revenue. This information can also be used for site content, by understanding the popular areas of the website and focusing efforts and expansion in these areas and possibly increasing traffic and revenue.

### 2.3.4 Proactive Assistant

Proactive Assistant is a recently released feature created by Apple Inc. in order to work along with many of their other existing applications such as Siri, the voice-activated assistant, found on many Apple devices. This system is designed to utilise the information gathered from the user’s data stored on the device, such as use of Siri, in order to form a user profile. This application was designed to be a competitor to Google Now, which is described above in section 2.3.1.

Proactive Assistant utilises the user’s profile in order to assist the user similar to those functions performed by Google Now. It will avail of other Apple services such as Wallet, an application to store coupons, tickets, and boarding passes, in order to tailor the service more closely to the user’s needs. The program will also recognise the patterns of use for certain applications and will provide a button to that application on the Proactive Assistant screen. Other examples are notifications for nearby restaurants at times such as lunch and dinner and other relevant locations, similar to those functionalities performed by Google Now. (Lardinois, 2015)

A key highlight of the Proactive Assistant is on user privacy. Much of the processing and information will be performed locally on the device and will not synchronise with the user’s Apple ID. Online processing will be done with a randomised identifier and no information will be shared with third parties. This focus on privacy was markedly different from Google Now and was considered while designing the implementation for the project.

### 2.3.5 Personal Smart Assistant for Digital Media and Advertisement

*Personal Smart Assistant for Digital Media and Advertisement* (Hussain, 2013) was a thesis found online from the University of Western Ontario. The thesis dealt with a subject similar to that of my project and had performed research of its own into related fields. The thesis detailed a generic personal smart assistant that provides relevant assistance to the user based upon a profile developed through modelling the user’s interests and behaviours. This report also proposed a service whereby the assistant would recommend an advertisement to the user. This idea was partially utilised when designing possible methods of commercialisation, as detailed in section 3.2.2 of this report.

The thesis spoke at length about the proposed concept of the Personal Smart Assistant and the methods used in designing the software. This detailed a User Model composed of a Behaviour Model, consisting of deliberative (Goal-driven) and reactive (Events-triggered) behaviour, which would capture the user’s interests into an Interests Model. Although the thesis dealt with a very similar topic to this final year project, it was not discovered until the later stages of the implementation, and had only a minimal impact upon the final product.

### 2.3.6 Summary of Existing Products

Will I do this?

# 2.4 Other Sources

Other than existing Academic Papers, a number of books were consulted about datamining and machine learning, as well as human-computer interaction. Many of the papers had mentioned the use of datamining for analysing the collected data, and further research was done into the possible techniques that could be used by the project.

The first book read was *Datamining: Practical Learning Tools and Techniques* (Witten, et al., 2011). This book detailed many of the popular datamining techniques and algorithms. It also contained numerous examples of the uses of these mining techniques in order to further explain when they are used. This was particularly useful for understanding these techniques before delving deeper into their uses. These explanations and ideals were greatly beneficial when deciding how to process the information that would be gathered.

After initialstudy of Datamining, *Machine Learning: A Probabilistic Perspective* (Murphy, 2012) was consulted in order to further research possible methods of gathering meaningful and useful information from the collected data. Many of the papers mentioned using clustering and other machine learning techniques in order to mine the data and this book was used to further research these techniques. This was important for full understanding of the techniques as well as selection of the processes used. This helped to improve upon the explanations of the procedures detailed in the previous book. Although the processing system used was different to those detailed in the book, many of the explanations were vital for creating the eventual processing algorithm.

As the interface and user-interactability was a vital part of the project design, a number of user-interface books and guides were consulted. *Don’t Make Me Think: A Common Sense Approach to Web and Mobile Usability* (Krug, 2014) was a very short but enlightening book revolving around the importance of simplicity in design. The book itself was made to be short in order to exemplify good design and simplicity. Much of the book detailed around creating interfaces that are self-explanatory and allow for quick navigation and require little “thinking” in order to perform an action. This thought process was carefully preserved when designing the interface of the project.

*The Design of Everyday Things* (Norman, 2013) was also selected in order to help understand the thought process behind many designs. This book delved into the psychology of everyday actions as well as understanding faults of bad design and how “Human Error” is often attributed to cases of faulty design. In this book, Norman stressed the importance of intuitive understanding and being able to use anything with as little instruction as possible. This book was important for understanding common pitfalls and constraints for the design of the program as well as techniques for the testing of the program interface.

Much of the ideology and information discussed within the books was utilised when designing the extension. The usability and design principles used when designing the extension were influenced greatly from these books. Many of the testing methods were also influenced by examples described within these books.

3. Design

# 3.1 Introduction

The design of the extension was of high importance. Much of the extension focused on appealing to the user and of being simple and straightforward. This meant that much time was spent designing the interface as was spent designing the software itself. Usability and Learnability were important characteristics considered when creating the interface. One of the vital necessities of the project was to be simple to use and unintimidating to new users. Further efforts were put into extensibility and maintainability, for adding and modifying the features of the software. Efficiency and reliability were also crucial for user acceptance of the intended solution. Security was another concern in design, and necessary measures needed to be put in place.

In the following sections you will find details of the design process for creating the extension. This will detail much of the thought process behind many of the decisions as well as the alternative ideas that were considered.

# 3.2 System Overview

### 3.2.1 Project Features

The intentions for the project are to create a profile for the user that will facilitate their computer use. The proposed solution was a browser extension that would consist of the following parts:

* Automatic Scheduling
* Website Recommendation
* Browsing Pattern Recognition
* Advanced Customisability

*Automatic Scheduling* would monitor the User’s web browsing and provide a schedule based upon patterns mined from the data. This schedule would prompt the user at certain times to open certain websites. This was intended to assist users by predicting their actions and providing a convenient method of opening the desired website.

*Website Recommendation* would provide the users with a list of similar websites to those visited by the user. The extension would automatically profile the websites visited by the User and prompt possibly similar websites. Recommendations would hopefully benefit the user’s experience by providing

*Browsing Pattern Recognition* would create patterns or profiles of the User’s tab configuration and recognise these patterns as they occur. Each tab would be categorised similar to the *Website Recommendation* and these patterns would be used to form a user profile. This user profile would be used to prompt recommendations similar to the scheduling above.

*Advanced Customisability* would allow the user to customise each of the features of the program. Part of the design for the extension highlighted using configurable variables. Each feature of the extension will also be able to be toggled on or off. This allowed the user to fully customise the many aspects of the extension.

### 3.2.2 Capturing User Preferences

As mentioned above, User Preferences would be gathered primarily through the monitoring of the user’s browsing. Browsing History will be processed by the application and used to create a schedule for the user. This schedule can be edited by the user at any time allowing them add or remove items in the schedule.

The Browsing Pattern Recognition and Recommendations will also gather browsing information automatically. This will be done by processing the URL and title of each tab and check for certain words or tags and assigning each tab to a category. This pattern of tabs and categories would be saved and could be used to predict user actions or preferences and would directly assist in forming the User Profile.

### 3.2.3 Processing User Information

Initial design stages involved a separate Java web server that would process the User’s information. This would have utilised the open-source WEKA plugin (WEKA, 2015) to process the user information. This server would have communicated with the extension and allowed for accessing other online resources. This was eventually rejected due to possible issues with latency, connectivity, as well as server upkeep and other possible complications that may arise.

Instead it was decided to process the information internally using JavaScript. This was performed internally by the extension. Data could be gathered from the extension using Google Chrome’s API and inbuilt functionality. This data was processed and saved locally, benefitting from the synchronised storage of Google Chrome, and allowing for further cross-compatibility across devices.

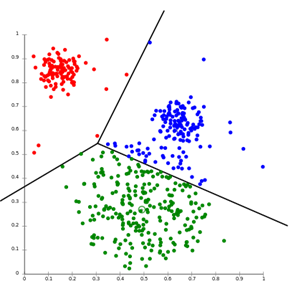
### 3.2.4 Processing Algorithm

Numerous machine learning or clustering algorithms were researched to be used in processing the user data. As the data-sets for each item were quite small, but a large number of sets would need to be processed, the data mining algorithms needed to be suitable. The initial algorithms tested were K-Means and EM.

