Syntax and Semantic

Learning Tool

Thomas Hutchinson

May 2018

Computing Science

Supervisor: Dr Lindsay Marshall

Word Count: 999,999,999,999.21

Abstract

This paper wishes to explore and build upon already available Syntax and Semantic Learning tools currently available on the market (such as Stack Overflow) to create an application. This application will then provide a better learning experience to the end user allowing them to find and pick up required syntax and semantics for any required language.

The motivation behind this Dissertation and application was an attempt to apply some of the advances in the pedagogy of programming/scripting languages to a real learning tool. The new application was meant to be exploration of what can be possible when the pedagogical theory is applied

Declaration

*“I declare that this document consists wholly of my own work except where explicitly stated”*

*Thomas Hutchinson*

Acknowledgements

I would like to thank my own lazy arse for being bothered to write this stupid paper.

**Table of Contents**

[1. Introduction 6](#_Toc510795864)

[1.1. Aims and Objectives 6](#_Toc510795865)

[1.2. Table of Figures 7](#_Toc510795866)

[2. Background Research 8](#_Toc510795867)

[2.1. Market Research 8](#_Toc510795868)

[2.1.1. Reasoning 8](#_Toc510795869)

[2.1.2. Method 8](#_Toc510795870)

[2.1.3. Findings 8](#_Toc510795871)

[2.1.4. Summary 8](#_Toc510795872)

[2.2. Audience Research 10](#_Toc510795873)

[2.2.1. Survey Rationale 10](#_Toc510795874)

[2.2.2. Survey Results 10](#_Toc510795875)

[2.3. Language Research 14](#_Toc510795876)

[2.3.1. Reasoning 14](#_Toc510795877)

[2.3.2. Method 14](#_Toc510795878)

[2.3.3. Findings 14](#_Toc510795879)

[2.3.4. Summary 14](#_Toc510795880)

[2.4. Academic Research 15](#_Toc510795881)

[2.4.1. Language Pedagogy 15](#_Toc510795882)

[2.4.2. Crowdsourcing and Gamification. 16](#_Toc510795883)

[2.5. Implementation Technology 18](#_Toc510795884)

[2.5.1. What is a Web Application? 18](#_Toc510795885)

[3. Methodology and System Design 20](#_Toc510795886)

[3.1. Overview 20](#_Toc510795887)

[3.2. Functional Requirements 20](#_Toc510795888)

[3.3. Non-Functional Requirements 21](#_Toc510795889)

[3.4. High Level Design 22](#_Toc510795890)

[3.5. Backend Server-Side Design 23](#_Toc510795891)

[3.6. Database Design 24](#_Toc510795892)

[3.7. User Interface Design 26](#_Toc510795893)

[3.8. Design Reasoning 29](#_Toc510795894)

[3.9. Project Plan 30](#_Toc510795895)

[3.9.1. Application Development Plan 31](#_Toc510795896)

[3.9.2. Optional Application Features 32](#_Toc510795897)

[4. Implementation 33](#_Toc510795898)

[4.1. Overview 33](#_Toc510795899)

[4.2. Implementing the Voting System 34](#_Toc510795900)

[4.3. Implementing the Post Tagging System 35](#_Toc510795901)

[4.4. Implementing the Search and Sort System 36](#_Toc510795902)

[4.5. Implementing the Real Time Data Aspect. 37](#_Toc510795903)

[4.6. Final Deliverable 38](#_Toc510795904)

[5. Testing 39](#_Toc510795905)

[5.1. Black Box Testing 39](#_Toc510795906)

[5.1.1. Method 39](#_Toc510795907)

[5.1.2. Report 40](#_Toc510795908)

[5.2. Functionality Testing 41](#_Toc510795909)

[5.2.1. Method 41](#_Toc510795910)

[5.2.2. Report 42](#_Toc510795911)

[5.3. Miscellaneous Testing 43](#_Toc510795912)

[5.3.1. System Stress Test 43](#_Toc510795913)

[5.3.2. Anything Else That Comes To Mind 43](#_Toc510795914)

[6. Evaluation 44](#_Toc510795915)

[6.1. Audience Survey Results 44](#_Toc510795916)

[6.2. Focus Group Testing 45](#_Toc510795917)

[6.3. Project Requirements 46](#_Toc510795918)

[6.4. App Reviews 47](#_Toc510795919)

[7. Conclusion 48](#_Toc510795920)

[7.1. Overview 48](#_Toc510795921)

[7.2. Aims and Objectives 49](#_Toc510795922)

[7.3. Summary 50](#_Toc510795923)

[7.4. Future Improvements 51](#_Toc510795924)

[8. References 52](#_Toc510795925)

[9. Appendices 53](#_Toc510795926)

[9.1. Appendix A: 1st User Survey Results 53](#_Toc510795927)

[9.2. Appendix B: A Load of Crap 54](#_Toc510795928)

# Introduction

This section provides a general overview of the Dissertation covering the Dissertation structure itself as well as the general aims and objectives.

## Aims and Objectives

**Aim:** To create a tool that can be used by both intermediate and expert level programmers to aid in their acquisition and learning of new [programming/scripting] languages syntax and semantics.

1. Research existing products currently available on the market and how my tool can offer a service that is different from competitors while still allowing its users to learn new syntax effectively.
   1. Efficiency will be marked by the user being able to find their required information with only a single search and to have found a suitable post within 5 minutes*If a suitable post exists*
2. Research the Pedagogy of [programming/scripting] languages to ensure the tool teaches its information in the most effective way possible.
   1. Effective learning will be defined as a user being able to acquire that knowledge then apply it in a project/ work environment.
3. Conduct a review into the user perception/usage of the currently available tools as well as my hypothetical tool.
4. Develop said tool using the collected information and research to influence design decisions and the overall user experience.
5. Evaluate the performance of the tool in relation to the research from the previous objectives once completed by measuring various statistics such as user activity and user retention.

