Chrome Extension Web Clipper and MEAN Web Application

Lauren Lewis
Supervisor: Professor Alan Dix
School of Computer Science
University of Birmingham

A dissertation submitted for the degree of
Master of Computer Science

September 2015

The material contained within this thesis has not previously been submitted for a degree at the University of Birmingham or any other university. The research reported within this thesis has been conducted by the author unless indicated otherwise.

Signed ....

Acknowledgements

First and foremost I offer my sincerest gratitude to my supervisor, Professor Alan Dix, who has supported me throughout my dissertation project with his wise advice and motivation. I would also like to thank Nadeem Shabir for offering me chance to partake in the internship at Talis; but mostly for being my mentor for the duration of the project and for his patience and knowledge whilst allowing me the room to work in my own way. I would also like to extend my gratitude to the Talis Developer Team for always having time to answer my questions and not mind being my targets for nerf-gun practice.

Abstract

This project was completed as part of a ten week internship at Talis, an educational software company. The brief outlined by the company proposed that the project should explore multiple areas, these are: how to develop a browser plugin that would allow students to capture content from any web page, a web application that would allow the students to login, organise, share and discuss the content captured, and how these two applications could make use of Talis' existing microservices. For the plugin, a Chrome Extension was built. The extension allows the user to select an area of the web page; this area is then captured and sent to the web application. The web application built is a full-stack single-page web application built using a framework consisting of Mongo, Express, AngularJS and Node.JS. The three Talis microservices were Persona for authentication, Babel to store annotations and Depot to store and retrieve documents. The web application allows users to login, view and comment on their own clips, create groups and share content, and view their activity feed. Automated tests were attempted, but not fully completed. Instead, manual browser tests were carried out and company feedback was gathered.

Keywords: Chrome, Extension, Plugin, Web, MEAN, Javascript, Education.

Contents

List of Figures

Chapter 1

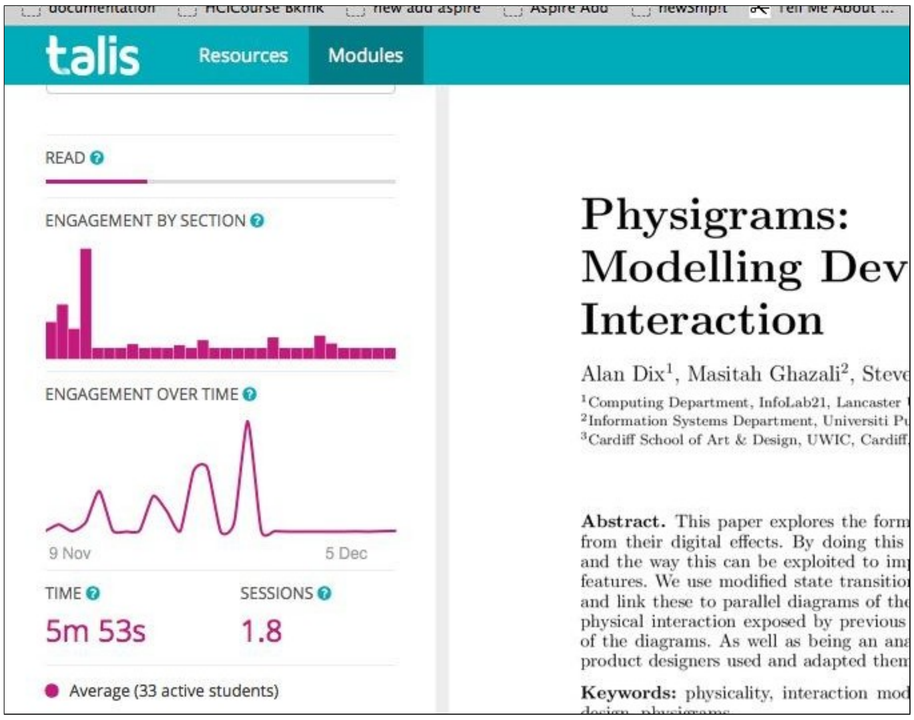
Introduction

The aims of the work described in this Report were to build a tool that could allow students to capture content from any web page, then be able to share and discuss the content that was captured. The work should demonstrate how such a tool could be integrated into *Talis'* own software in future development. This work was completed as part of an internship at *Talis*, a high tech educational software company in Birmingham. The project brief can be found in Appendix A. The brief specifies that the tool should also make use of *Talis* microservices. The software that the tool would be integrated with is called *Digital Content* and is part of an internal project at *Talis* called *Lighthouse* [33]. The software is currently being piloted in many universities around the world. A brief description of the software can be found in Section 1.0.1.

The work done for this Report was to provide a proof of concept to how this tool could be developed and integrated into existing software. It was developed with the intention of exploring what could be feasible and not as product to be deployed. In addition to the tool created to capture content, the Report will describe the work involved in building a small web application. The application was developed to demonstrate how the company's microservices could be incorporated into the system. The web application will be developed as a Javascript full-stack single-page application similar to existing products provided by the company.

1.0.1 About Talis Digital Content

Digital Content is a software that can be integrated into the university's existing virtual learning environment. It provides a service that allows lecturers to upload any type of digital content, such as PDF, video or slides. The documents can be viewed in a universal player that provides the ability to highlight, annotate and search. There are two modes, class view and individual view. This allows the student to add private notes or work collaboratively on a document. The software is available on a range of devices and synchronises between them.



Physigrams:
Modelling Dev
Interaction

Alan Dix[1], Masitah Ghazali[2], Steve

[1]Computing Department, InfoLab21, Lancaster
[2]Information Systems Department, Universiti Pu
[3]Cardiff School of Art & Design, UWIC, Cardiff,

Abstract. This paper explores the form
from their digital effects. By doing this
and the way this can be exploited to im
features. We use modified state transitio
and link these to parallel diagrams of the
physical interaction exposed by previous
of the diagrams. As well as being an ana
product designers used and adapted them

Keywords: physicality, interaction mod

Figure 1.1: An example analytics gathered using Talis Digital Content.

It is also makes content available offline to allow students to work anywhere at anytime. What makes the software stand out from other existing products is the learning analytics it provides, for example determining students engagement by looking at a document's activity (See Figure 1.1).

1.0.2 About the Report

This Report will begin by exploring the motivations and considerations of building a tool to aid learning in higher education, and investigate existing products that have a similar focus. It will explain the background knowledge required to create a dynamic online software system. It will then describe how the project will be managed and clarify the requirements and specification of the software. The chapter on design and implementation will thoroughly explain the process executed to complete the clipping tool, web application and integration of *Talis* microservices. Finally, the software will be tested to ensure a stable prototype has been completed, then the project will be evaluated. A time-line of the project can be found in Appendix B.

Chapter 2

Related Work

This chapter will explore the motivation behind creating new technology that could be used in higher education to aid learning. It will examine what tools already exist and how they are used to create an effective working environment. It will also focus on the aspect of collaborative work flows and what obstacles may arise when individuals contribute to a shared document.

2.1 Literature Review

There are three main themes that will be explored throughout this literature review. The themes will be first personal information management (PIM), followed by a deeper look at information gathered by social bookmarking, then how these previous themes may relate to peer learning in higher education.

The research of PIM looks at how people interact with the vast amount of information they accumulate in their daily lives. Jones and Teevan[38] define PIM as:

"finding, keeping, organising and maintaining information. It is also about managing privacy and flow of information."

PIM research also investigates what tools are used to help with PIM, but it is often found that even though a tool may be useful in one area, it often can worsen a problem with information fragmentation [23]. This information fragmentation often becomes more obvious in group work, when individuals try to share their work when they've used a variety tools[1]. It would be advantageous for these new tools to be integrated into a single working environment that already existed and was familiar to the students.