##### K-Means

K-Means clustering is a vector quantization method. The name was first used in 1967 (MacQueen, 1967), though similar algorithms were proposed in numerous forms before then. This algorithm maps the data as a vector and partitions the data into clusters. This is quite a simple algorithm that was selected due to its accuracy when working with small data-sets. The algorithm works as follows:

1. Each item is mapped as a vector on a plane. The algorithm will insert k nodes at random points on the plane, with k being the number of clusters required.
2. These nodes will then calculate the distance to each vector. Each vector will then be assigned to its nearest node.
3. The node will then relocate to the centroid, or average position, of each of the vectors it has been assigned.
4. Steps 2 and 3 will repeat until the nodes no longer move. This is described as *Convergence* having been reached.

  
Figure 3.1: K-Means separating data into 3 clusters.  
By Chire - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=17085714

Data would be mapped using the time the address was visited in microseconds per day. This would be processed into a number of clusters that could be set by the user. Each of these clusters would be presented to the user for them to select from.

This method was dismissed for a number of reasons. The number of clusters could intimidate new users with its high quantity of schedule items. Many of the items would be very close to each other also, causing duplicates. The option of rounding the time to a certain value would cause even further duplication. Finally, when tested versus a simple program that would simply find the average time, the value did not differ greatly, though the processing time was much larger.

##### Expectation-Maximisation (EM)

EM is an iterative unsupervised clustering method. It uses iteration and log-likelihood to cluster values. Expected Maximisation is an algorithm developed in 1996, (McLachlan & Krishnan, 1996) though it has been noted by the authors as having been informally proposed in other literature. If used, this algorithm would also cluster the data and present the results for the user to select from.

EM iteration works by alternating between:

* An expectation (E) step, which creates a function for the expected log-likelihood using the current estimate of the model parameters.
* A maximisation (M) step, which maximises the expected log-likelihood function to get a new estimate of the model parameters.

These steps are repeated until convergence is reached.

This method was dismissed for use in the application for reasons similar to K-Means. The method of evaluating the log likelihood had a much larger processing time when compared to K-Means and provided similar results. Processing the data took far too long a time and the added complexity of the algorithm led to a greater possibility of failure and human error.

##### Simplified Centroid (SC)

Simplified Centroid was the algorithm that was eventually used. This was a very simple algorithm that was created for this project and would compute the average time for each item, similar to a single iteration of K-Means. In addition the “skew” of the results was checked. The skew was determined by comparing the result to the maximum and minimum values. If the result was greatly skewed, the algorithm would run again with the outlying results removed. Sample code is shown below. (PUT SAMPLE CODE AT END OR HERE?) (It’s at the end for now)

Though this was less accurate than the previous two algorithms, the time taken was reduced greatly. This enabled the extension to process the information quickly for the user. Although the efficiency of all 3 algorithms can be said to be *O(n),* k-means and EM run at efficiency *O(nkt),* and SC runs at *O(nt)*. Where ***n*** is the number of elements, ***k*** is the number of clusters, and ***t*** is the number of iterations. (Davenport, 2014) In addition, SC would rarely iterate more than twice, whereas k-means and EM would frequently iterate many more times.

The reduction in accuracy was negligible when compared to the increase in processing speed. As the processing worked with microseconds, precise results were unnecessary. When factoring for further time rounding as set by the user, the difference in accuracy between algorithm results could be overlooked. This algorithm would also present only one result, hopefully providing a much less intimidating result for the User.

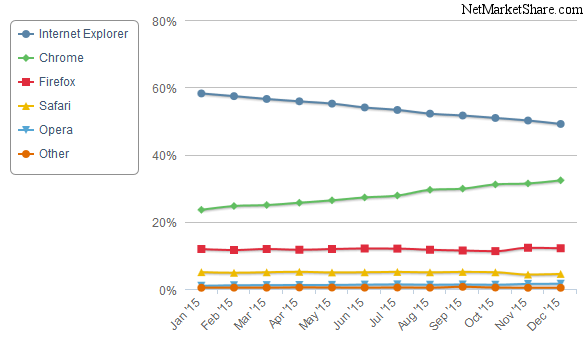
### 3.2.4 Commercial Viability

Will I do this section?

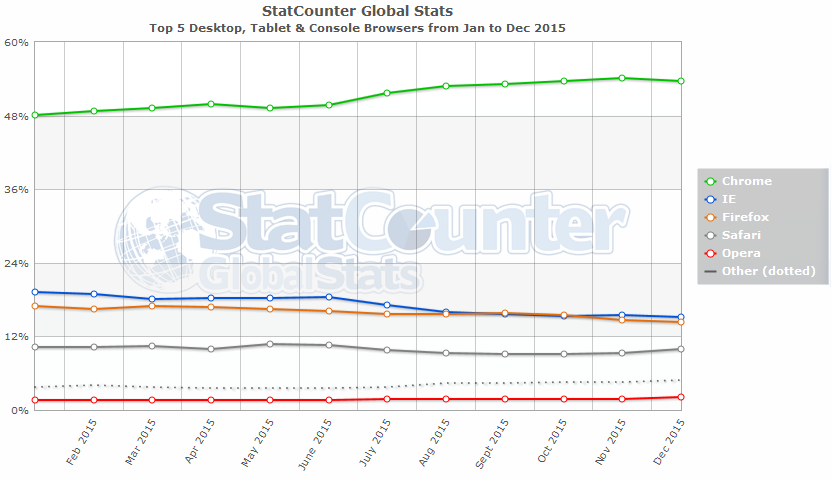
# 3.3 Software

### 3.3.1 Environment

From the initial stages the idea of a Web Browser Extension was decided. With the intention of allowing the extension to be as widely used as possible, the most popular web browser needed to be determined. Research into the usage of web browsers determined that the most commonly used web browser was Google Chrome, followed by Microsoft’s Internet Explorer and Mozilla Firefox. (StatCounter Global Stats, 2016). This measured the traffic of over 3 million websites and recorded data based upon page views. Another source ranked Internet Explorer as the most commonly used, followed by Google Chrome (NetMarketShare, 2016), determined by unique daily users. This method also weighted records by average country traffic, meaning the recorded traffic from one country would be weighted to relate more closely to national internet traffic for that country.



#### Fig. 3.2 - Diagram of Browser usage from NetMarketShare.com for the year 2015



#### Fig. 3.3 - Diagram of Browser usage from gs.StatCounter.com for the year 2015

The information above was assessed and Google Chrome was decided upon for creating the extension. The use of Google Chrome had numerous other benefits, such as greater portability across different operating systems and a more simplified approach to installing extensions. The release of Windows 10 with the new Edge web browser also meant a more limited life-span for Internet Explorer. The browser can also be seen to be already declining in both of the graphs above. The Google Store would also be beneficial in the distribution of the extension for testing.

### 3.3.2 Languages

After the selection of the project type and environment, the means of creating the application were determined. Google Chrome Extensions were primarily written in HTML, CSS, and JavaScript. Although HTML and CSS had already been briefly covered in previous modules throughout the course, JavaScript had not. This required the learning of JavaScript in order to create the extension.

Numerous resources were consulted in the early stages of development such as *JavaScript and JQuery: Interactive Front-End Web Development* (Duckett, et al., 2014) and online resources were consulted in order to properly learn the API of both JavaScript, and Google Chrome. Though the requirement of learning a new programming language slowed the initial stages of the project, progression improved throughout the course of the implementation.

As mentioned previously, initial designs included an external server, using JavaServer Pages (JSP) or Active Server Pages (ASP), which would process information gathered by the application. This feature was reconsidered and eventually replaced with internal processing using JavaScript. This is further detailed in section 3.2.3 above.

### 3.3.3 Storage

Storing the information used by the extension was greatly facilitated through the use of the Google Chrome API. Though other storage methods were possible through HTML5, the Chrome API allowed for further options for storage such as synchronised storage. Synchronised storage allowed for information to be stored on the User’s profile, to allow settings and other information to be accessed across multiple devices. This was a major benefit of using Google Chrome for my application. All storage access was completed using asynchronous functions. This meant that much of the functionality needed to accommodate for this. This will be detailed further in the Problems Faced section in Chapter 5 Testing and Evaluation of this report.

For storing many of the lists and static components in the extension, JSON was used. JSON stands for JavaScript Object Notation, and is a “lightweight data-interchange format” based upon a subset of *JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999* (Standard, ECMA, 2011)*.* The formatis similar to that of JavaScript Objects, and this means that the format is easy to read and write for both humans and computers. Information could be stored as an array or object, and could be easily accessed through the extension.