## Table of Figures

|  |  |  |
| --- | --- | --- |
| Figure Number | Description | Page |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Background Research

## Market Research

### Reasoning

When creating the new application, it was important to review the current applications available and used on the market. It would be a waste of time and resources to create a product that’s main feature is already encompassed by another developer. Hence, why it was important to conduct a throughout review of the currently available products taking note of their primary features, positives and negatives.

### Method

To ensure each application is reviewed thoroughly and only the points of interest are examined each application will be subjected to a specific process. The reviews will primary focus on how the platform delivers, sorts and presents its content to the end-users. On top of this since the created application will be crowd sourced only currently available platforms that also make use of this content creation system will be reviewed. This is done to ensure that all conclusions are related to the created application and to also help reduce the scope of related applications.

Once a valid product to review had been identified the main features would then be carefully examined to see how the audience used and reacted to those features. A valid product once again is anything that teaches programming/scripting languages to an audience with that content being crowd sourced. A main feature can then also be defined as any feature that greatly helps the learning experience or any feature that is present in that application but no others on the market.

### Findings

For a full detailed list of findings check appendix () for the full write up.

### Summary

While searching the market for products like the one outlined in this paper one product stood out from among the rest and that was Stack Overflow. Due to it’s wide spread use and popularity it boosts one of the widest ranges of content on the internet. This directly feeds into it’s popularity as it provides tutorial material on just about every topic. Its also helped by having very good SEO (Search Engine Optimisation) allowing users to easily find posts through popular search engines such as Google. In the audience research this was one of the most popular points of the system *(see chapter2.2).*

So, it can be concluded that Stack Overflow simply works by being one of the most popular systems on the market. However, it does fail in the fact it doesn’t adequately reward posts that fully explain their topic. As mentioned in the full market report *(see appendix )* users creating content are incentivised to provide straight solutions to responses. This does not encourage learning of the background syntax and semantics and this is where my system could provide additional functionality.

By learning from the downsides of big sites such as Stack Overflow it is possible to further improve the outlined application. In this case by focusing on giving the user tools to upvote a post on a range of qualities that have been found to improve learning. The application can perform better than these other services. That said It will be incredibly difficult to ever compete with these services simply due to their large content and user base. That is why when it comes to the final evaluation it will be important to remind users to judge on the potential of the new system. With that in mind it will be easier to judge how the altered focus on learning and improving over simply solving issues will improve programmers greatly.

## Audience Research

### Survey Rationale

Another important part of determining which direction my system should go in is by asking my target demographic a few questions relating to their current tool usage and learning habits. I can achieve this through focus group testing but also an audience survey. This survey would be targeted at both student/” junior” developers and senior developers alike to gauge opinion about my possible tool. I primarily want to find out how developers learn new syntax and how/what tools they currently use to solve problems. Below is a list of possible questions for the survey alongside a short purpose of asking that question.

### Survey Results

After hosting the survey online for one month a total of 16 answered it. While this response number is not brilliant for forming conclusions about my hypothesis. Using it in conjunction with my academic research as well as the follow up survey after the completion of the app will help to give more reliable data about the validity and correctness of my hypothesis.



Another point of note is the distribution of occupations. Due to the limited reach of the survey it was primarily completely by Full-Time Students. While this isn’t necessarily a negative point it does mean that my conclusions must be adjusted to account for the primary demographic of my application being students.



The survey also revealed that Stack Overflow is by far the most popular language learning tool amongst the student demographic. This means that the market research should primarily focus on what makes Stack Overflow so popular with this demographic.

On top of this it will also be worthwhile to explore why the other tools do not hold as much favour with this selection of students. As you can see from the above graph Stack Overflow was marked as a 7 in usefulness by most students who answer this section. That is directly compared to an Article based site like Dev.to which received a high selection of 1 responses. This data will be useful in future conclusions regarding the effectiveness and popularity of certain applications over others.



Another interesting response is trend towards tool dependency. Out of the 16 people who answered the survey 62.5% of the respondents voted 4 or above claiming that losing access to their preferred tool would hold a detrimental effect on their normal workflow. Once again this is useful information for later as it shows that users may not be learning or retaining information which requires them to make multiple visits to their tools. On the other hand, it could just show that students are required to develop in wide range of languages which means they are in constant need of additional learning materials.



Regarding the layout and development of the app one surprise was the number of respondents who prefer a solution-based search system. The respondents answer clearly show that it is greatly preferred when a tutorial is given with a specific real-world example. When it comes to developing the application, it will be important to attempt to implement this idea and to allow the linking of tutorials and specific examples. This could be a link to the idea that high level theory is difficult to learn by itself so showing the application of that theory makes it easier for a wider range of programmers to learn and apply that concept. Therefore, for the tool to be successful with this user base it needs to rely on linking theory to concreate examples.



Yet another interesting point of note is the number of respondents who rely on Stack Overflow posts to acquire and learn new languages. Once again this speaks volumes about Stack Overflow’s ability to provide tutorial material to its users who on average return whenever they need to learn a new language. The second highest response for this question with 13 answers is “taught by a teacher/expert”. So, the best way or at least the most favoured way to learn is by example or by expertise. This will be an important element when it comes to my application as It needs to allow the experts of a language to fully demonstrate and create tutorials for specific topics. By looking at these survey results it can be concluded that the respondents prefer learning from experts and being given examples of the theory in motion.

## Language Research

### Reasoning

Another aspect important to the background research is the study of Computing language itself. By studying this area, it will give some direction to how the application should be developed and what types of features would be useful. Giving the users the correct tools to create posts that help to teach a range of languages and language features will be incredibly useful in keep users retained. On top of that making sure the created application is robust enough to support a range of languages will be important in getting plenty of users creating content for the application.

### Method

Simply research languages through available material such as: official documentation, online forums, unofficial documentation and created tutorials. All this data will be gathered, compared and reviewed to determine conclusions and ideas about specific languages. Those conclusions will then be reviewed and used to help determine how the Syntax and Semantic Learning tool should be created and what types of features it should have.