The term *information* in the context of PIM could refer to paper documents or online material. Online materials can include emails or web pages. This review will focus on the gathering of web-page information, usually referred to as web bookmarks. The sharing of

[1]I've personally spent an entire day gathering resources from Google Drive, Facebook groups, Whatsapp chat and email before work could begin on a group report.

bookmarks is known as social bookmarking[31]. Wikipedia defines social bookmarking as a method for Internet users to store, organise, search and manage bookmarks of web pages on the Internet with the help of meta-data[5]. This meta-data is usually defined by the web page and may not be a useful decription for the student. Some tools allow users to add their own tags. Adding your own tags can be more meaningful and memorable, therefore making it easier to search for the information in the future.

One particular group of students who can benefit from social tagging includes those taking online courses[31]. By sharing information online they can create learning communities based on their course or interesting topics. It allows students to develop new insights about a topic by discovering the views and perspectives of others. It is discussed in Peer Learning in Higher Education[14] that collaborating through learning experiences is vital in e-learning. Interaction with other people encourages deeper learning and more active learning engagement. It also encourages peer-to-peer learning which can take some strain off the tutor in larger class sizes.

2.1.1 Summary of Literature Review

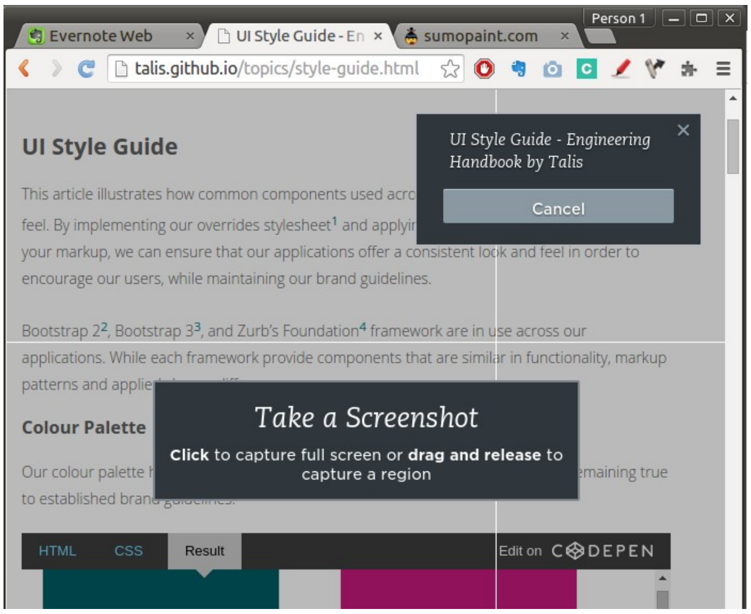
This review has emphasised that a product created to aid learning should integrate easily into the students existing learning environment. It should not disrupt the students flow of work and be easy to use. Also discussed is the importance of being able to readily form peer groups, and how discussing the information gathered can help develop a deeper understanding of a topic. Finally, it is important to note that adding tags or customising the content captured from web pages can make it more personal or memorable. This will, therefore, make it easier to find again when needed.

2.2 Existing Products

This section will investigate existing products that provide the ability to share information, annotate documents and capture content from the web. It will look at what features are similar to *Talis' Digital Content*, and additionally what other features are offered.

2.2.1 Document Sharing

Google provides three web-based services that allow you to create and edit text documents, spreadsheets and slides online [19]. Each document created or shared is retrievable from the user's *Google Drive*, an online storage service. The documents can be organised into folders. A document can be shared with other users individually or whole folders can be shared. Within a single document a user can add annotations to the page or hold discussions in chat box. The entire application is synchronised to allow easy collaboration between users. It's



UI Style Guide

This article illustrates how common components used acro feel. By implementing our overrides stylesheet1 and applyi your markup, we can ensure that our applications offer a consistent look and feel in order to encourage our users, while maintaining our brand guidelines.

Bootstrap 22, Bootstrap 33, and Zurb's Foundation4 framework are in use across our applications. While each framework provide components that are similar in functionality, markup patterns and applie

Colour Palette

Our colour palette maining true to established bran

Take a Screenshot

Click to capture full screen or **drag and release** to capture a region

Figure 2.1: Evernote Web Clipper

a successful application for sharing documents and is used by many professional companies for this purpose.

Evernote is another popular online document editor [17]. *Evernote* appears to place more emphasis on the individual user and less on the collaboration of work. Evidence for this can be seen in the amount of functionality and ease of creating and collecting documents for a user's own collection. It seems less straight forward to share and edit the same documents compared with *Google Docs*. *Evernote* does not provide the ability to add annotations, discuss within an individual document and it is not synchronised between two users on the same page. However, what makes *Evernote* standout from the crowd is it's *Web Clipper*, which will be discussed in the next section.

2.2.2 Capturing Web Content

The *Evernote Web Clipper* [16] is what makes *Evernote* a popular web application to gather notes from the web. The *Web Clipper* is a browser plugin that has several actions available to the user, these include: capturing the entire article (grabbing the html of the page), a whole page screen shot, a simple bookmark or a screen shot clip. Once a content has been captured, the user has the opportunity to annotate the image and add descriptive tags. However, it was noticed that to view the new clip immediately the web application had to be refreshed- a small inconvenience.

Clipular [12] is another browser plugin that captures content from the web. *Clipular's* plugin works in a similar way the the *Evernote clipper* (See figure 2.1); when the user clicks a button to start clipping, the page is greyed out and the user can use the mouse to draw out a rectangular area to capture. Once the clip is captured the clips are sent to the application. *Clipular's* web application is not as advanced as *Evernote*, so it will not be discussed in detail. It does allow clips to be grouped into folders, but does not support sharing within the application. *Clipular*, however, did have an additional feature that had not been observed in other products - intelligent search. It's not fully understood how the product searches through clips, but it seems that is able to search the text contained in the clips.

An alternative product that uses an intelligent search for content captured from the internet is *Snip!t*[1]. Snip!t has been developed as a bookmarklet instead of a browser plugin. Bookmarklets are just like normal browser bookmarks but when you click one, instead of taking you to another web page, it performs some action within the current web page. Snip!t captures text from a page and *'intelligently'* detects words such as ISBN numbers, postcodes or dates. Given the highlighted words it can suggest actions, such as view the postcode on a map.

2.2.3 Summary of Existing Products

All of the products in this section were chosen for their popularity or a feature that they offered was unique. They all were easy to use and self explanatory ; they also provided some kind of organisation structure. Individually they all had features that worked successfully and allowed them to stand out in the market. *Google Docs* success was due to the ease at which you can collaborate work on a range on documents. *Evernote* had a wide variety of options for gathering content from the web, and clean, simple structure for organising the information. The other two, *Clipular* and *Snip!t*, even though they were not as well known, provided an interesting insight on how information can be tagged or searched in order be useful for future studies.

Chapter 3

Background

This section will describe the project software from a top-down perspective. It will start with a general overview of distributed systems; following this it will focus on the advantage using a microservice architecture and then the role of the a single-page application. It will then discuss in a little more detail why Javascript and NoSQL are the preferred technologies for web development.

3.1 Web Distributed Systems

A distributed system [6] is a software system in which components in a network communicate and coordinate in order to achieve a common goal. Most modern technology companies will use a type of distributed system; usually separated into three tiers: client, server and data. In the time available for this project only a small system will be developed, even so, it is sensible that some time should be spent in understanding the considerations and trade-offs encountered by big websites. The main goal of a distributed system is to make it easy for users to access remote resources, and share them in a controlled way. It makes it possible for users to collaborate, exchange information and work together.