### 3.3.4 System Architecture

An important part of the design was usability. A large factor with usability is the efficiency of the program. This is because users will often abandon a service that has overly lengthy loading times. Much of the design of the extension incorporated many factors such as binary searches and proper utilisation of asynchronous functions. Asynchronous functions are functions that run on another thread and this allows them to run concurrently, or in parallel to other functions. This meant that functions can be run in the background and allows for faster loading of important items. Along with proper search and sorting techniques this ensured that the program would load all necessary information in as short a time as possible.

Another key ideal when designing the software was that of maintainability and extensibility. The code was designed to be easy to read and understand, as well as reducing the amount of “hard coded” values and code duplication. This meant that extending the project to include additional features, or perform maintenance such as updates or bug fixes would be much easier, even if performed by somebody new to working on the project.

# 3.4 Interface

As mentioned above, the usability of the extension was a major focus of the extension. Much of this usability relied on the User Interface. As discovered in the research, many well-designed interfaces rely on intuitive understanding and minimal effort. A large amount of time and effort was put into providing a clear interface that could be easily understood. Extensive user testing was performed to decide upon seemingly small items such as button names and colours. This was all to ensure that the interface could be easily and intuitively understood as described in the resources consulted during research, (Norman, 2013) and (Krug, 2014).

An important part of the usability for a program interface is the aesthetics and design. With this in mind the interface was designed with a minimalistic approach. Buttons were made large and of a different colour. The background was made white so as to provide contrast to and highlight the coloured features. Much of this functionality was made possible through the use of CSS. The use of classes allowed standardisation and uniformity across the extension. This meant that even new pages would have a familiarity and level of consistency.

This focus on uniformity and consistency was a marked improvement on the usability of the extension. Initial designs had flaws that had been overlooked due to the disparity in computer usage between the developer and initial testers, and the intended user base. This is detailed further in Chapter 5 Testing and Evaluation of this report.

4. Implementation

# 4.1 Introduction

The implementation of the information and techniques learned through the research the previous semester was done with an extension for Google Chrome. This extension, titled “Jenkins Web Assistant” was intended to facilitate the User’s use of the computer through three main components.

The first is a scheduler, allowing users to set notifications for certain web pages at certain times. This also featured an algorithm for automatically discerning schedule entries.

The second component is a Site Recommendation system. This would process the current page and provide recommendations for similar websites.

The third component is a Profiler that would categorise each of the tabs opened by the user at certain times and use this information to help form a User Profile.

The final section will describe the User Settings and Options Pages, as well as the popup notifications.

This chapter will outline how each of the components of the project solution was implemented. This will detail the work completed for each component and contain small code fragments illustrating the solutions and aiding the explanation of the system. There will also be a short section on each component evaluating the work as well as providing areas for expansion. The final section will deal with the problems faced during the implementation of the project and the solutions found.

# 4.2 Scheduler

### 4.2.1 Feature Description

The scheduler of the extension was a key component of the implementation and took the longest amount of development time. A large factor in the development time for the scheduler was the design of the processing algorithm, of which the design and algorithm comparison has been detailed at length in section 3.2.3. As this was the first component developed, additional time was spent learning the JavaScript Language and restructuring or rewriting existing code.

The primary feature of this component was the history processing. This was the primary method by which the User’s preferences would be mined. This information would be displayed to the user for feedback. The secondary feature was the popup notifications. It was necessary for these notifications to facilitate the user’s browsing experience. Details for these features can be found in the following sections.

### 4.2.2 Explanation and Code Samples

The history processing is performed by the background page (background.js) and will process the user’s history based upon the user’s settings.

##### Start-up Functions

A function is called upon opening the browser that will call a number of functions to perform acts such as checking the schedule and processing the history. This will also initiate the timer to check the schedule regularly. All actions are performed only permission has been given in the options page.

|  |
| --- |
| //Calls clearHistory when the extension is loaded  chrome**.**runtime**.**onStartup**.**addListener**(*function*(){**  //retrieve the data from the JSON file on opening  readFromFile**(**recListFile**);**  //This puts it into the recObject object for use by the recommender  //Clear last session history if setting is enabled  clearHistory**();**  //Calls the other functions too.  chrome**.**storage**.**sync**.**get**({**  checkFrequency**:** 5**,**  history**:** ***true*,**  organiser**:** ***true*,**  startupProcess**:** ***true*,**  tempNotify**:** ***false***  **},** ***function*** **(**items**)** **{**  //First check schedule  checkSchedule**();**  //Convert from old format to new format  //This is necessary because the older versions of the schedule used  // a string format rather than an object format  convertFromStringFormat**();**  //Check recommendations list  sortRecommendations**();**  //if tempnotify was selected, reset it  // temp notify is the option to temporarily prevent notifications  // and can be selected in the popup  ***if*(**items**.**tempNotify**)** **{**  chrome**.**storage**.**sync**.**set**({**  tempNotify**:** ***false*,**  notification**:** ***true***  **});**  **}**  ***if*(**items**.**startupProcess **&&** items**.**history **&&** items**.**organiser**)** **{**  //If set in permissions, run organiser on startup  processHistory**();**  **}**  //How often in microseconds it will check  ***var*** checkTime **=** items**.**checkFrequency **\*** microsecondsPerMinute**;**  //Also initiates loop for checking schedule  setInterval**(**checkSchedule**,** checkTime**);**  **});**    **});** |

#### Figure 4.1: Functions called on Browser start-up.

##### Checking Schedule Times

The schedule is checked by examining whether the time saved for each schedule item has fallen between the current time and the estimated time of the last check. This means that the system will examine the time prior to the start-up time when the browser is first opened. This allows the user to be notified even when the extension is first opened, but prevents the possibly overwhelming amount that may be caused if the lastChecked object is saved.

|  |
| --- |
| ***function*** checkSchedule**()** **{**  //Get the schedule  chrome**.**storage**.**sync**.**get**({**  checkFrequency**:** 5**,**  autoNotifications**:** ***false*,**  schedule**:** **[]**  **},** ***function*(**items**)** **{**  //Get the current time  ***var*** now **=** ***new*** Date**();**  ***var*** microsecondsSinceLastInterval **=** microsecondsPerMinute **\*** items**.**checkFrequency**;**  ***var*** lastCheckedVal **=** **(*new*** Date**).**getTime**()** **-** microsecondsSinceLastInterval**;**  ***var*** lastChecked **=** ***new*** Date**(**lastCheckedVal**);**  //Check time with each element of list  ***var*** scheduleArray **=** items**.**schedule**;**  ***for*(*var*** x **=** 0**;** x **<** scheduleArray**.**length**;** x**++)** **{**  ***var*** currentItem **=** scheduleArray**[**x**];**  ***var*** time **=** ***new*** Date**(**currentItem**.**time**);**  //Get hours + minutes from schedule item  ***var*** itemHour **=** time**.**getHours**();**  ***var*** itemMinute **=** time**.**getMinutes**();**  //Check values versus times  ***if*(**itemHour **<=** now**.**getHours**()** **&&** itemHour **>=** lastChecked**.**getHours**()){**  //If hours correct check minutes  ***if*(**itemMinute **<=** now**.**getMinutes**()** **&&** itemMinute **>** lastChecked**.**getMinutes**()){**  //If the time has passed send notification  ***if*(**items**.**autoNotifications **||** currentItem**.**approved**)** **{**  ***var*** destination **=** currentItem**.**url**;**  destination **=** destination**.**toLowerCase**();**  //set colour based upon object type  ***var*** notificationColour **=** 0**;** //default  //Set the picture as green if it's approved  ***if*(**currentItem**.**approved**)** **{**  notificationColour **=** GREENSUIT**;**  **}**  //Set as blue if it's an automatic creation  ***else*** **{**  notificationColour **=** BLUESUIT**;**  **}**  //Do with a function now to prevent display error  checkTabs**(**time**,** destination**,** notificationColour**);**  **}**  **}**  **}**  **};**  **});**  **}** |

#### Figure 4.2: Checking the schedule

##### Checking Current Tabs

The checkTabs() function will scan through each of the tabs the user has opened, and if the URL is found amongst them, will switch the user to that tab if the notification is clicked. If the URL is not found, it will be opened in a new tab if the notification is clicked. If the User has the URL open in their current tab then no action will be performed.