### Findings

For full findings please read the full report in Appendix ()

### Summary

Languages have a wide range of features and an even wider range in which those features are defined and implemented. For specific examples please refer to the *appendix* but it is clear than even from a brief review of various popular languages that they can differ rather largely. That produces the conclusion that users must be able to search via specific languages and able to filter content based on the language they are currently interested in. Users wanting to learn how to perform a specific technique in one language need specific syntax for that language so by putting a specific tag for languages. Users will be able to more easily find specific content to their liking. That said many of the languages share the same semantical features such as if statements or for statements. These posts could be created with specific syntactically definitions which then link into a single semantical post that explains the background workings. This is because over many of the languages the semantics remain the same however the syntax can change.

## Academic Research

### Language Pedagogy

Teaching languages can be a difficult and complex topic to discuss because every single learner will be subject to different parameters and situations. A learner’s previous knowledge and understanding of programming paradigms and concepts has been found to play a big role in their aptitude for understanding new language. Leon Winslow concluded that an expert can create various mental models and implement generic programming concepts easier than novices. He also states that a novice may only become an expert through years and year of practice and refinement *[Winslow 1996]*. It then stands to reason after these years of training and practice they have an understanding and skill that allows them to acquire these languages faster.

There is another aspect to this practice though. The simple act of constant practice isn’t the only element that increases one’s aptitude for language acquisition. Semantics also plays a huge role in determine a “programmer’s general skill/ability” found Allan G. Bateson *(Allan G. Bateson, Ralph A. Alexander, Martin D. Murphy 1987)*. A programmer’s understanding of core fundamentals and topics will aid them far more than specific syntax knowledge when understanding new languages. Bateson also concluded that the greatest gauge of a programmer’s general aptitude is their semantical knowledge over their syntactical knowledge.

Both Bateson’s and Winslow’s work combined help paint a more detailed picture of how one learns a programming language and learns it effectively. A novice will simply focus on surface features such as control structures and simple iteration while an expert will dive deeper and focus on more complex aspects of a language much faster *(Winslow 199)]*. This is because of their ability to understand complex features such as object orientation and they can manipulate that to solve problems much faster and to a much higher standard. Their code will consist of less errors, be more efficient and reach specifications much easier. A new language shouldn’t affect an expect as much as a novice because they understand and can apply their knowledge regardless of the environment.

Regarding my own application these papers have helped to direct my own tool in the direction of providing syntactical assistance while rooting that assistance in semantic information. By this I mean, users will be rewarded for thoroughly explaining code snippets and referring them to generic programming structures and concepts rather than just writing the syntax with no explanation. This benefit both experts and novices as the experts get the code syntax they require, and novices can expand their knowledge of programming semantics. This over time will help develop them into better programmers by improving their semantical knowledge rather than just their syntactical knowledge. Which in turn improves their overall programming aptitude.

### Crowdsourcing and Gamification.

For the syntax/semantic tool to be of use to anyone it is going to require a large amount of data. This data will have to be created covering a different topic and be sourced in a short time-frame. That is why crowdsourcing will be an indispensable technique in helping to build up the resource base and to sustain its growth. By gathering a community based around the application and encouraging accurate content creation the potential of the application can be realised.

What is crowdsourcing though and how can it be used effectively to gather a large quantity of data in a short time span (1 to 2 months)? Well David Geiger summarised crowdsourcing into 4 major categories: Crowd Rating, Crowd Solving, Crowd Processing and Crowd Creation. These categories where devised on the type and complexity of contributions required from the user as well as how those contributions were used and evaluated *(Geiger 2011)*. After analysing the requirements of my system, I devised that the best styles of crowdsourcing for me would be the Crowd Creation and Crowd Rating.

These two styles would be separated across the two primary user groups. The posters and the searchers. Users may travel between these groups freely but when completing a task, they will be a part of either one group or the others. The posters will follow a Crowd Creation crowdsourcing. This involves the users being given the freedom to create certain content within the specification of the site *(Geiger 2011)*. Using a form that requires them to enter certain information but then allows them to create whatever relevant text-based content they want they will have the freedom to provide high quality content. Searchers can then make use of a Crowd Rating system to assign value to these posts and ensure high quality content is rewarded. Geiger himself commented on this relationship between creation and rating saying that a larger audience of creators requires a larger audience of people rating *(Geiger 2011)*. This is because after a certain point and a large enough community managing in-coming content becomes impossible and a larger crowd is required to keep on top of it.

Unfortunately, crowdsourcing is useless without a crowd and for my system I would require a constant healthy stream of new content meaning the tool not only needs a crowd but a consistent one at that. Rather than amassing a crowd it would instead be better for the system if a community was constructed. Communities often develop around a mutual point of discussion or common goal and a healthy community remains as bonds and trust are developed *(Zwass 2010)*. By encouraging user interaction through the voting system and comments sections a community will develop around the application. This in turn encourages further user involvement and increases the chances of the user’s continued use of the application. Which is what the application requires to provide up to date syntactical resources to the searching users.

On the topic of community retention. Keeping a community focused and attached to a specific application or concept can be the most difficult aspect of any crowdsourced project. It is the natural way of humans to start of enthusiastic about topics and activities and then gradually over time become board or disinterested. This then leads to them leaving the platform and migrating somewhere else. This would be categorised as a steady increase in users then over the span of several weeks and months the user count slowly stops increasing and then starts decreasing. This then induces a death spiral in users as the content begins to stagnate and more users leave because of the other users leaving. This then concludes in the platform only being left with a skeleton of its former self and it can be nearly impossible to revive a platform after that. See the History of Myspace as a perfect example of this. A more interesting platform appeared, and a clear majority of the users quickly migrated. *(Wikipedia-Myspace 2017)*

So, to improve user retention the application requires something else. An extra system needs to be developed and deployed to attract new users and keep the old, invested, users around. This system could be based around the theory of Gamification. Gamification is the process of adding features and elements of games to non-game tasks. This has been shown to increase user retention as well as increasing the chance of attracting users in the first place *(Luis von Ahn 2008)*. There are several ways of implementing game features and these ways can have very levels of effectiveness based on how they are implemented and for what purpose. Luis spoke about user ranking in his 2008 magazine article. This is the process of taking a user’s input then quantifying that contribution in terms of effort and quality and giving them a score. That score can then be applied to ranks and leader boards *(Luis von Ahn 2008)*. This could be a brilliant feature for my site as it can be applied to both the searchers and the posters.