When using a distributed system security must be a key concern, especially when sending users private information across a network. The precautions taken for security and authorisation will be explained in more detail in the Chapter 6, when discussing the design and implementation of the Chrome extension and web application.

Another property of distributed systems is that they should be designed to be open and allow easy interaction between other open systems. This leads us onto microservice architecture that will be described in the next section.

3.2 Microservices

Microservices are small autonomous services that work together for a common goal [29]. They are designed to exist as their own separate entity that change independently from any other. They should be designed to have only one task and be very efficient at completing that task. The advantage of using microservices is that it allows the developer to pick the right tool for the job, rather than a more standardised, one size fits all approach. However, they do require more overhead work to design and implement; but once developed they are easier to maintain. They can easily be reused in other applications, upgraded or swapped for more updated software. They are a good architecture to choose if you are working with new technologies as they provide protection against bugs cascading through the entire project. When debugging the application only small amounts of code will be affected which will also save time by not having to test the entire system each time.

Another advantage of using a network of services instead of one large monolithic program is that when you come to scale you have the ability to pick and choose which area of the system to expand. It is also easy to keep the software updated as you can release smaller builds (single services) more often, and they are smaller enough that it is not too big a task to have to rewrite one.

The requirements for this project specify that *Talis'* microservices should be integrated in the web clipping service. These will be explained in more detail in the Chapter 6.

Several third party Javascript libraries will also be used in this project, for example *Chrome** API. Libraries have similar properties to services in a way that their complex code is encapsulated with a clear and well defined interface. They are purposely designed to be reusable code; this should be kept in mind while the design of the project is in progress.

3.3 Single Page Application

Many large web companies such as *Github, Gmail* or *Talis* use single page web applications. This is *Wikipedia's* [4] definition of a single-page application:

"A single-page application (SPA), also known as single-page interface (SPI), is a web application or web site that fits on a single web page with the goal of providing a more fluid user experience akin to a desktop application."

Compared with a traditional web site, a SPA does take more time to initially download, however, the user will not have to wait for any page to load again during their visit [36]. Resources used by the web page are dynamically loaded when needed, usually in response to users actions. As discussed in Chapter 2 it is important to develop a application that is simple and requires no effort for the user.

3.4 Asynchronous Communication

To achieve a seamless dynamic web application the user should be able to continue to use the web page without having to wait for it to load [9]. This can be accomplished by communicating with the server asynchronously; this means that the application does not have to stop what it is doing and wait for a response. Instead it passes a callback function to handle the request when it is complete. The messages sent should also only try to retrieve a small amount of data that is being modified and not the entire web page.

Javascript is a programming language developed for the web. Javascript has a special property in the fact that functions are considered as normal objects. This means that functions can be passed as arguments or returned at the completion of methods [32]. This ability to pass a function as a callback make calling methods asynchronously possible.

Javascript will be used on both the front-end and back-end of the application, for more information see Chapter 6.

3.5 NoSQL Database

NoSQL document based databases are another popular trend emerging within large web companies. The appeal of document based databases is that they are easy to scale horizontally [27]. This means that instead of having to upgrade the server to hold more data, the database can be spread among a horizontal network of servers. NoSQL document based databases consist of a set of collections that hold a set of documents; this compares with a table and rows in a relational database. A document is a single entry, and has a similar structure to JSON (Javascrpt obejct). Unlike row entries in a table, documents in a collection do not have to have the same schema. This is useful because if the software involves and the data changes, the database can adapt without having to be re-designed. NoSQL does not support the join operation used to select data from two collections, which can often be a costly operation. To improve the efficiency of data retrieval, related data should be stored in the same collection. The disadvantage of this is that data may need to be duplicated in more than one collection and inconsistencies may occur. If the main operations used on the database are *GET* or *POST* and not *UPDATE* then this is an acceptable trade-off for performance.

Chapter 4

Project Management

4.1 Risk Analysis

During the development of the scope of the project the following points were recognised as potential risk factors.

- Computer malfunction or failure.
- New Technology.
- Unforeseen time delays.

Care was taking when planning the duration of the project to help reduce and prevent these risk factors. The initial step was to set up a reliable backup service to protect against lost work. Github, a version control tool, was chosen for this purpose as the developer had past experience using it, and it could easily be shared with the Supervisor. The highest risk to consider is the fact that all technology proposed for this project is new for the developer. It is key to the project that enough time is set aside for learning. The risk was taken with the knowledge there would be a steep learning curve; however, as the developer would be working in a professional environment with informative and supportive colleagues it was worth taking. The new technologies were also chosen because of their popularity with the online community, this meant that were plenty of examples and well tested documentation. The incremental software engineering approach was also chosen to support the risks involved in the project, as it allowed the implementation to be broken down into fortnightly builds. If a problem did occur the most recent build could be presented as a completed prototype.

4.2 Approach

After some consideration, it was decided that the incremental software engineering model[13] was most appropriate life cycle for this short project. Others, such as the waterfall model,

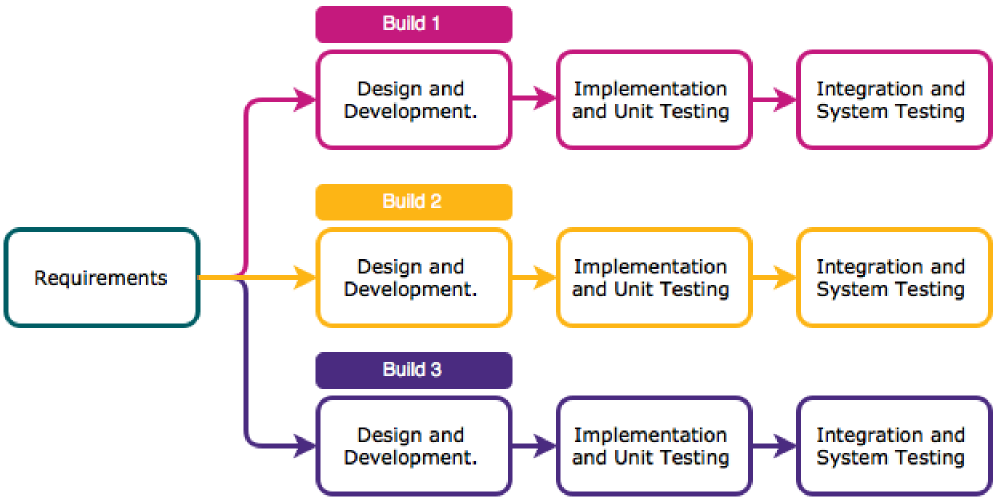

Build 1
Design and Development.
Implementation and Unit Testing
Integration and System Testing
Build 2
Requirements
Design and Development.
Implementation and Unit Testing
Integration and System Testing
Build 3
Design and Development.
Implementation and Unit Testing
Integration and System Testing

Figure 4.1: Incremental Life Cycle

were rejected due to being too rigid in design and implementation, which would not work in the case where the technology is not fully understood. On the other end of the scale some were rejected because they were not rigid enough, as in the spiral model in which an end is not known. Planning and design must be considered as high importance in a project with such a tight time constraint.

The incremental model divides requirements into several builds, with easy to manage modules[8]. Figure 4.1 shows a schematic representation of the incremental model. The advantage of working with modules is that if part of the system fails, or contains bugs, the problem is self contained and does not cause a ripple effect through the entire project. It also means that modules can easily be replaced if needed, improved independently or reused in other systems. The life cycle also easily supports quick integration of third party system that will be large part of this project.