|  |
| --- |
| ***function*** checkTabs**(**time**,** destination**,** notificationColour**)** **{**  ***var*** itemHour **=** time**.**getHours**();**  ***var*** itemMinute **=** time**.**getMinutes**();**  //Get all tabs open in the window  chrome**.**tabs**.**query**({},** ***function*** **(**result**)** **{**  ***var*** tabToOpen**;**  ***var*** current **=** ***false*;**  //If it found no results then the page is not open  ***if*(**result **==** null**)** **{**  console**.**log**(**destination **+** " not found in tabs. Function will open Tab."**);**  //Appropriate function...  tabToOpen **=** null**;**  current **=** ***true*;**  **}**  //Check whether the website is already open.  ***for*** **(*var*** index **=** 0**;** index **<** result**.**length **&&** **!**current**;** index**++)** **{**  //For ease of reading  ***var*** item **=** result**[**index**];**  //If this tab contains url to be opened  ***if*(**item**.**url**.**includes**(**destination**))** **{**  //if it's the current window+tab  ***if*(**item**.**currentWindow **&&** item**.**active**)** **{**  //Window is open. Ignore?  //Notify?  current **=** ***true*;**//?  ***return*;**  **}**  //If tabToOpen has no value  ***if*(**tabToOpen **==** null**)** **{**  tabToOpen **=** item**;**  **}**  **}**  **}**  //If tab was found in loop above  ***if*(**tabToOpen **!=** null**)** **{**  //To prevent 12:7 etc.  ***var*** itemMinuteText **=** "" **+** itemMinute**;**  ***if*** **(**itemMinute **<** 10**)**  itemMinuteText **=** "0" **+** itemMinuteText**;**  ***var*** title **=** itemHour **+** ":" **+** itemMinuteText **+** " reminder"**;**  //var title = "Website reminder";  ***var*** textContent **=** "Click here to open tab containing " **+** destination**;**  //Switch to that tab when notification clicked  notificationFunction**(**title**,** textContent**,** ***function*()** **{**  //Switch to that tab  chrome**.**tabs**.**update**(**tabToOpen**.**id**,** **{** active**:** ***true*** **});**  **},** null**,** notificationColour**);**  **}**  ***else*** **{** //Tab not found to be open  //Open url in a new tab  //Make sure the url works  destination **=** makeURL**(**destination**);**  //To prevent 12:7 etc.  ***var*** itemMinuteText **=** "" **+** itemMinute**;**  ***if*** **(**itemMinute **<** 10**)**  itemMinuteText **=** "0" **+** itemMinuteText**;**  ***var*** title **=** itemHour **+** ":" **+** itemMinuteText **+** " reminder"**;**  ***var*** textContent **=** "Click here to open " **+** destination **+** " in a new tab."**;**  //Show notification  notificationURL**(**title**,** textContent**,** destination**,** notificationColour**);**  **}**  **});**  **}** |

#### Figure 4.3: Checking whether the URL is already open in another tab.

##### Processing Web History

The processor accesses the User’s web history, if the option is selected, and processes the web pages contained within. The function iterates through the history results and counts each occurrence of a website. In order to reduce duplication, results ending with a query or “?” will have those removed.

History objects that with a count below the limit set by the user are simplified to just the domain and this is added to the list. The function iterates through the list again and any object counts that are below the threshold are removed.

The remaining items are then compared to the user ignore list and the internal blacklist. If any URL contains an item from these lists then the history object will be removed. The remaining items are then processed in order to find the appropriate time.

|  |
| --- |
| ***function*** processHistory**()** **{**  //array for individual sites  ***var*** commonSites **=** **[];**  //Get all settings and schedule  chrome**.**storage**.**sync**.**get**({**  ignoreQuery**:** ***true*,**  history**:** ***true*,**  organiser**:** ***true*,**  recommender**:** ***true*,**  visitThreshold**:** 3**,**  pageVisitThreshold**:** 9**,**  typedWeight**:** 2**,**  timeThreshold**:** 28**,**  ignoreList**:** ""**,**  clearhistory**:** ***false*,**  schedule**:** **[]** //For adding on changes  **},** ***function*(**items**)** **{**  //Default ignore list for stopping checking private details  // get this from recObject  ***var*** defaultBlacklist **=** recObject**.**defaultBlacklist**;**  //Stop if we don't have permission  ***if*(**items**.**history **!=** ***true*){**  ***return*;**  //This will end the function prematurely  **}** ***else*** ***if*(**items**.**organiser **!=** ***true*)** **{**  ***return*;**  **}**  ***var*** microsecondsPerDay **=** 1000 **\*** 60 **\*** 60 **\*** 24**;**  //Multiply 1 day by the amount of days set by user.  ***var*** historyCutoff **=** **(*new*** Date**).**getTime**()** **-** **(**microsecondsPerDay **\*** items**.**timeThreshold**);**  ***if*** **(**items**.**timeThreshold **<=** 0**)**  historyCutoff **=** 0**;**  // Access history and process results  chrome**.**history**.**search**({**  'text'**:** ''**,**  'startTime'**:** historyCutoff  **},**  ***function*(**historyItems**)** **{**  //For each item check if it reaches the threshold  historyItems**.**forEach**(*function*(**item**,** index**,** array**)** **{**  ***var*** visits **=** item**.**visitCount**;**  //Add weighted typedCount but remove one from weight  // to account for initial recording of visit  visits **+=** item**.**typedCount **\*** **(**items**.**typedWeight **-** 1**);**  //If it's the id of the extension then it ignores it  ***if*** **(**item**.**url**.**includes**(**myid**))** **{**  ***return*;**  **}**  ***else*** ***if*(**items**.**ignoreQuery **&&** item**.**url**.**indexOf**(**"#"**)** **>** **-**1**)** **{**  ***return*;** //ignore any query or hash entries  **}**  //For single pages with many visits  ***else*** ***if*(**visits **>=** items**.**pageVisitThreshold**)** **{**  item**.**visitCount **=** visits**;**  //remove # from end  ***var*** urltemp **=** item**.**url**;**  ***var*** hashIndex **=** urltemp**.**indexOf**(**'#'**);**  ***if*(**hashIndex **>** **-**1**)** **{**  item**.**url **=** item**.**url**.**substring**(**0**,** hashIndex**);**  **}**  //remove ? from end  ***var*** qmarkIndex **=** urltemp**.**indexOf**(**'?'**);** //Easier than urltemp.endsWith("?")  ***if*(**qmarkIndex **==** urltemp**.**length**-**1**)** **{**  item**.**url **=** item**.**url**.**substring**(**0**,** qmarkIndex**);**  **}**  //Set object as a single page  item**.**singlePage **=** ***true*;**  **}**  //else check just the domain  ***else*** **{**  item**.**url **=** getHostname**(**item**.**url**);**  //and set as NOT a single page  item**.**singlePage **=** ***false*;**  **}**  ***var*** found **=** ***false*;**  //Check if the site/domain already exists in the list  //This is O(X^2) so maybe look into better method  ***for*(*var*** i **=** 0**;** found **==** ***false*** **&&** i **<** commonSites**.**length**;** i**++)** **{**  ***if*(**commonSites**[**i**].**url **==** item**.**url**)** **{**  commonSites**[**i**].**visitCount **+=** visits**;**  found **=** ***true*;**  **}**  **}**  //if not found add to list  ***if*(**found **==** ***false*)** **{**  commonSites**.**push**(**item**);**  **}**  **});**  //Remove elements of list below threshold or containing blacklist  ***for*(*var*** index **=** 0**;** index **<** commonSites**.**length**;** index**++)** **{**  ***if*(**commonSites**[**index**].**visitCount **<** items**.**visitThreshold**)** **{**  ***var*** removed **=** commonSites**.**splice**(**index**,** 1**);**  //remove from index because array is shorter  index **-=** removed**.**length**;**  **}**  ***else*** **{**  //Check user blacklist  ***var*** blacklist **=** items**.**ignoreList**.**split**(**","**);**  commonSites **=** checkBlacklist**(**commonSites**,** blacklist**);**  //Check default blacklist (Defined at top of page)  commonSites **=** checkBlacklist**(**commonSites**,** defaultBlacklist**);**  **}**  **}**  //Process this list for patterns  commonSites**.**forEach**(*function*(**item**,** index**,** array**)** **{**  //Single pages are treated differently to domains  ***if*(**item**.**singlePage**)** **{**  processSinglePage**(**item**.**url**);**  **}**  ***else*** **{**  processDomain**(**item**.**url**);**  **}**  **});**  **});**  **});**  **}** |