Posters can be awarded based on the communities’ evaluation of their post. Searchers can then be awarded for providing frequent comments and rating on these posts. The level of reward would have to be based on level of contribution as if users are rewarded for simple tasks such as giving rating they have been shown to give more spurious ratings *(Zwass 2010)*. So, a smaller score could be given when a user rates and a larger score given to high quality posts. This score can then be acquired over the user’s lifespan and as they reach certain milestones they can “rank up”. This is a badge they can display in comment boards as well as their profile page. It can be deduced that a user who has spent a long time acquiring a high rank will be less likely to leave the site.

With the techniques of Crowdsourcing and Gamification combined the syntax tool should be able to gather a large data set of learning resources and then maintain a steady stream of new data as required. Users will be encouraged to join in and stay around because of the community of fellow posters and researchers who can communicate in comment sections. They can then strive together to gain more points by posting and completing tasks that benefit the site such as creating posts and rating effective content. This in turn will then attract new users to the site due to the range of high quality content. If all goes to plan it should be steady upwards spiral of user numbers as the invested community rises thanks to word of mouth and recommendations.

## Implementation Technology

The system will be implemented as a progressive web application. This decision is based off the audience survey *(result below).* As you can see all platforms was selected as the most popular option closely followed by Desktop Application (windows). Due the large different in application types this Dissertation does not have the resources to develop on all platforms. It is therefore decided that a Web Application will be created as it services the most users with the available resources.



### What is a Web Application?

A Web Application is rather like a normal website though it attempts to mimic the behaviour of a native application. This means that the user has a constant state that is linked to their account, so they can use the application across various computers and even in some cases offline. This is achieved through extensive caching both on the user end and the server end storing any settings or posts they create. That means that when they re-connect that content is readily available.

Another advantage is that the software is Operating System free. As outlined in the above results the users want the application on a range of platforms however there is only so much development time. A Web Application can perform on several platforms at once simply because it is a normal website. Any platform that has a modern browser such as Google Chrome, Firefox or Edge can view and use the application.

What makes a Web Application stand out compared to a normal website is a cross between its cache ability, platform independence and real time nature. Users do not need to press buttons to load content it is simply loaded on demand and then cached so it may be viewed at any point. This will be achieved through Google’s Polymer as well as Google’s Firebase platform. These two technologies allow real time database calls and updates which means the user can stay constantly update to date with the latest content. This platform has also been chosen due to prior experiences which will help to speed up development.

Other platforms are available such as Re-act from Facebook however, I have not had as much experience of this platform or any others. The project will be held to a tight deadline, so all advantages must be taken to ensure the application is made to the highest possible standard in the shortest time possible.

# Methodology and System Design

## Overview

The aim of the Syntax and Semantic learning tool is to facilitate the learning of new syntactical features of programming/scripting languages. On top of that the tool will also allow users to gain a greater understanding of the background semantics of that specific syntax. This is done to help increase the programmer’s overall proficiency as they will have a fuller understanding of that syntax and how it works. All of this will be achieved through a crowd sourcing model. The reason the application will rely on crowd sourcing is because there are far too many languages that are used by professionals. Creating content for these languages would require more effort than is currently available for this project.

Crowd Sourcing allows the application to rely upon it’s community to keep it alive. This is a double-edged sword however as if the community does not support the application the application will fail due to a lack of content. That is why this prototype aims to get an idea of what the community would like from a syntax and semantic learning tool. This information could then be used to fully develop the application in a manner that the general programming community would approve of. This in turn will increase community retention and therefore support the life of the application itself.

## Functional Requirements

|  |  |  |
| --- | --- | --- |
| Number | Name | Description |
| FR1 | User can create tutorial posts | A singed in user can create a post detailing a specific syntactical/semantical feature of a language. |
| FR2 | Users can tag other posts to their created post | A signed in user can tag other posts (creating a link between them) to help the explanation of specific topics by linking to other user’s (or their own) content. |
| FR3 | Users can view other users posts and rate specific attributes of that post. | A user will have access to a feed where they may view other users posts and either upvote or downvote those posts based on various criteria such as: well explained, well linked, and general quality. |
| FR4 | Users can Search for other user’s posts based on specific criteria such as username, post title, post language and sort that search data based on its attributes. | A user can search for posts using the username, post title, the language that post is explaining and then sort the post based on different attributes (well explained, well linked etc) |
| FR5 | Users can create bounties, rank other bounties and mark their own bounties as fulfilled | A bounty is a request for a specific tutorial to be made. Other users can then view these bounties and create a specific post for them. Users can also rank these bounties to help improve its visibility to other users. |
| FR6 | Users may view their own created posts and delete them | If a user is unhappy with their post they may delete it from the website. |
| FR7 | Users can save posts for offline viewing. | By viewing a post, a user can save it to an offline cache where they may view it later. |
| FR8 | Users can comment on posts | By viewing a post, a user can place a comment on that post discussing it’s contents. Other users can then view this comment. |
| Documentation | | |
| FR9 | The code will be thoroughly documented | The code base will contain an audited amount of comments to explain it’s working as well as to credit any externally created code. |
| FR10 | A user manual will be created | A public use user manual will be created to allow users to understand how to use the application. |

## Non-Functional Requirements

|  |  |  |
| --- | --- | --- |
| Number | Name | Description |
| NFR1 | The site should be inherently easy to understand and use. | A user should be able to use the site without any prior guidance. This should be achieved by basing the interface of currently available applications to reduce the time to learn. |
| NFR2 | Posts that encourage semantical learning over simple solution-based responses should appear at the top of feeds and searches where possible. | Where available posts that thoroughly explain concepts and don’t just provide solutions should be favoured in results. While this can’t be directly controlled post, attributes should help sort this content. |
| NFR3 | Creating a post should be simple and easy to understand not requiring any additional knowledge. | This will be achieved through keeping the text editor and post submission form as simple as possible. |

## High Level Design

The Web Application itself will be created with a cross of Google Firebase Technologies as well as Google Polymer 2.0. Polymer focuses on the combination of multiple web components which are individually created and defined in custom DOM scheme. This helps to abstract away complex HTML, CSS and JavaScript which could become quite difficult to manage in a larger project.