As the project is broken down into several builds it allows time to learn new technologies as the project progresses. It also provides functional software quickly and early in the life cycle, which, in such with a tight time frame gives some security in case of unforeseen delays or circumstances in which the work cannot be completed.

The disadvantage of this model is that from the beginning a good clean set of requirements is needed so that a concise plan can be developed. In the case of this project the requirements given were slightly vague, so more responsibility was given to the developer.

In reflection, this model was a good choice for this project, however, a few features had to be adapted as the project progressed. The focus for the first build was to build a solid framework that could easily be added to in later builds. Due to lack of experience with the technology, previous builds had to be revisited and re-factored as skills were developed.

Testing stages were quite weak until the relevant skills were acquired, and again previous builds had to be revisited. Even though a few overlaps occurred within the time line, all milestone were reached in the proposed times.

4.3 Management

Organisation and management were a key focus in this project. Working in a professional environment encouraged the use of shared open resources, so any drift from the current plan could quickly be noticed and rectified. *Evernote*[17] was used daily to record progress, save reference material and develop a history of the project. *Trello*[35] is an organisation tool that allows you to define easy to reach goals and record progress. *Github*[20] is a version control system which was used regularly to commit changes to the code.

Figure 4.2: Organisation and Management Tools

Chapter 5

Software Requirements Specification

This chapter will discuss the functional and non-functional requirements of the software that is being developed for this report. It also includes use-cases that describe the expected interactions between the users and the software.

5.1 Functional and Non-Functional Requirements

Table 5.1 lists the functional requirements are a criteria for the functional behaviour expected of the system. They describe *what* the software is supposed to do. Each requirement is given a priority defined by MoSCoW [7] analysis. There are four categories of priority: must, should, could and want. The requirements labeled *want* will only be implemented if time is available.

Table 5.2 lists the non-functional requirements for the project. These requirements describe the constraints of the system and how the operations should perform. They describe *how* the system is suppose to be. These have been categorised into what type of quality of the system they define.

5.2 Use Cases

Figure 5.1 shows a simplified model of the use case diagram for the software. The use case shows what actions the user can do with the software. Instead of representing every possible use case that may occur in this scope generalizations have been made. The first generalization is for the three main objects used in this project: the clip, an annotation and a group. Each of these three objects should support CRUD [26] operations, where CRUD stands for Create, Retrieve, Update and Delete. One function that is not covered by the CRUD operations is the action to share a clip with a group, thus, this has been included

Table 5.1: Functional Requirements

No.	Requirement	MoSCoW
1.0	The system shall allow the user to log-in.	M
1.1	The log-in system shall support the Institutes email address.	M
1.2	The system shall allow a user profile to be updated.	W
2.0	The system shall allow the user to capture content from any web page.	M
2.1	The system shall allow the user to select an area of the page to capture.	S
2.2	The system shall allow the user to modify the position and dimensions of the clip before capturing.	W
2.3	The system shall allow the captured content to be viewed.	M
2.4	The system shall allow the captured content to be deleted.	C
3.0	The system shall allow a user to create a group.	S
3.1	The system shall allow a user to leave a group.	S
3.2	The system shall allow a user to add another user to a group.	S
3.3	The system shall allow a user to share a clip with a group.	S
3.4	The system shall allow a user to remove a clip from a group.	C
3.5	The system shall give the option to search through the clips.	W
4.0	The system shall allow private comments to be added to a user's clip.	C
4.1	The system shall allow public group comments to be added to shared clips.	S
4.2	The system shall allow a comment to be deleted.	W
4.3	The system shall display past comments in time order.	S
5.0	The system shall provide a record of user and group activity.	W
5.1	The system shall notify the user of group activities.	W
6.0	The system shall display analytics on the frequency of activity of a clip.	W

Table 5.2: Non-Functional Requirements

No.	Requirement	Type
7.0	Capturing content from a web page should be done with a minimal number of mouse clicks.	Usability
7.1	If a page is waiting for a process to complete a loading page should be displayed to indicate the page is still working.	Usability
8.0	Talis Persona authentication service should be used to authorize a user.	Security
8.1	To reduce the risk of cross-site request forgery each request to the server must include an Persona access token to authenticate the user.	Security
9.0	The system should always have all comments saved and available to view when requested.	Dependability
9.1	The system should never remove (or lose) a clip unless a user requests it.	Dependability
10.0	A clip that has not been shared should be only available to the author.	Privacy
10.1	Private comments on clip should not be able to accessed by unauthorised users.	Privacy
10.2	A user's profile should be private.	Privacy
11.0	The web clipper should respond to mouse and keyboard events without obvious lag.	Performance
11.1	Comments that are available to a group should appear in the text box without obvious lag.	Performance
11.2	New clips should appear in the web application in less than two minutes.	Performance
12.0	Consideration should be taken for scalability for the next prototype.	Scalability
13.0	If the state of the page renders a button unusable it should be disabled.	Fault Tolerance
13.1	Where possible reduce circumstances in which a user would type an input.	Fault Tolerance
13.2	If a user is to type input, then input should be regarded as a possible threat and should be checked.	Fault Tolerance
14.0	Code should be well documented for maintenance and resources referenced.	Documentation
15.0	The code should be divided into easy to manage modules for testing and reuse.	Testability
16.0	While developing the web clipper for chrome considerations should be made to reduce dependability on chrome* API so that it will be easy to transfer to other browsers.	Extensibility

as a separate use case. The second generalisation involves the databases (Local, *Depot* and *Babel*) used in the project . Again, in cases of functionality they do not differ too much and have been grouped into a single actor. The user object will also have CRUD operations involved behind the scenes, but the user will interact differently than the previously mentioned objects; therefore, use cases involving the user object have been separated.

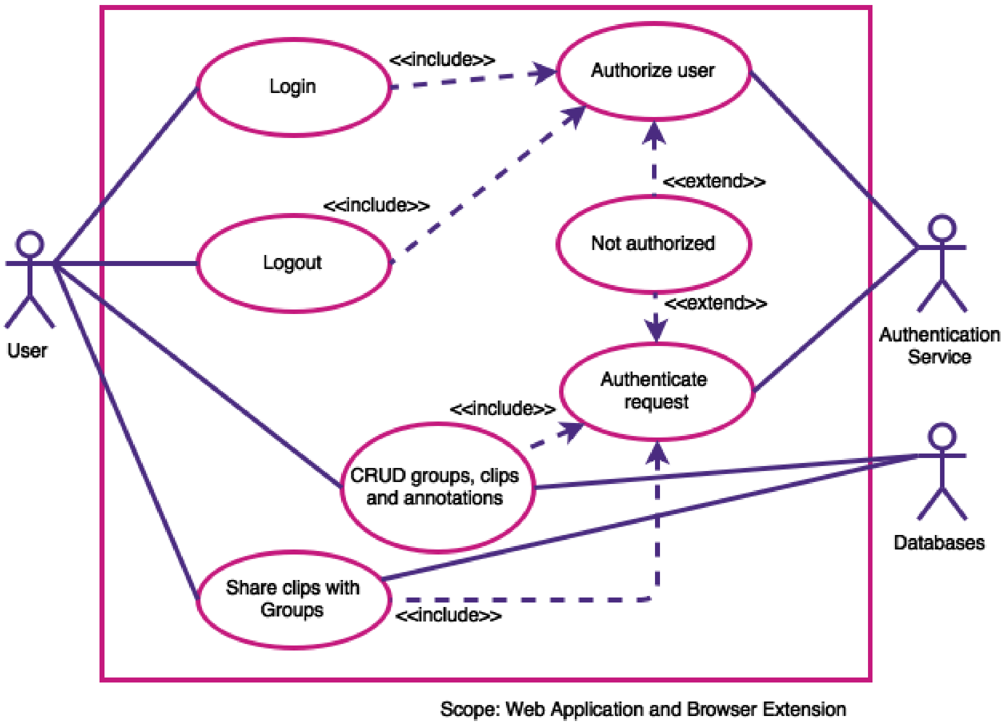


Figure 5.1: Project Scope and Use Cases

Chapter 6

Design and Implementation

This chapter will begin by describing the overall view of the system developed for the project and how it works as an integrated system. It will then be followed with a more in depth description of the *Chrome* extension. Then a build by build description of the web application.