#### Figure 4.4: Processing the History Objects

##### Time Processing

Once the History Objects have been processed, each object is processed using the Simplified Centroid Algorithm. This algorithm is detailed in 3.2.4 Processing Algorithm section of this report.

|  |
| --- |
| //Find the centroid  historyArray**.**forEach**(*function*** **(**item**,** index**,** array**)** **{**  //For simplicity it only counts hours  ***var*** modifiedTime **=** 0**;**  ***if*(**type **==** "domain"**)** **{**  modifiedTime **=** item**.**lastVisitTime **-** NZModification**;**  **}**  ***else*** ***if*(**type **==** "single"**)** **{**  modifiedTime **=** item**.**visitTime **-** NZModification**;**  **}**  ***else*** **{**  //error  ***return*;**  **}**  ***var*** time **=** modifiedTime **%** microsecondsPerDay**;**  //Make sure the new hours aren't different to the original hours  //This is mostly to deal with Daylight Savings  ***if*(*new*** Date**(**time**).**getHours**()** **!=** ***new*** Date**(**modifiedTime**).**getHours**())** **{**  time **+=** microsecondsPerHour**;**  time **=** time **%** microsecondsPerDay**;**  **}**  //Set new min and max if changed  ***if*(**time **<** minTime**)** **{**  minTime **=** time**;**  **}** ***else*** ***if*** **(**time **>** maxTime**)** **{**  maxTime **=** time**;**  **}**  **});**  //Get average visit time  ***var*** averageTime **=** timeSum **/** itemCount**;**  //Round to the preset number  averageTime **-=** **(**averageTime **%** **(**items**.**timeRounding **\*** microsecondsPerMinute**));** |

#### Figure 4.5: Finding the Centroid

|  |
| --- |
| //Check if it needs to be broken up further  // if the min and max are more than X hours apart and the average is close to the middle  // then we'll split it up further.  ***var*** difference **=** **(**maxTime **-** minTime**)** **/** microsecondsPerHour**;**  ***if*(**difference **>** items**.**timeDeviation**)** **{**  ***var*** lower **=** **((**averageTime **-** minTime**)** **/** microsecondsPerHour**)** **/** difference**;**  ***var*** upper **=** **((**maxTime **-** averageTime**)** **/** microsecondsPerHour**)** **/** difference**;**  ***var*** positiveSkew **=** lower **/** upper**;**  ***var*** negativeSkew **=** upper **/** lower**;**  ***if*(**positiveSkew **>** items**.**skewnessThreshold**)** **{**  //If skewed positively, set the average to be the new upperThreshold  ***var*** tempDate **=** ***new*** Date**(**averageTime **+** NZModification**);**  ***var*** newUpperThreshold **=** tempDate**.**getHours**()** **+** ":" **+** tempDate**.**getMinutes**();**  items**.**trackBefore **=** newUpperThreshold**;**  //Then run the process again with the new limit  processTime**(**url**,** items**,** historyArray**,** type**);**  **}**  ***else*** ***if*(**negativeSkew **>** items**.**skewnessThreshold**)** **{**  //If skewed negatively, set the average to be the new lower limit  ***var*** tempDate **=** ***new*** Date**(**averageTime **+** NZModification**);**  ***var*** newLowerThreshold **=** tempDate**.**getHours**()** **+** ":" **+** tempDate**.**getMinutes**();**  items**.**trackAfter **=** newLowerThreshold**;**  //Then run the process again with the new limit  processTime**(**url**,** items**,** historyArray**,** type**);**  **}**  ***else*** **{**  ***var*** scheduleTime **=** ***new*** Date**(**averageTime **+** NZModification**);**  //Important to add back on the change  addToSchedule**(**url**,** scheduleTime**);**  **}**  **}**  ***else*** **{**  ***var*** scheduleTime **=** ***new*** Date**(**averageTime **+** NZModification**);**  //Important to add back on the change  addToSchedule**(**url**,** scheduleTime**);**  **}** |

#### Figure 4.6: Checking whether the result is skewed

4.2.3 Results (Or should I move this to Evaluation?)

# 4.3 Site Recommendations

### 4.3.1 Feature Description

The second part of the extension to be written was the Recommender System. This system would recommend websites to the user based upon their browsing. This was designed to show in the popup window once the icon was clicked. This functioned by categorising a page by processing the title and URL of a page and searching for keywords. This would then be used to offer other websites that may be in the same category and therefore may be similar to the current page.

### 4.3.2 Explanation and Code Samples

##### Reading from the JSON File (Rephrase?)

The information for the Recommendations was stored in a JSON file contained within the extension folder. This allowed the file to be more easily edited and maintained in further versions of the extension. Upon browser start-up, this file is read by the extension and saved as an object. This information is accessed through the background page (background.js) and saved as the global RecObject, which stands for Recommendation Object. This data can then be easily accessed by the functions within the background page.

|  |
| --- |
| //Sends a request to read the file.  ***function*** readFromFile**(**textFile**)** **{**  ***var*** xhr **=** ***new*** XMLHttpRequest**;**  xhr**.**open**(**'GET'**,** textFile**);**  //On reading the file etc. calls the function show  xhr**.**onload **=** show**;**  xhr**.**send**();**  **}**  //Puts the information into an object that can be used by the other functions  ***function*** show**()** **{**  //Put all of the text into a string  ***var*** allTheText **=** ***this*.**response**;**  //make an object (Already defined globally) of the parsed string  recObject **=** JSON**.**parse**(**allTheText**);**  **}** |

#### Figure 4. : Reading from the recommendations.JSON file.

This information is processed on start-up and checked with the user information saved in the local storage. Any changes to the recommendations from the file, such as from version updates, are added to the data saved in the local storage. This is similarly performed for the keywords array. This data is then used to categorise the web pages and present recommendations to the user.

##### Website Categorisation

Recommendations for the user are generated by clicking on the extension icon at the top of the screen. Once the popup page is loaded, it will compare the URL and Title of the current window and tab with the recommendations and keywords arrays and try to categorise the current web page.

|  |
| --- |
| //Compares the elements of the tab with the tags  ***function*** categoriseTab**(**tab**,** items**,** previousCategory**)** **{**  //For ease of reading and writing etc.  ***var*** recommendations **=** items**.**recommendations**;**  ***var*** tags **=** items**.**automaticClassification**;**  ***var*** url **=** tab**.**url**;**  ***var*** title **=** tab**.**title**;**  ***var*** categoryIndex **=** **-**1  //Find the category containing the url starting from just after the previous index  // (Usually 0 or (-1)+1)  categoryIndex **=** searchForUrlCategory**(**url**,** recommendations**,** previousCategory**);**    //Returns -1 if not found  ***if*(**categoryIndex **<** 0**)** **{**  //Try and automatically categorise using url  categoryIndex **=** guessCategory**(**url**,** tags**,** previousCategory**);**  **}**  //If still not found  ***if*(**categoryIndex **<** 0**)** **{**  //can't find anything in url. Try title  categoryIndex **=** guessCategory**(**title**,** tags**,** previousCategory**);**  **}**  //If it's playing sound it might be music???  ***if*(**categoryIndex **<** 0 **&&** tab**.**audible**)** **{**  categoryIndex **=** MUSIC\_CATEGORY**;**  **}**  //If none of these work try meta tags  ***if*(**categoryIndex **<** 0**)** **{**  chrome**.**tabs**.**sendMessage**(**tab**.**id**,** **{**text**:** 'get\_meta\_tags'**},** checkMetaTags**);**  **}**  ***return*** categoryIndex**;**  **}** |

#### Figure 4.: Categorising the Recommendation

|  |
| --- |
| //Functionality functions. Used in previous functions simply for ease of reading/modification  ***function*** searchForUrlCategory**(**url**,** recommendationsArray**,** previousCategory**)** **{**  //previousCategory is for trying again later for a different category.  //Search array for url  ***for*(*var*** returnIndex **=** previousCategory **+** 1**;** returnIndex **<** recommendationsArray**.**length**;** returnIndex**++)** **{**  //search each element of each category.  // (Might switch this for a binary search later)  ***for*(*var*** urlIndex **=** 1**;** urlIndex **<** recommendationsArray**[**returnIndex**].**length**;** urlIndex**++)** **{**  ***var*** recommendedUrl **=** recommendationsArray**[**returnIndex**][**urlIndex**].**url**;**  ***if*(**url**.**includes**(**recommendedUrl**))** **{**  ***return*** returnIndex**;**  **}**  **}**  **}**  //If URL is not found  ***return*** **-**1**;**  **}** |