This works by creating an element much like a normal HTML element would be created. For example, you would create a form using the standard HTML 5.0 mark-up. You can then import some elements from the Polymer library and define that HTML snippet as a custom element. That element can then be imported into another HTML document and used much like a standard HTML tag. So that form could be called “my-Form” and now you can implement simply by creating a set of <my-form> tags. Once again, the main advantage of this is to abstract complexity away when it is not being worked on.

In my specific use case. I can create separate pages and elements and then import them into a single application element (see diagram below). This means I can easily focus on one element at a time rather than creating a single huge page. It also makes it very easy to swap elements in and out if I want to experiment or try different implementation strategies.



On top of the custom created elements there will also be the Bower components. Bower is a component management software that assists in the installation and management of dependencies of all the additional third-party components I will be using. Most of these components are constructed by Google or Google afflicted companies and include such things as: layout assistance, loading bars, view swaps and other smaller components. These will help save development time by cutting down on the development of items which have already been created by others.

The Polyfire components are the components which allow the combination of the Polymer and Firebase. These components handle elements such as user authentication and interaction with the real time database. By importing them through the index page rather than codex-app the entire site can have access to the data from the Firebase connection. This data is then protected by Polymer’s shadow-DOM. This is a ratification DOM structure created to handle all the various components. It also helps to prevent attacks by making it incredibly difficult to poll data from the website itself.

## Backend Server-Side Design

Firebase will be used to handle all back-end data processing. This primary involves user authentication as well as requests and posts to the database. Thankfully, Firebase directly supports both functions and has cross-over components that allow the communication between polymer and firebase. These are included in a library called Polyfire.

Starting with the database. The application will make use of Firebase’s “Real Time Database” structure. This is very similar to a SQL-less database. That means it does not rely on common SQL style functions and it is not a relational style database. Meaning that it does not create relations between records. This has both its advantages and disadvantages. The disadvantages being that having relations between records becomes difficult/impossible and the repetition of data can easily happen. That means careful consideration must be taken when designing the database.

“Real Time Database” from Firebase works on a Document foundation. This means that the database consists of objects. These objects then consist of fields and those fields can also be arrays/lists of objects within themselves. Through this mechanic one may achieve a relation between data. So, when creating the database relations will be displayed through these internal lists. For example, the post document will be a list of posts and within those posts each post will have another list representing its comments.

The primary advantage to using this approach is that is becomes faster and easier to process single objects. Codex (the application) will often be dealing with a single post or a small number of posts (less than 50). That means when pulling a post, it will be much easier to handle of its data together in an object style paradigm. This is because all the data is present upon call avoiding the resource intensive procedures such as table joins. Real Time Database avoids this by having all the objects essentially being pre-joined saving the server from performing this expensive operation. This in turn speeds up processing allowing the site to process more users at once which is important in a site that may need to scale to a large user base quickly.

## Database Design

As mentioned in the previous section to increase the speed of the web server and therefore reduce user-end load times a document style database will be implemented. This foregoes costly operations such as table joins for storing “objects” within a database. These objects can then be individually queried from the application which will return all the data held within that specific object. With that in mind certain design constraints must be considered when designing the database.

For a start if not handled properly the database could suffer from a great deal of anomalies. Anomalies can be created when there are duplicate versions of data present and one item of data is edited or deleted which creates inconsistencies across the database. These inconsistencies can then escalate resulting in wasted memory space from storing the same data twice or confusion when it becomes difficult to tell which item of data is the most current. On top of that anomalies can also occur if two data items rely or reference one another and one of those items is deleted. This now means that an object has a “Null” field and is missing data making that record invalid. So, when it comes to developing the database all objects must contain all data relating to them where possible.



See above figure for full design of the database. As you can see the post object is by far the most complex out of all the objects and that is simply because it is the centre of the application’s design. All aspects of the system rely on posts being managed correctly and that means that the post must contain a lot of data so that I can do its job successfully.

The log objects attached to the post objects are lists that are utilised in the up-voting procedures. When a user upvotes a specific attribute of a post their unique identification number is added to this log object. This means that in the future if the user attempts to vote on the same post again the server can check to see if that user has already voted. If they have voted it will simply disallow them from voting further. A separate running number total of total votes is also kept, and this is to save the server performing a count operation on the list every time the rank is requested. It can be expected that the rank will be requested far more than the upvote procedure, so it makes sense to store that number for easy retrieval rather than calculating the number of votes from the log.

Another aspect of note is how the comments are stored alongside the post they are related to. Once again, this document style database does not support conventional relations so to emulate that behaviour comments are stored within the post object. Meaning when a user views post the comments are passed alongside that post saving additional queries and complex joins. This is done to reduce load on the server to allow the application to serve as many users as possible on limited hardware and bandwidth.

## User Interface Design

Linking back to the market research in chapter 2.1 all the successful learning tools implement a simple yet powerful interface. This style of interface can be defined as having a navigation bar with minimal options upon it and relying upon the user to utilise a search feature to narrow down content. As seen with Stack Overflow the search bar takes up prominent space at the top-centre of the page.

This is something that should be directly applied to the application. These sites are successful because users can quickly and easily navigate them to find their required content. Any site that impedes the users progress will quickly fall out of favour. Once again linking back to Geiger [Geiger 2011] on Crowdsourcing systems. The system must retain users can work together with their community to encourage a healthy life cycle. The cycle consists of new users discovering the site through search engines and word of mouth. They then begin to integrate that tool into their daily work routine and eventually they will begin to make content for the site encouraging more users to join.