6.1 System Overview

The entire system that was developed for this project consists of three main components; the *Chrome* extension created to capture content from web pages; a full-stack web application to allow users to view, share and discuss the captured content, and finally the integration of the three microservices provided by *Talis*. Figure 6.1 shows the high level structure of the architecture for the entire system. For a more detailed view of the web application see Appendix C for component diagrams of the client and server.

The *Chrome* extension, is small piece of software that can be uploaded to the *Chrome Browser* and used when browsing the web. It uses *Persona*, *Talis'* authentication service to authenticate the user. Once logged in and authorised it allows the user to select part of the web page to capture and then sends the clipped content to the web application. The web application handles the upload and stores the clip in *Depot*, *Talis'* document storage and retrieval service. It also authenticates the user using *Persona*, and uses the access token provided to authorise requests between the client and server. Comments and logs are stored in *Babel*, *Talis'* annotation microservice.

6.2 Chrome Extension

The extension was broken up into three builds to support the learning of new technologies. The aims of the first build were to incorporate the functional requirements 2.0-2.4. It concentrated on creating a basic extension that involved injecting a script into a web page

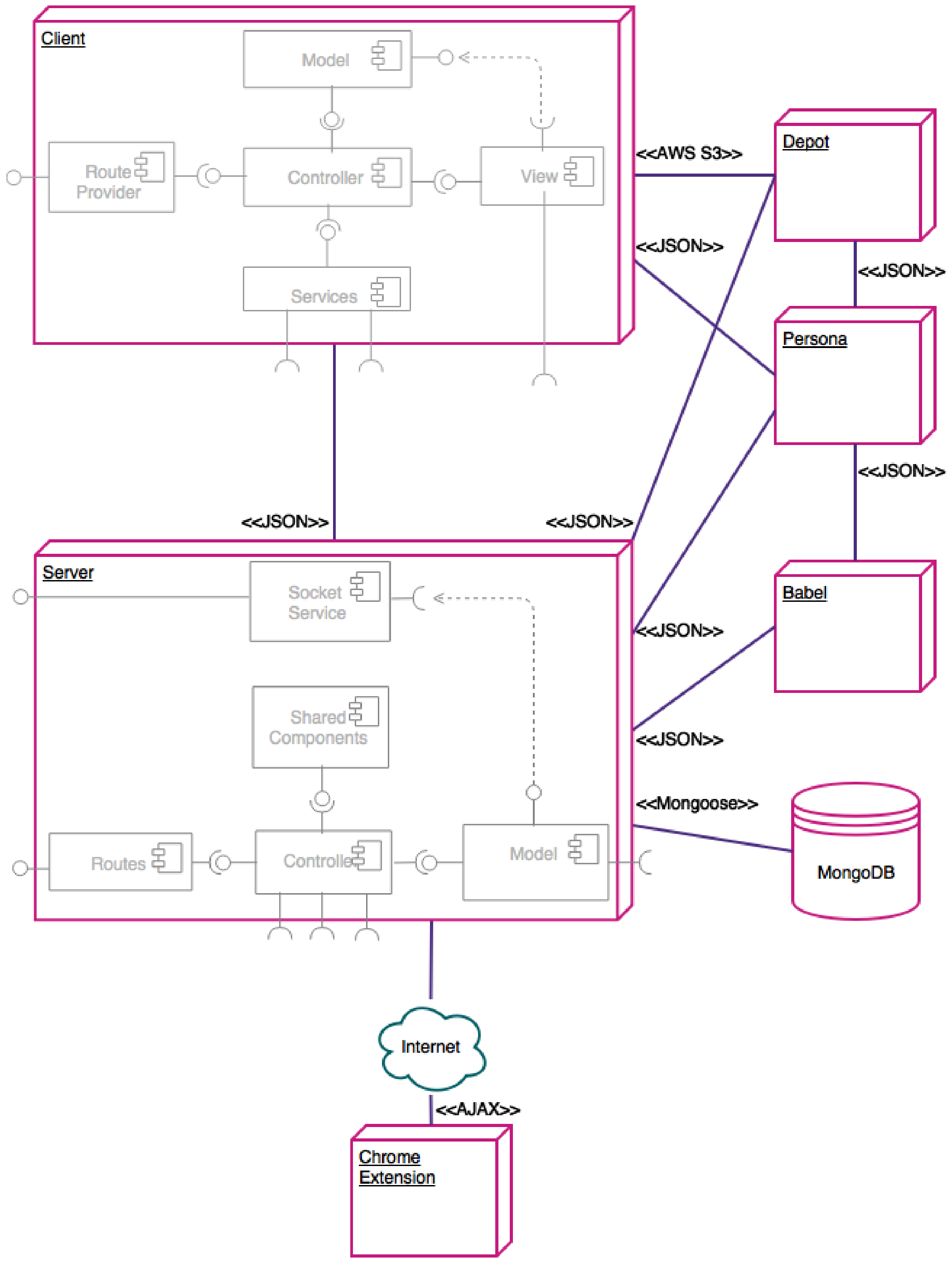


Figure 6.1: Deployment Diagram

to allow the user to define an area to capture. *Chrome** API was introduced in the first build to capture the content from the web page and to send messages between the extension's other scripts. Finally, communication was established with the web application. The second build, looked at establishing functional requirements 1.0-1.3. This build involved integrating a log-in page and authorisation. The final build then concentrated on making existing features more efficient, and improving the user experience.

A *Chrome* extension is a small application that can modify or enhance the functionality of a chrome browser. The application is written in HTML, CSS and Javascript, then packaged into a *.crx* file which can be uploaded by the *Chrome Browser*. Each package must consist of a manifest file, a HTML file, and optional Javascript files. Figure 6.2 shows the scope for each type of file and descriptions of these different types will follow.

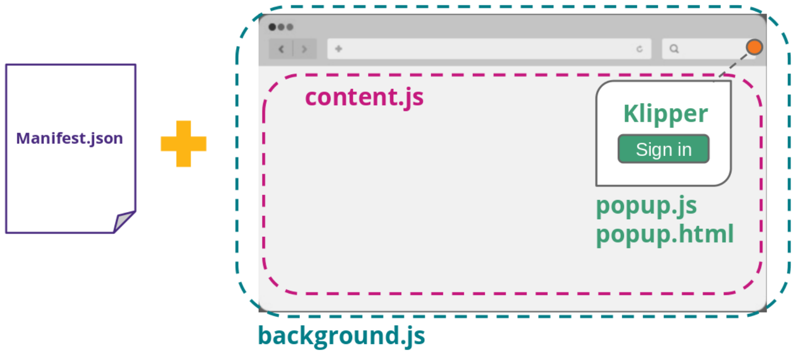


Figure 6.2: Chrome Extension File Types

6.2.1 Manifest

The manifest is a JSON file containing the metadata for the extension. This includes the version, properties, permissions and references to the other files in the extension. The manifest is mandatory and must follow strict guidelines in order for the extension to run properly. For the web clipper the popup html file and the background script were declared in the manifest, the other scripts are loaded when needed. The permissions needed were http/https so that the web clipper was available on any webpage, and tab[22]/ activeTab[21] to allow access to certain *Chrome** API.