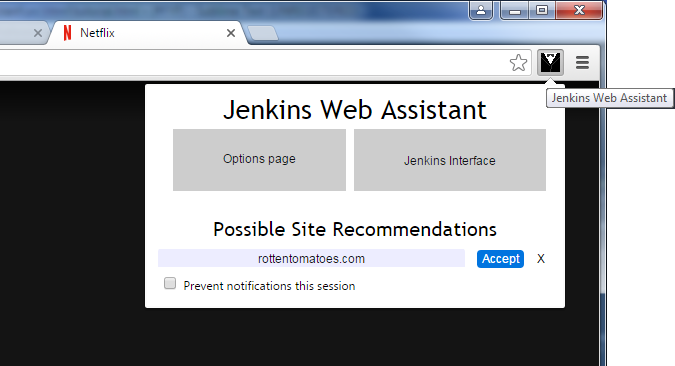
#### Figure 4.: Searching for site among existing recommendations

|  |
| --- |
| //Takes words from the url or title and tries to guess the category  ***function*** guessCategory**(**tabInfo**,** tagArray**,** previousCategory**)** **{**  //previousCategory is for trying again later for a different category.  //make lower case  tabInfo **=** tabInfo**.**toLowerCase**();**    //search each array, previousCategorying AFTER the previous  ***for*(*var*** returnIndex **=** previousCategory **+** 1**;** returnIndex **<** tagArray**.**length**;**  returnIndex**++)** **{**  //check each element or tag  ***for*(*var*** tagIndex **=** 0**;** tagIndex **<** tagArray**[**returnIndex**].**length**;** tagIndex**++)** **{**  ***var*** tag **=** tagArray**[**returnIndex**][**tagIndex**];**  //If the tabInfo contains the tags  ***if*(**tabInfo**.**includes**(**tag**))** **{**  //return the index of the category  ***return*** returnIndex**;**  **}**  **}**  **}**  //if nothing is found return default  ***return*** **-**1**;**  **}** |

#### Figure 4.: Processing tab using tags.

##### Displaying Recommendation

If the web page is successfully assigned to a category, the popup page will then display a table showing a recommendation. This recommendation can be accepted by clicking the Accept button, or dismissed by clicking the X button. If the category selected contains no valid recommendations, the system will attempt to re-categorise the web page. If the system is unable to categorise the web page then the recommendations table will not be shown.



#### Figure 4.: Sample recommendation display from Tutorial page.

### 4.3.3 Results (Or should I move this to Evaluation?)

The recommender system works. As decided in the design brief, the system checks the tab information and prompts the user to accept or decline the recommendation. Though the current implementation does not utilise the accepted recommendations, possible expansion goals are for the extension to utilise this feedback to form a more accurate user profile.

Another possible expansion goal would be to alter the order for presenting the recommendations and methods of categorising the web pages. The recommendations were presented in alphabetical order but future prospects would be to order the recommendations based upon both user feedback and, allowing for commercial viability, and customer priority.

The current method of categorising works similarly in alphabetical order. Room for expansion of this feature would be to form a heuristic algorithm based upon certain keywords and phrases in order to more accurately categorise each web page.

# 4.4 Profiler?? Options?

### 4.4.1 Feature Description

The profiler is the final component of the extension. This monitors the user’s tab patterns and helps form a profile based upon these patterns. This is done based upon the tab categories as well as retaining the information anonymously. This was performed similarly to the Recommender system in that it would process the title and URL for certain keywords in order to classify the tabs.

### 4.4.2 Explanation and Code Samples

##### Tab Categorisation

Tabs are categorised using methods similar to those used by the Recommendations component. Each tab is categorised and they are all stored in an array.

|  |
| --- |
| //Categorises all open tabs  ***function*** checkTabs**()** **{**  //get necessary variables etc.  chrome**.**storage**.**local**.**get**({**  recommendations**:** **[],**  automaticClassification**:** **[],**  patternLastChecked**:** 0**,**  browsingPatterns**:** **[]**  **},** ***function*(**items**)** **{**  ***var*** rightNow **=** Date**.**now**();**  //Only check it it has been some time since the previous check  ***if*((**rightNow **-** items**.**patternLastChecked**)** **>** **(**patternCheckFrequency **\*** microsecondsPerMinute**))** **{**  //Change when last checked  items**.**patternLastChecked **=** rightNow**;**  //get ALL tabs  chrome**.**tabs**.**query**({** **},** ***function*** **(**result**)** **{**  //to save results  ***var*** tabArray **=** **[];**  //Process tabs and record pattern (Alphabetically?)  ***for*(*var*** tabIndex **=** 0**;** tabIndex **<** result**.**length**;** tabIndex**++)** **{**  //For ease of reading/writing.thinking etc.  ***var*** currentTab **=** result**[**tabIndex**];**  ***var*** categoryIndex **=** **-**1**;**  //Initialising as -1 prevents errors when going to the same or lower category  //Categorise using the same function as the recommendations  categoryIndex **=** categoriseTab**(**currentTab**,** items**,** categoryIndex**);**  //Add elements to array  tabArray**.**push**(**categoryIndex**);**  **}**  //Sort the array  tabArray**.**sort**();**  //Add to patterns array as an object  ***var*** pattern **=** ***new*** Object**();**  pattern**.**pattern **=** tabArray**;**  pattern**.**time **=** getNow**();**  pattern**.**count **=** 1**;**  //Check existing and save  savePattern**(**pattern**,** items**);**  **});**  **}**  **});**  **}** |

#### Figure 4.: Categorising all currently opened tabs.

This pattern is then compared to the existingPatterns array and if the pattern already exists, the count is incremented. If the pattern does not already exist then it will be added to the array as a new pattern.

|  |
| --- |
| //Saves the pattern to the array  ***function*** savePattern**(**tabs**,** items**)** **{**  //For ease of reading/writing etc.  ***var*** existingPatterns **=** items**.**browsingPatterns**;**  ***var*** found **=** ***false*;**  //Simplify the pattern into just  ***var*** simplified **=** removeDuplicates**(**tabs**.**pattern**);**  //Scan through existing patterns to see if it exists  ***for*(*var*** patternIndex **=** 0**;** **!**found **&&** patternIndex **<** existingPatterns**.**length**;** patternIndex**++)** **{**  ***var*** currentObject **=** existingPatterns**[**patternIndex**];**  //See if patterns are equal OR if the pattern exists with different numbers of tabs  ***if*(**equalArrays**(**tabs**.**pattern**,** currentObject**.**pattern**)** **||** equalArrays**(**simplified**,** currentObject**.**pattern**))** **{**  currentObject**.**count **+=** 1**;**  found **=** ***true*;**  //Also change the average time  // Multiply the existing time by previous count,  // add on new time and divide by new count  currentObject**.**time **=** **((**currentObject**.**time **\*** **(**currentObject**.**count **-** 1**))** **+** getNow**())** **/** currentObject**.**count**;**  **}**  //else keep looping  **}**  //If not found, add to the array  ***if*(!**found**)** **{**  existingPatterns**.**push**(**tabs**);**  //Also save simplified version?  **}**  //Then save the changed existing patterns array  // and also the last checked value  chrome**.**storage**.**local**.**set**({**  browsingPatterns**:** existingPatterns**,**  patternLastChecked**:** items**.**patternLastChecked  **},** ***function*()** **{**  //Successfully saved  **})**  **}** |

#### Figure 4.: Adding the pattern to the array.

### 4.4.3 Results (Or should I move this to Evaluation?)

Although the component is able to create patterns of the User’s browsing, the system does not currently utilise this information in any way.

# 4.5 Options and Notifications (Interface?)

### 4.5.1 Feature Description

This section will detail the implementation of the User Options contained within the Options, Advanced Options, and Interface Pages. These options were necessary for allowing the user full customisability of the extension. Saving and retrieving user settings was facilitated greatly through the use of Google Chrome’s API. Synchronised storage allowed for options to be saved across multiple devices through the use of Google Profiles.

The notifications were used by a number of components and were an integral feature of the application. A large amount of effort was put into properly customising these notifications so that they would assist the user and would inconvenience them as little as possible.