With that in mind the interface needs to be easy to grasp. The initial designs for said interface are as follows in the figures below:



The above figure shows the Homepage/feed page. This is the initial design on what the user will see when they first login and it is an attempt to encourage them to explore and view new content they may not otherwise see. What is important though is the prominence of the search bar. The home page aims to give users an idea of what types of content will be present on the site meanwhile, not obstructing them if they are looking for something specific. If a user wants a specific post they can quickly and easily search for it.

This feature has been directly borrowed from Stack Overflow as makes the site easy to navigate. This featured was also praised in the audience survey *(see appendix bleh)*.So, it’s an important feature to add to the site.



Following the homepage is the create a post page. This is the second most important part of the site as it is where more dedicated users will spend most of their time. Here users will be able to create new posts, tag over posts and push that to the public feeds. Once again, it’s important to make this part of the site easy to learn and use. That is achieved by keeping the interface minimal and not over cluttering it with buttons. Between this design and the final product some of the buttons may even be removed to help keep this page clean.

Finally, there is the search page. This after the home page and the post page is the most important page of the site. It is where most users will spend most of their time as they search for specific content. In the audience survey *(see appendix )* users favoured sites with powerful search tools that allowed them to not only search for posts to but search in different manners. This could take the form of sorting posts based on specific criteria. It could also take the place of allowing users to search by programming/scripting language or maybe be a favourite author. Experimenting with this search tool then gauging user response will be a key aspect of this Dissertation to see how users utilise search tools.

## Design Reasoning

All the previous design features have been directly influenced by the research detailed in chapter 2. This sub chapter aims to provide a little more detail on the exact links between the research aspect and the design aspect.

Crowd Sourcing is a key word of this project as the system will succeed or fail based on how the audience perceives and uses the application. Zwass *(Zwass 2010)* spoke a great deal about communities and what impact they have on the crowd sourcing model. With that in mind during the research of various market products *(Chapter 2.1)* and the audience research *(Chapter 2.2)* a focus was placed on how the application developed their community. Stack Overflow became quickly clear that it’s community was based around solving problems and gaining points to increase one’s ranking. Meanwhile sites like Dev.to focused around personalities creating articles. This is then reflected in how the audience survey *(Chapter 2.2)* perceived these applications. The audience had integrated Stack Overflow into their work routine meanwhile far fewer users made use of Dev.to on a regular basis. This could just be down to Stack Overflow being a far more popular site, but it could also be down to users preferring solutions and direct learning over interest articles.

Taking that into account it is important that every aspect of design of the application leans into this research. The entire site should focus directly on teaching user’s new topics they want to find out about and stay away from opinion articles and interest articles. One way of encouraging solution and tutorial posts is through the rating system. By allowing users to rate posts based on their conciseness, well explained, well tagged then allowing them to filter posts based on those ranks it should encourage posters to adopt those qualities. Once again this is no certainty until the application is passed onto testing and the end users return their feedback on the feature.

The primary focus of the site is enabling users to teach themselves so when it comes to evaluating the successfulness of the rating and search systems a great deal of care will placed into getting throughout feedback. Testers will be questioned on how they used the rating system as well as to what extent they used the sorting system on the search page. Using this feedback, it can be hypothesized how users prefer to learn. As mentioned previously, it will be extremely difficult to gauge if the users have improved their skills through the usage of the application, so this paper must rely on the opinion and views of the users/testers time with the application.

## Project Plan

****Due to the limited development resources the project will generally take the form of a Waterfall approach. This means that tasks will be tackled sequentially and as sections become complete new tasks will become available. If the development team was larger an Agile development style could be adopted where tasks were committed to in stages and the order of completion in those stages wouldn’t be of importance.

However, the development team only consists of one developer, so the development schedule will fall under a waterfall mentality. Development will begin with the design and research section. During these phases a focus will be applied to deciding what features of the application should be considered critical and what features will be less critical.

In the above figure *(which you can find a larger version of in appendix )* the application development has been condensed into a single task. This is to save space on the plan and to help to keep development simple and on track. The application development will be broken down further as follows:

### Application Development Plan

|  |  |  |
| --- | --- | --- |
| Task Name – *(Component Name)* | Description | Estimated Duration |
| Workspace Setup | This includes setting up the server, downloading and installing all the required packages and setting up version control | 2 Days |
| General Layout Construction *(codex-app)* | Creating the structure of the site. Creating the skeleton of each of the components and connecting those components together. | 2 Days |
| Resolving Initial Dependency Issues | With so many packages working together there is a high risk some packages may conflict with one another. | 1.5 Days |
| Implementing O-Auth *(codex-login)* | Implement the ability for users to log in with Google O-Auth | 1 Day |
| Implement Polymer Real Time Attribute Binding | For the site to work correctly it is important that all the components can access the data they require. This task involves ensuring that all components have access to the data/objects they require. | 1.5 Days |
| Implement Basic Post Creation *(codex-posts)* | Implement the basic ability for users to create posts (does not involve post tagging) | 1.5 Days |
| Implement the Basic Post Feed *(codex-feed)* | Create a page where users can view all the created posts in one feed | 1 Day |
| Implement Post Voting | Creating the functions and procedures that allow users to vote on the various attributes of a post. | 3 Days |
| Implement Bounty Posting *(codex-bounty)* | Give users the ability to create bounties, upvote other bounties, view bounties and mark their own bounty as fulfilled. | 2 Days |
| Implement Post Search *(codex-search)* | Implement the ability for users to search for specific posts based on: author name, post title, post language and then sort results based on the posts attributes | 3 Days |
| Implement Basic Profile Screen *(codex-profile)* | Add a page where the user can view their posts | 0.5 days |
| Implement a 404 Page *(codex-404)* | Add a page that will display if the user enters an invalid address or attempts to access resources they do not have privilege for. | 1 Day |
| Create Home Page with News Feed and general site information *(codex-home)* | Add a news feed to the homepage to keep testers up to date with the progress on the application. | 1 Day |
| Implement View Post Page *(codex-view)* | Add a view details page for posts. Users will be able to eventually view tagged posts and comments on this page | 1.5 Days |
| Implement Post Tagging *(codex-post) (codex-view)* | Add the ability for users to tag posts in their post and then to view those tagged posts on the more detail screen | 3 Days |
| Implement Offline Caching *(codex-saved)* | Add the ability for users to save posts from the view post screen to an Offline cache so they may view posts offline. | 1.5 Days |
| Add Styling to the Navigation Bar *(codex-app)* | Add some additional styling and colour to the navigation bar to make it more appealing | 1.5 Days |
| Add Styling to the Entire Site | Add some additional styling and layout features to the entire site to make it more appealing. | 4 Days |
| Implement Real Time Post Commenting *(Codex-view)* | Create a discussion section on the view post page where users can create and view other people’s comments | 2 Days |
| Total Days: | | 48 |