6.2.2 Browser Action/ Page action popup

This is the little window or popup opened from the *browser action button* (Seee Figure 6.2 for location of *browser action button*). The background script listens for events that occur when the user clicks a button. For the purposes of this project it will have limited functionality and only allow the user to *login/logout* and *start clipping*.

6.2.3 Background Scripts

A background script runs in the background of the browser and has no access to the content of the page. The background script is the only script that can access the more advanced *Chrome** API. It has the ability to access a browser's tabs, history or favourites. The background script is used to *control* the behavior of the browser actions as described above.

Regular web pages can send and receive messages using XMLHttpRequests[11] but are limited to the same origin policy; where the origin is defined as a combination of URL scheme, hostname and port numbers. The policy prevents a malicious script from one page obtaining access to sensitive data on another web page. However, background pages can send messages if permissions are given in the manifest, so all XMLHttpRequests to an external server must be sent from the background script.

6.2.4 Content Scripts

The content script is a script that has been injected into the content of a web page [10]. A content script is written in Javascript and can read or modify the DOM (Document Object Model) displayed in the web page. A content script cannot modify the DOM of its parent background page, but must communicate by passing messages. Appendix D has two sequence diagrams that explain the process of message sending between scripts. The first scenario in figure D.1 describes the log-in process and communication with the external authentication service Talis Persona. The second scenario in figure D.2 explains the complex process of injecting a script into the web page, capturing screen content and sending a *POST* request to the server.

6.2.5 Chrome* API

Each web browser has different ways in which plugins or extensions are built. *Chrome** API are unique to Chrome, and allow the developer to easily control the browser and implement simple communication between scripts. Most of the *Chrome** API are asynchronous which reduces unresponsiveness and allows for a better user experience. If possible the use of *Chrome** API should be encapsulated into a single file so that if this extension was to be built for a different browser only that minimal changes would be needed.

6.2.6 Communication with the Web Server

Content scripts can communicate with cross origin servers by passing messages through the background script. The *Chrome** API offers two types of messages: one-time requests or long-lived connections. There is also an optional callback parameter that allows to handle the response. Sending file, such as an image, requires a more specialised type of object called dataForm. A dataForm object is handled differently to a normal JSON object because the request is handled as a multi-part object which is uploaded using a stream.

6.2.7 Authentication

The extension uses *Talis'* authentication service *Persona*. *Persona* is a microservice for authorization, identity and profiles. Communication with the service can happen either directly or using a registered application client. The extension is a browser based application and therefore explicitly trusted by the user and does not have to be registered with the service. To sign in the user is redirected to Institutes's sign in page. Once the user has authenticated with *Persona* the user will be redirected back to the application with the access token as a fragment in the url. The access token is then sent with the data to the server to authorise the request. As mentioned earlier in the report see Appendix D for a detailed sequence diagram of this process.

6.3 Web Application

The web application was built as a platform in which the proof of concept of the web clipper could be demonstrated. The choice of the technologies and framework were mainly prescribed by the company; the author accepted them with the intention of developing new programming skills throughout the duration of the internship. The following sections will describe the process of building the web application in a series of builds. Each build was designed to be achievable within two weeks; they are all of similar complexities[1].

6.3.1 Build One

This section will describe the motivations and decisions made in the first build. The aims of this initial build were to create a solid framework for the application to be developed on and investigate how the new technologies could be used. At the end of the first build the full stack web application was created; and a simple UI was developed to allow users to add

[1]Build one was the build with the steepest learning curve. And would not take two weeks if I was to build a web application again. But I went from asking "Where do website live when you are not looking at them?" to a working full-stack web application in two weeks.

comments to a single clip. This build tried to establish functional requirements 2.0-2.4 and 4.3.

6.3.1.1 MEAN stack

Yeoman [39] is a scaffolding tool for generating web applications, the scaffolding chosen for the project was the Angular-Fullstack. This created a *MEAN* stack which has a modular architecture consisting of an *AngularJS* client side, *NodeJS* with *Express* framework on the server and a *Mongo* database. The advantage of using MEAN is that there is a uniform language throughout the stack. It also means that the *JSON*-like objects stored in the database are essentially the same objects seen in the client Javascript too. The generated application comes with a small sample application, integrated tests and examples of routes between the components. The advantage of using this generated stack is that all the complicated routing between components is done automatically and development on the application itself can begin immediately; this was extremely important in this project so functionality could be developed in such a restricted time frame. Understanding the structure of a full stack is difficult if a developer is new to web languages. The learning curve can be quite steep at the beginning, however, once some functionality has been implemented it is easy to follow the same patterns to add more. Another helpful feature of using Yeoman and the MEAN stack is that there is a large active online community, many tutorials and well tested documentation.

Yeoman uses three main tools to generate the application: *Yo, Grunt* and *Bower/Npm* [37]. *Yo* scaffolds out the application and writes the build configuration for test, development and deployment modes. *Grunt* is a build service that allows you to use simple terminal commands to compile the application. In development mode a persistent connection is set up so that any modifications to the code is immediately reflected in the browser running the application. *Bower* and *Npm* are package managers to manage dependencies. These tools mean that the developer does not have to manually download libraries and there is a simple command to add them to the application.

6.3.1.2 Client

The client-side application was implemented using HTML, CSS and AngularJS. The design of the client-side is based on the MVVM [24] (Model-View-Modelview) architectural pattern. MVVM was designed to make use of the data-binding tool available from AngularJS. Data-binding [30] is an automatic way to synchronize the view and the model. The difference between MVC (Model-View-Controller) pattern and MVVM is that the flow of control is not cyclic and the model and view are not aware that each other exist, all control must pass through the view-model controller. Figure 6.3 shows the flow of control between components in the client. The controller and model have been represented as one component because of the amount of overlap caused by lack of experience using the MVVM model and will be discussed further in the evaluation.

The views have a minimalistic design and were not the focus of this project as the aims was only to demonstrate a proof of concept.

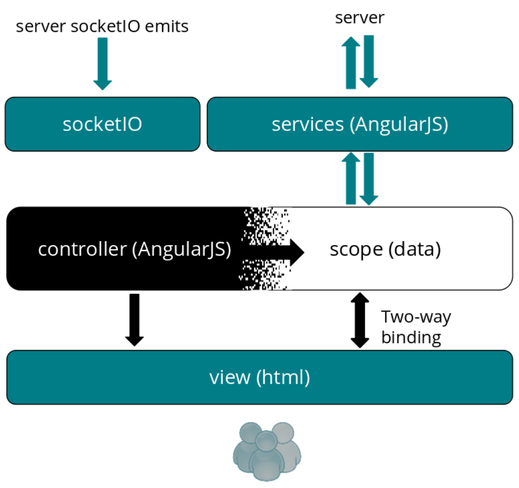


Figure 6.3: Client-Side Architecture

6.3.1.3 Server

The server-side was built using Node.js with an Express framework. Node.js is an an event-driven server-side Javascript environment built on *Google's V8 Engine*. Node.js uses non-blocking single-threaded I/O, this would not be ideal if you are using CPU intensive computation as it would block Node.JS' responsiveness. Express is a Node.js framework designed for building single page web applications. The most important feature Express