User Settings were saved using Google Chrome’s Synchronised storage. In order to access the synchronised storage, the function would call upon the Google Chrome API. This called an asynchronous function that would retrieve the relevant data. This was used numerous times throughout the project.

### 4.5.2 Explanation and Code Samples

##### Retrieving Settings

Settings are retrieved by the system by calling the chrome.storage.sync.get() function. This calls an asynchronous function that returns an items object. This object contains the values specified in the initial query. If the object property cannot be found, it will use a default value that must be supplied. An example is shown below.

|  |
| --- |
| // Use default value of true for all and none for blacklist  chrome**.**storage**.**sync**.**get**({**  history**:** ***true*,**  notification**:** ***true*,**  organiser**:** ***true*,**  recommender**:** ***true*,**  visitThreshold**:** 3**,**  timeThreshold**:** 28**,**  ignoreList**:** ""**,**  clearhistory**:** ***false***  **},** ***function*(**items**)** **{**  //Function goes here  **});** |

#### Figure 4.: Retrieving saved settings

##### Options Pages

Upon loading one of the options pages, a function is called populating each of the page elements with the relevant information. This is retrieved as shown above, and is assigned to each element based upon the element ID.

User settings are saved in a similar fashion, where settings are retrieved from the page based on element ID. On page load, a trigger is assigned to the “Save” button and this will call the function to save the page options.

|  |
| --- |
| //set triggers  document**.**addEventListener**(**'DOMContentLoaded'**,** ***function*()** **{**  restore\_options**();**  document**.**getElementById**(**'save'**).**addEventListener**(**'click'**,** save\_options**);**  **});**  // Saves options to chrome.storage  ***function*** save\_options**()** **{**  //retrieve the settings from the page  //Permissions  ***var*** historyPermission **=** document**.**getElementById**(**'checkHistory'**).**checked**;**  ***var*** notifyPermission **=** document**.**getElementById**(**'checkNotificions'**).**checked**;**  ***var*** organiserPermission **=** document**.**getElementById**(**'checkOrganiser'**).**checked**;**  ***var*** recommendPermission **=** document**.**getElementById**(**'checkRecommendations'**).**checked**;**  //Processing  ***var*** visitSite **=** document**.**getElementById**(**'visitThresholdSite'**).**value**;**  ***if*** **(**visitSite **<** 1**)** //Default to 1 if they set it too low  visitSite **=** 1**;**  ***var*** timer **=** document**.**getElementById**(**'timeThreshold'**).**value**;**  ***if*** **(**timer **<** 0**)** //Default to 0 if they set it too low  timer **=** 0**;**  ***var*** ignored **=** document**.**getElementById**(**'blacklist'**).**value**;**  //History Clear  ***var*** clearhistory **=** document**.**getElementById**(**'checkClearHistory'**).**checked**;**  //Save the settings to memory  chrome**.**storage**.**sync**.**set**({**  history**:** historyPermission**,**  notification**:** notifyPermission**,**  recommender**:** recommendPermission**,**  organiser**:** organiserPermission**,**  visitThreshold**:** visitSite**,**  timeThreshold**:** timer**,**  ignoreList**:** ignored**,**  clearhistory**:** clearhistory**,**  **},** ***function*()** **{**  // Update status to let user know options were saved.  ***var*** status **=** document**.**getElementById**(**'status'**);**  status**.**textContent **=** 'Options saved.'**;**  setTimeout**(*function*()** **{**  status**.**textContent **=** ""**;**  status**.**appendChild**(**document**.**createElement**(**'br'**));**  **},** 750**);**  **});**  **}** |

#### Figure 4.: Setting button trigger and saving user settings.

##### Notifications

Notifications are called by each component through the background page (background.js). Each notification consists of a title, text, and type, as well as other parameters depending on the function. The type of the notification alters the colour of the icon displayed to the User.

#### D:\Users\Mike\DropboxRPG\Dropbox\#FYP\Jenkins Web Assistant\images\128.pngD:\Users\Mike\DropboxRPG\Dropbox\#FYP\Jenkins Web Assistant\images\128-blue.pngD:\Users\Mike\DropboxRPG\Dropbox\#FYP\Jenkins Web Assistant\images\128-green.png

#### Figure 4.: Sample coloured icons. From left: BLACKSUIT, BLUESUIT, GREENSUIT.

When clicked, the notifications will perform a different action based upon the function. The first function, notificationURL() is used primarily by the scheduler in order to open the web page on the schedule. When clicked, the extension will open the destination in a new tab. The second function, notificationFunction() calls a function when clicked. This is used to switch the user to an existing tab, as well as performing other functionalities. Both functions utilise Function Overloading in order for them to be called without certain parameters.

|  |
| --- |
| //Notify user  ***function*** notificationURL**(**notificationTitle**,** bodyText**,** destination**,** type**)** **{**  ***var*** iconImage **=** '/images/128.png'**;** //Default. When type == 0  ***if*** **(**type **==** REDSUIT**)** **{**  iconImage **=** '/images/128-red.png'**;**  **}**  ***else*** ***if*** **(**type **==** BLUESUIT**)** **{**  iconImage **=** '/images/128-blue.png'**;**  **}**  ***else*** ***if*** **(**type **==** GREENSUIT**)** **{**  iconImage **=** '/images/128-green.png'**;**  **}**  ***else*** ***if*** **(**type **==** GREYSUIT**)** **{**  iconImage **=** '/images/128-grey.png'**;**  **}**  ***else*** ***if*** **(**type **==** NOSUIT**)** **{**  //Whatever for no suit  **}**  //Check whether notifications are enabled  chrome**.**storage**.**sync**.**get**({**  notification**:** ***true***  **},** ***function*(**items**)** **{**  ***if*** **(**items**.**notification **==** ***true*)** **{**  ***var*** notification **=** ***new*** Notification**(**notificationTitle**,** **{**  icon**:** iconImage**,**  body**:** bodyText**,**  **});**  //Whatever I want the notification to do  notification**.**onclick **=** ***function*** **()** **{**  ***if*** **(**destination **!=** null**)**  window**.**open**(**destination**);**  //Close notification after clicking  notification**.**close**();**  **};**  **}**  **});**  **}** |

#### Figure 4.: Notification URL function

##### D:\Users\Mike\DropboxRPG\Dropbox\#FYP\Other Reports\Notifications.PNG

#### Figure 4.: Example notifications. Notification on the right shows an automatically processed schedule entry.

##### Adding Schedule Entries

Schedule entries are added on the Interface page. The time and URL are entered into the text box and the “Submit” button is clicked. The entry is compared using Regular Expression, and if the format is correct, the entry is saved to the list. If the format is incorrect, an error message will be displayed.

#### D:\Users\Mike\DropboxRPG\Dropbox\#FYP\Other Reports\addItem.PNG

#### Figure 4.: Error message display

If the “Accept” button is clicked, the entry will become validated and turn green. If the X button is clicked, the entry is removed from the list and added to the rejectedSchedule array in local storage. A notification is created if either button is clicked, notifying the user of their action and allowing it to be undone.

Local storage was used to store the rejected items due to possible size constraints. Synchronised storage is limited to 100kB of storage, but each individual item is limited to 8kB and an error is created if the item exceeds this value. Local storage is not limited in this way.

# 4.6 Problems Faced

Name

Asynchronous Functions

Switching from Strings to Objects

Retrieving Tabs. Regarding windows etc. Existing tabs

Reformatting code, Variable Names and Code Duplication

List size and removing entries, multiple entries

Daylight Savings

Array on Startup

Automatic categorisation? And incompleteness of Patterner thing…

Meta tag checking

5. Testing and Evaluation

# 5.1 Introduction

Le’me introduce

# X.Y Problems Faced

Oh so many...

### 5.1.1 JavaScript Language

WORK ON THIS NOW THAT YOU’VE FINISHED

In order to create the extension for Google Chrome, it was required to use JavaScript for a large section of the functionality. This required learning JavaScript without any prior experience with the language. This was initially performed primarily online, intending to avail of the many free services available online.

JavaScript is an interpreted, dynamic, and untyped programming language based off of ECMAScript (Standard, ECMA, 2011)

Learning JavaScript proved to be more demanding than initially expected as it was different to many of the programming languages taught during the course of the degree, such as Java, C, and C++. JavaScript is dynamically typed and interpreted; this means that … which made certain familiar processes more complicated. Though the syntax for the language is derived from C, the semantics and design are influenced by other languages. The resources available for JavaScript

This meant that large a large amount of time was spent analysing tutorials and guides that dealt with already familiar subjects. The book *JavaScript and JQuery: Interactive Front-End Web Development* (Duckett, et al., 2014) was also consulted later in the semester due to the number of problems faced learning the language online. These problems learning the language meant significant delays for the implementation of the application.