With the above plan that means development will take roughly 48 days. Within the allotted time on the project plan being 59 days. The stated days on the application development do not equate 1:1 to actual days and instead refer to average work days so the required time to develop the application may be considerably less. With that said problems can arise and that is why ample slack time has been provided just in case.

The above application development also outlines the minimum viable product. All the above features and tasks must be complete by the deadline of the project to achieve the minimum functional product. This is a concept commonly used in industry where optional features are added but the developers and client decide on what features are critical to the application. The above table outlines those critical features without them the application cannot be declared finished. That said if develop finishes earlier than expected some optional features could be added.

### Optional Application Features

|  |  |  |
| --- | --- | --- |
| Feature Name | Description | Estimated Duration |
| Additional O-Auth Options | Add additional o-auth options such as Facebook, Twitter, Email and Password etc | 2 Days |
| Author Following and Advanced Feed Control | Give the users the option to customise what appears on their feed through following other users and defining preferences | 3 Days |
| Detailed Profile Page | Add more information and statistics to the profile page about the user’s activities | 2 Days |
| View Other Users Profile Pages | Give the ability for users to view each other’s profile pages if the user has enabled it in their privacy settings | 2.5 Days |
| Settings Page | Create a settings page where the user can adjust their experience with the application such as enabled accessibility options or changing the applications theme | 4 Days. |

# Implementation

## Overview

This section will cover the implementation of the application. As mentioned in chapter 3.9 development will follow a mostly waterfall method. Development will start with the set-up and installation of all the required dependencies and requirements. Installation involves setting up the hosting and back end server so that requests can be processed and sent onto the database were new data can be created and pulled.

This set up also involves integrating version control. Version control is a very important aspect of any large development project. In this case the project will be utilising GitHub to manage all the version of the application. As Codex is developed those developments will be pushed to GitHub where those changes can be managed and integrated into a public build. On top of that GitHub can be used to revert any changes that have unforeseen negative consequences. Version control is an incredibly useful tool that allows the easy management of large project through the logging of development commitments. Hence why it will be heavily utilised in this project.

## Implementing the Voting System

A system which contained a surprising amount of complexity was the voting system. The difficulty arose when it came to ensure that users could only vote once. Creating a system that allowed users to vote many times was very simple but allowing the user to only vote once and to enforce that became tricky.

Any attempts to stop the user voting multiple times on the front end would be useless as it would be very easy for a malicious user to circumvent those barriers. So, an approach that validated the user’s vote on the backend would be required. In the end a log approach was implemented.

Viewing figure *bleh* you can see that the post object contains several lists called <attributename>log. These are lists of user ID’s who have voted on a post. When a user votes their unique ID is added to that list. That paired with front end preventions can prevent users from maliciously voting multiple times. On the front-end a check is passed that disables the functionality of the up-vote button after a single press. This could be circumvented by manually editing the page but that would not allow the user to make multiple votes due to the server-side check. If the user re-enabled the button and submitted another up vote that vote would then be discarded by the server when it checked the vote log and found that user ID is already attached to that post.

The downside of this approach is that it increases the complexity of the post object and therefore requires more memory to store each post object. It also means that at larger sizes voting may take longer as the system checks for a user ID within a huge list of user IDs. If there was more development time was available this system could be improved by creating a user object and attaching the post ID to a list within the user object. That means when making a upvote the server would check a shorter list on the user to make sure they weren’t voting multiple times.



The above figure is the proposal for the new user object. If there was more available development time this user object could be implemented and whenever they voted the post ID would be added to the users list. This would avoid the possible huge list of user IDs currently present in the current application. These huge lists could be a potential performance bottleneck so if development were to proceed this implementation should be taken forward.

## Implementing the Post Tagging System

Another tricky aspect is that of the post tagging system. Due to the current layout of the Codex application posts do not have unique address. This was done to close certain avenues of attack. With open external links it opens the possibility of malicious users attempting to provide invalid address that have the possibility to send corrupted data to the server. Therefore, in and effort to increase security individual posts were not given unique links.

Unfortunately, that means that tagging posts becomes somewhat difficult as users cannot simply copy and paste a link into their posts. In future development it might be worth using a hashed version of the post’s unique ID as a HTML GET request to provide a unique link to the post and provide that functionality. However, as currently implemented a simplified version of the search component is included in the post creation suite. Here users can search for posts and instead of viewing them they can tag them. This will then connect the tagged posts ID to the new posts tagged post list. This is then stored in a string array and attached to the new object.

http://www.codexapp.reivew/home?post=abcdefg123

If unique post links were to be implemented, they would be implemented through get requests as you can see above. This does however open the site up to various attack such as editing the data on the user end. This could possibly lead to malicious users attempting to query data they are not meant too. That would be why the site would have to carefully vet the hashed post ID before processing it. If the key was invalid the user would be returned to the home page and given an error.

It will be a point of note in the final audience survey to see how users react to the post tagging system. There is a chance they would prefer to just include links in there post in which case the above method would have to be used. However, they may also like the post tagging system as it does all users to search for posts while remaining in the post creation suite. So, a combination of the two approaches might be best for the application. There is no way of concluding on this issue until the final audience survey is completed.