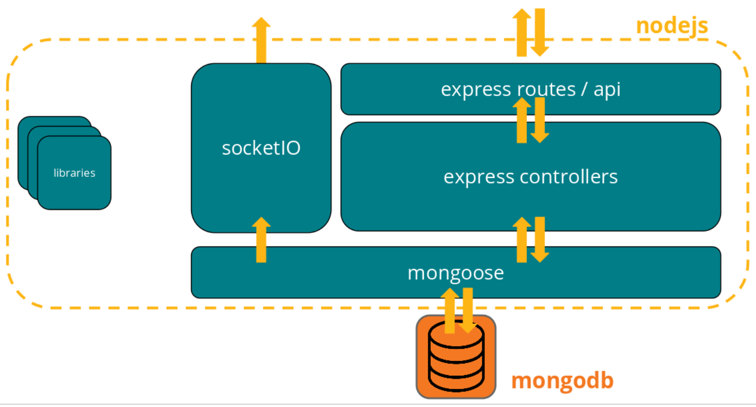


Figure 6.4: Server-Side Architecture

provides a simple interface for routing; it allows simple communication with the client using REST API. In addition to the Express library, the server also uses the Mongoose library. Mongoose provides an *SQL*-like API and easy to use schema to help model the data.

6.3.1.4 Database

A NoSQL document-based database was selected for this project as discussed in Chapter 3. The three main entities that needed to be considered were User, Group and Clip. Appendix F shows two Entity Relational Diagrams (ERD); the first displaying what the data needed to be stored in the database. In the second ERD the structure of the data has been normalised to reduce data redundancy. It was apparent that Comments should be extracted from Clip and Group to become it's own separate entity. This decision was made so that it would be easy to adapt the database in a later build to accommodate the integration of *Talis' Babel* annotation service.

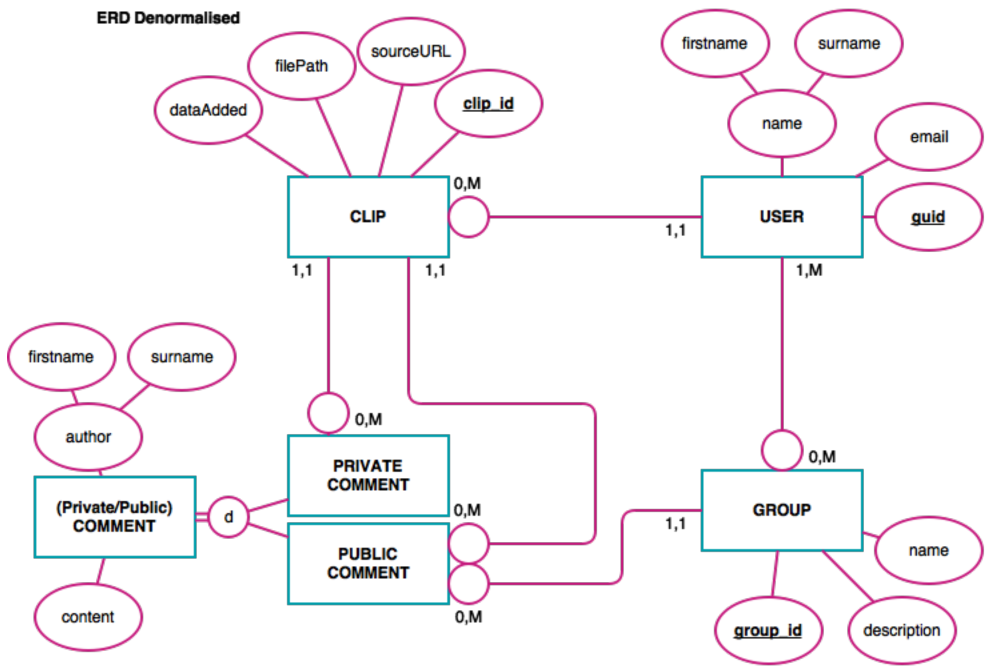


Figure 6.5: ERD after de-normalisation

Due to inexperience of using NoSQL the normalised design was built, but it soon became clear that this design was not suitable for this application. NoSQL database do not support relational joins, and to retrieve data often required more than one query. One example of this was when trying to retrieve a list of a user's clips, first you would need to query user-clip for the user Id, then a second query on clip to search for those clip Ids retrieved in the first query. So it was decided that the database design needed to be de-normalised.

The final de-normalised ERD can be seen in figure 6.5; the lines that connect between entities represent relations in a relational database, in a document based database these are structured as embedded documents. Embedded documents are just an array of ids relating to other documents. This structure was designed so that any request for data only required one query.

6.3.2 Build Two

Build two involved integrating *Talis'* authentication microservice into the application. This build was focused around introducing security into the software, both authorising the user and securing communications over the networks. At the end of this build the application should allow a user to log in using their Institutes's email address, the user should not be able to access pages or data they are not authorised to do so, and the application should protect itself against fake requests from malicious software. This build should encompass functional requirements 1.0-1.3 and 4.0.

6.3.2.1 Persona service

Talis' Persona microservice is used to authenticate users and supply tokens used to authorise requests to the server. This was a challenging part of the project as it required using documentation written only for internal use at the company. The documentation was often out of date and the only examples available were fully integrated into *Talis* software. The API used by the existing software was used sporadically throughout several layers of code, which was quite difficult to decipher for in inexperienced Javascript developer. Due to the difficulties[2] that occurred whilst integrating *Persona* into the project and the limited time available it was advised that the project use *Talis* live *Persona* service. This did restrict the project in minor ways, which will discussed in later in this chapter and Chapter 7.

6.3.3 Build Three

This build begins the implementation of creating groups and sharing clips, covering functional requirements 3.0-3.4 and 4.1. The most important feature for this build will be to create a fully synchronised application so that user's can add new clips, share and comment without having to refresh the page. It will also focus on integrating two more of *Talis'* microservice; *Babel* for annotations and *Depot* to store the images.

[2]It took me four frustrating days to integrate Persona into the web application, in which a Senior Developer told me he was impressed how quickly I managed it.

6.3.3.1 Websockets

Websockets [25] provide a persistent communication channel between client and server without having to use HTTP protocol, it uses it's own upgraded protocol. Before communication can begin first a connection must be established with a "handshake". The server must initially set up a dedicated socket for a client to send requests to connect. The client sends a usual HTTP request to connect, then once the process know as a "handshake" is complete, the connection can be upgraded to websocket. This application uses a Node.js library called *Socket.IO* [2] which provides API for both client and server.

The websockets used in this application are programmed to emit messages each time a model is updated or deleted from the database. The client listens for these message events and synchronises it's data model. This allows real-time communication between users and groups.

6.3.3.2 Babel and Depot Services

Babel is a microservice for storing annotations. It does not enforce a certain schema, however, it does state in the documentation that it strongly recommended that the programmer models the schema based on the *Open Annotation Data Model*[3]. *Babel Node Client* is an API developed for communication with the *Babel Server*. To use any of *Babel Client* API you must supply a token from *Persona* to authorise the request. Each annotation has an attribute called *'hasTarget'*; this is where the ID of the clip and/or group that is being referred to is referenced.

Depot is a microservice for submitting, converting and storing digital resources. This service also requires a *Persona* token to access it's API. *Talis* uses this service for uploading resources into it's player, such as PDFs, videos, word documents, and more. This project will be using *Depot* to store the images clipped from the web. When a successful post is made to *Depot* it returns the image id and url. This url is stored in the web application's own *Mongo* database and is used by the client to display the images on the web page.