### 5.1.? Netflix API

Initial plans for the extension were for it to work along with the Netflix streaming website. After completion of research and the commencement of the implementation of the extension, it was realised that the API for Netflix was no longer publicly available. This caused obvious issues with interfacing between the extension and the web application. Of the existing API that can be found online, a vast majority of it is outdated due to frequent updates by Netflix.

Due to the aforementioned issues with the availability of the API, the project was changed. The initial plan of an extension for Google Chrome was maintained, but was altered in order to work under a different medium while maintaining the ideals and intentions of the project. Thankfully much of the work completed thus far had been research or general extension functionality and allowed for a much more painless transition.

### Time Processing and Daylight Savings

Oh what a bother that was…

### Miscellaneous

Sgsgsgsgs

# 5.2 Project Testing

# 5.3 Evaluation and changes?

6. Conclusions and Recommendations

# 6.1 Introduction

The following sections will outline the conclusions gathered from the project as well as recommendations for further or similar work.

# 6.2 Conclusions

Thus I conclude that it is done

# 6.3 Recommendations

Don’t do drugs kids!

# Citations and Bibliography

Apple Inc., 2011. *iOS - Siri - Apple.* [Online]   
Available at: http://www.apple.com/ios/siri/  
[Accessed 10 April 2016].

Barakova, E., Mestrom, R. & Willemsen, W., 2011. The Application of Learning Algorithms in the Development of Natural Interaction. *Procedings of the Second Conference on Creativity and Innovation in Design,* pp. 81-84.

Burby, J., Brown, A. & WAA Standards Committee, 2007. *Washington DC: Web Analytics Association.* [Online]   
Available at: http://www.digitalanalyticsassociation.org/Files/PDF\_standards/WebAnalyticsDefinitions.pdf  
[Accessed 10 April 2016].

Codecademy, 2015. *JavaScript | Codecademy.* [Online]   
Available at: https://www.codecademy.com/learn/javascript  
[Accessed 7 December 2015].

Davenport, K., 2014. *The Cost Function of K-Means.* [Online]   
Available at: http://kldavenport.com/the-cost-function-of-k-means/  
[Accessed 6 February 2016].

Duckett, J., Ruppert, G. & Moore, J., 2014. *JavaScript and JQuery: Interactive Front-End Web Development.* First ed. s.l.:Wiley.

Godoy, D. & Amandi, A., 2005. User Profiling in Personal Information Agents: A Survey. *The Knowledge Engineering Review,* 20(4), pp. 329-361.

Google, 2016. *JavaScript APIs - Google Chrome.* [Online]   
Available at: https://developer.chrome.com/extensions/api\_index  
[Accessed 7 January 2016].

Grčar, M., Mladenič, D. & Grobelnik, M., 2005. User Profiling For Interest-Focused Browsing History. *Proceedings of the Workshop on End User Aspects of the Semantic Web,* pp. 99-109.

Guha, R., Gupta, V., Raghunathan, V. & Srikant, R., 2015. User modeling for a personal assistant. *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining,* pp. 275-284. ACM.

Holland, S., Ester, M. & Kießling, W., 2003. Preference Mining: A Novel Approach on Mining User Preferences for Personalized Applications. *Springer Berlin Heidelberg,* pp. 204-216.

Holland, S. & Kießling, W., 2004. User Preference Mining Techniques for Personalized Applications. *Wirtschaftsinformatik,* 46(6), pp. 439-445.

Hussain, A., 2013. *Personal Smart Assistant for Digital Media and Advertisement (Doctoral dissertation, The University of Western Ontario).* [Online]   
Available at: http://ir.lib.uwo.ca/cgi/viewcontent.cgi?article=2371&context=etd  
[Accessed 10 April 2016].

Iglesias, J. A., Angelov, P., Ledezma, A. & Sanchis, A., 2012. Creating Evolving User Behavior Profiles Automatically. *Knowledge and Data Engineering, IEEE Transactions on,* 24(5), pp. 854-867.

Krug, S., 2014. *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability.* Third ed. s.l.:New Riders.

Lardinois, F., 2015. *Apple Announces Proactive Assistant, Its Google Now Competitor| Tech Crunch.* [Online]   
Available at: http://techcrunch.com/2015/06/08/apple-announces-proactive-assistant-its-google-now-competitor/  
[Accessed 12 January 2016].

MacQueen, J., 1967. Some methods for classification and analysis of multivariate observations. *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability,* 1(14), pp. 281-297.

McLachlan, G. & Krishnan, T., 1996. *The EM Algorithm and Extensions.* Vol. 382: John Wiley & Sons.

Microsoft, 2015. *Cortana - Meet your personal assistant - Microsoft - Global.* [Online]   
Available at: https://www.microsoft.com/en/mobile/experiences/cortana/  
[Accessed 10 April 2016].

Mozilla Foundation, 2016. *JavaScript | MDN.* [Online]   
Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript  
[Accessed 16 March 2016].

Murphy, K. P., 2012. *Machine Learning: A Probabalistic Perspective.* First ed. s.l.:The MIT Press.

Nakatsuji, M., Yoshida, M. & Ishida, T., 2009. Detecting innovative topics based on user-interest ontology. *Web Semantics: Science, Services and Agents on the World Wide Web,* 7(2), pp. 107-120.

NetMarketShare, 2016. *Browser Market Share.* [Online]   
Available at: https://www.netmarketshare.com/browser-market-share.aspx?qprid=2&qpcustomd=0  
[Accessed 20 March 2016].

Norman, D., 2013. *The Design of Everyday Things: Revised and Expanded Edition.* Revised ed. s.l.:Basic Books.

Oxford Dictionary, 2016. *User - Definition of 'user' from the Oxford Dictionary.* [Online]   
Available at: http://www.oxforddictionaries.com/definition/english/user  
[Accessed 10 April 2016].

Perkowitz, M. & Etzoni, O., 2000. Towards adaptive Web sites: Conceptual framework and case study. *Artificial Intelligence,* 118(1), pp. 245-275.

Rossi, G., Schwabe, D. & Guimaraes, R., 2001. Designing Personalized Web Applications. *Proceedings of the 10th international conference on World Wide Web,* Volume pp. 275-284. ACM.

Roush, W., 2010. *The Story of Siri, from Birth at SRI to Acquisition by Apple—Virtual Personal Assistants Go Mobile.* [Online]   
Available at: http://www.xconomy.com/san-francisco/2010/06/14/the-story-of-siri-from-birth-at-sri-to-acquisition-by-apple-virtual-personal-assistants-go-mobile/#  
[Accessed 9 April 2016].

Silverstein, C., Marais, H., Henzinger, M. & Moricz, M., 1999. Analysis of a very large web search engine query log. *ACM SIGIR Forum,* 33(1), pp. 6-12.

Sipper, M. et al., 1997. A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems. *Evolutionary Computation, IEEE Transactions on,* 1(1), pp. 83-97.

Standard, ECMA, 2011. *262: ECMAScript Language Specification, December 1999.* [Online]   
Available at: http://www.ecma-international.org/publications/standards/Ecma-262.htm  
[Accessed 10 April 2016].

StatCounter Global Stats, 2016. *StatCounter Global Stats - Browser, OS, Search Engine including Mobile Usage Share.* [Online]   
Available at: http://gs.statcounter.com/  
[Accessed 20 March 2016].

Vogeley, K. & Bente, G., 2010. "Artificial humans": Psychology and neuroscience perspectives on embodiment and nonverbal communication. *Neural Networks,* 23(8), pp. 1077-1090.

WEKA, 2015. *weka - Use WEKA in your Java code.* [Online]   
Available at: https://weka.wikispaces.com/Use+WEKA+in+your+Java+code  
[Accessed 26 October 2015].

Witten, I. H., Frank, E. & Hall, M. A., 2011. *Data Mining: Practical Machine Learning Tools and Techniques.* Third ed. s.l.:Morgan Kaufmann.

Ask about referencing the APIs and other sites frequently visited.

ASK ABOUT REMOVING CODE NOT MENTIONED AND OTHER THINGS