## Implementing the Search and Sort System

As mentioned throughout the Market Research *(chapter 2.1)* and the audience research *(chapter 2.2)* the ability to quickly and easily search for specific content is incredibly important to the average user. That is why in Codex it must be implanted thoughtfully and be extensive in its usage. Plenty of tools and options must be available to the user to allow them to search through a possibly huge range of content to find something specific to their needs.

The above figure shows the final design of the search page. Here the user has multiple options of how to search and how to order the data they receive. Linking to Stack Overflow which provided to be the most popular tool in the Audience Research *(chapter 2.2)* having a search function which queries multiple properties is more popular. That could be down to the fact it makes it easier for a user to find specific content which they may not fully know how to search for.

In the of the codex search bar it will look for partially linked tags to post titles. It will also at the same time look for author names. On top of that it also makes the search case insensitive. One issue that can appear is content not being displayed because of formatting. Search results can be greatly increased by making it as simple as possible for users to find content via known tags. The language is a separate field as a syntax or semantic may be present in multiple languages. However, users may only care about that syntactical feature in a single language. This feature gives that user to search for a topic then limit it down to a single language. Once again giving the user more choice and power when searching for specific posts.

Finally, we have the sort feature. This is a new feature to Codex that hasn’t been directly implemented in any other major applications currently available. Other sites like Reddit have a similar feature when users can sort based on “controversial” or “hot”. These are not perfectly defined elements though with their underlying algorithms being hidden from the user. The goal with Codex though is to make that functionality clear to the user. If a user sorts based on conciseness they can see the conciseness rank and easily understand why one post was sorted other another post. This function the same with the other criteria too.

## Implementing the Real Time Data Aspect.

These huge crowd sourced sites when they finally develop a community are constantly being updated with new content every second. YouTube for example is estimated to be updated with over 300 hours of video every minute *[YouTube-Press 2018]*.While it is highly unlikely Codex will receive that level of traffic it is an example of how quickly content can be produced. So, with that in mind the application needs to be able to serve the user new content as and when it is created.



This Real Time Data is enabled through the Polyfire library. Within that library there is a component called firebase-query. The Query component is its own self-contained collection of HTML5 and JavaScript will directly communicate with the Firebase database. Starting at the top attribute of the element the ID tag works the same way as a normal HTML ID attribute. It allows for easily access by JavaScript. This in turn allows the easy alteration of the path and data attributes. Speaking of which, the path attribute refers to the location within the Real Time Database. Finally, the data attribute is the location of any pulled results. This is then exposed to the Polymer databinding using the double curled brackets. Meaning when the data is updated the Polymer engine will be notified allowing other elements access to that data.

The real power behind this system is that all this complexity is given at a great level of abstraction. That means it does not take very long to get complex real time functionality up and working. Using this element alongside some simple JavaScript and Polymer data bindings. The application now can query the database and self-update when it detects a chance.

Once again, this was implemented to make a truly responsive system. As users create posts that content will be immediately pushed to other users creating the feeling that the site is constantly changing. This does however have a minor set back that the site must regularly check for updates to the database. That means the application may use slightly more bandwidth than other apps. That is offset though by the Realtime functionality and that most browsers will auto limit traffic if the user is on a limited bandwidth.

## Final Deliverable

The final site consists of just under 8,000 files. These include custom created components for the Codex application as well as third party components and modules required to quickly achieve complex functionality. Developing a system for querying a database in real time and then returning new data as it is created would take a rather long time for a single person development team. That is why third-party components have been relied upon to save time and focus effort on the system unique to the Codex Platform.

# Testing

## Black Box Testing

### Method

### Report

## Functionality Testing

### Method

### Report

## Miscellaneous Testing

### System Stress Test

### Anything Else That Comes To Mind

# Evaluation

## Audience Survey Results

## Focus Group Testing

## Project Requirements

## App Reviews

# Conclusion

## Overview

## Aims and Objectives

## Summary

## Future Improvements

# References

* *(Winslow 1996) –* “Programming Pedagogy – A Psychological Overview”, Leon E. Winslow, ACM SIGCSE Bulletin – Volume 28 Issue 3, Published: September 1996, Pages 17 to 22
* *(Allan G. Bateson, Ralph A. Alexander, Martin D. Murphy 1987)* – “Cognitive Processing Differences Between Novice and Expert Computer Programmers”, Allan G, Bateson, Ralph A. Alexander, Martin D. Murphy, International Journal of Man-Machine Studies – Volume 26 Issue 6, Published: June 1987, Pages: 649-660
* *(Wikipedia*-Myspace 2010) – Wikipedia Myspace (2017) Available at: <https://en.wikipedia.org/wiki/Myspace> (Accessed: 12th October 2017)
* *(Luis von Ahn 2008) – Luis von Ahn, Laura Dabbish ‘Communications of the ACM Volume 51 Issue 8’ (August 2008), Pages 58-67*
* *(Zwass 2010) –* “Co-Creation: Toward a Taxonomy and an Integrated Research Perspective”, Vladimir Zwass, International Journal of Electronic Commerce – Volume 15 Issue 1 Published 2010, Pages 11-48
* *(Geiger 2011) –* ‘Crowdsourcing information systems: a systems theory perspective’ *In proceedings of the 22nd Australasian Conference on Information Systems (ACIS 2011)* Sydney, Australia, Sydney: ACIS Pages 0 – 12
* [rigaux.org](http://rigaux.org/language-study/syntax-across-languages.html#VrsMnlMmrllc) (2017) Available at: <http://rigaux.org/language-study/syntax-across-languages.html#VrsMnlMmrllc> (Accessed: 12th October 2017)
* (YouTube-Press 2018) – Google, YouTube (2018) *YouTube Press* Available At: <https://www.youtube.com/yt/about/press/> (Accessed: 06th April 2018)

# Appendices

## Appendix A: 1st User Survey Results

## Appendix B: A Load of Crap