6.3.4 Final Build

The final build focused on creating a finished prototype to present as the work achieved during the project. This last build should make sure that all functional requirements with *S* or *M* priority have been met, and standards are also met for the non-functional requirements. Once these requirements have been met then work can begin on the less urgent functional requirements. The only new construction for this build was a new page that displayed a simple log of the user and user's group activity. The log was created as simple as possible due to the limited time available. To add more interaction it required some complex re-factoring of the code, which will be explained in more detail in the evaluation and reflection. Babel also had an API that generated a feed, but this could not be used because the application was hosted locally and did not have permission to use the API. Instead the log utilised the already existing annotation methods by posting new annotations each time an activity occurred, for example the deletion of a clip.

This build also focused on creating a clean UI for the prototype to be presented with. The re-designed interface used *Twitter's Bootstrap* to create a minimalistic and simple view. Some usability features were also added to help users navigate the application, such as tool-tips.

Most of the final build concentrated on developing automated tests for the entire stack. These tests will be discussed in full in Chapter 7.

Some of the time was also spent re-factoring the code so it had a more modular architecture. By encapsulating some of the more lengthy complex methods into services the code became a lot more maintainable and readable.

All the code can be found on the university SVN, please read the README first.

Chapter 7

Evaluation

The evaluation of the project will be carried out by a series of tests, a review of the company feedback, a discussion on the limitations of the final product and finally a look at what further improvements can be made.

7.1 Testing

This section will discuss what testing should be implemented for a Javascript full-stack web application. It will begin by describing simple unit tests and gradually increase the scope to the entire system. A representation of the scope of the tests can be seen in figure 7.1. The larger the scope, the more complex the test and fewer are usually written. Not all the tests that are described in this section were implemented due to their complexities, however, they have been included so that the reader can understand what is usually involved in creating a full test suite for a this type of system.

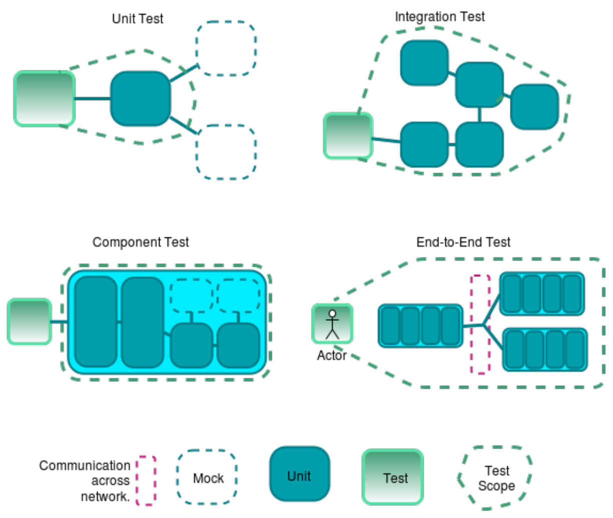


Figure 7.1: Testing Strategies

7.1.1 Automated Tests

7.1.1.1 Unit Tests

A unit test[18] is a low level test that focuses on a small part of the system; in the context of this project a unit will be a single method in a module. Unit tests are often written by the programmer at the time of development. These tests should be ran regularly, usually before each commit, therefore, it is ideal that these tests are able to execute and complete within a couple of minutes.

In some cases a unit collaborates with another service, in these circumstances test doubles can be used. It's important to use test doubles so that the entire environment is controlled while testing. A test double is a mocked version of a service.

Tests were written using Mocha a testing framework built for Javascript that runs on Node.js. Should.js was the library used to test assertions. When the testing stage of the development began more than sixty unit-tests were written to test for the server side methods. Unfortunately, these were more like integration because they were still coupled to Talis' microservices. Nock[34] was used to mock responses from *Persona*, and several new tests were developed[1]. Nock is a tool that allows you to intercept http calls from the server. The tests can be found in the uploaded code at kahoots/kahootsApp/server/api/clip/clip.spec.js . *Clip* can be replaced with group or user. They can be ran automatically in the terminal, instructions can be found in Appendix I.

7.1.1.2 Integration Tests

Fowler in his presentation about *Testing Strategies in a Microservice Architecture* states that "an integration test verifies the communication paths of interactions between components to detect interface defects"[18]. The unit tests defined in the previous section had test doubles isolating them from external services. In the integration tests for this project these test doubles are replaced with the actual services. This level of testing should test for timeouts and catch errors such as missing http headers. The tests should attempt to verify that the integration modules can handle failures gracefully.

As previously mentioned, instead of unit tests, integration tests were written for the server. The developer has less control over these tests as they reply on external services. These tests can also been seen in the .spec files found in the server-side code.

[1]Not enough time was available to rewrite all sixty tests and it was quite complicated to mock Depot and Babel too. Furthermore, it also was not possible to use the real services because they required a real Persona token to access their API. The original tests can be seen in the test files, however, at present they are commented out.

7.1.1.3 Component and Contract Tests

Contract tests are a type of integration test which focus on the boundary of the external service. These tests verify that the service meets the contract expected by the software being developed. This project uses a variety of third party API, contract tests should be written for each one. As this was an internal project for *Talis* using their own API, these tests were not of high priority.

The scope of a component test covers a well-encapsulated part of a system being developed. These tests can be implemented using a tool called Postman which allows the developer to write a set of tests that send http requests. Other test were the priority, so only a few examples of these tests were written.

7.1.1.4 End-to-End Testing

End-to-end tests cover the entire scope of the system from the front-end to the back-end, it verifies the system meets requirements as stated in the design. The tests should cover the entire system including internal databases. In this project these tests will also encapsulate external microservices. These tests may suffer from reliability problems out of the control of the programmer. These tests are the most difficult to write, and suffer greatly from flakiness; therefore, as few as possible should be implemented. Usually these are written with a tool called Protractor, that runs the application through a browser just like a normal user would. Due to the complexity of these tests and the limited time available only a manual version of these tests were implemented.

7.1.2 Manual Tests

These tests were written to replace the automated end-to-end tests. They prescribe a sequence of actions that the evaluator could follow to check for potential usability problems and that the system requirements have been met. A set of scenarios based on possible actions available on the user interface were described; and for each scenario the evaluator decided on a score. The scores were based on the five-point rating scale for the severity of usability problems[28] which are outlined below:

1. Evaluator believes these is not a problem.

2. Cosmetic only, not significant.

3. Minor problem, does not affect functionality too much. Low priority.

4. Major problem, must be fixed. High priority.

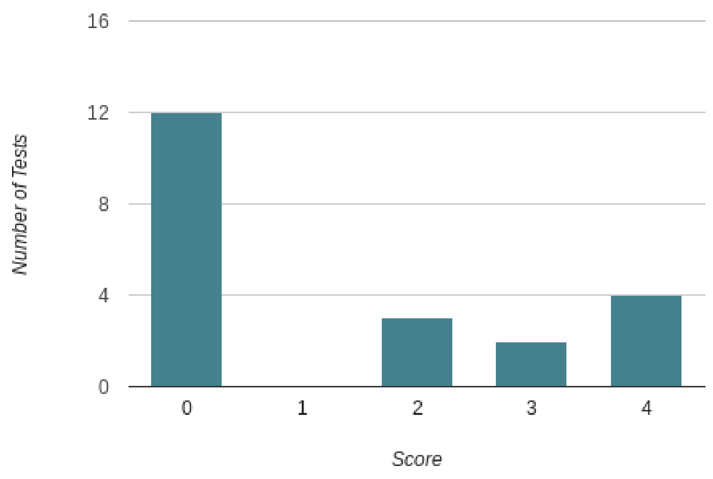


Figure 7.2: Usability Scores for Manual Browser Tests

5. Usability Catastrophe!

The full set of results for these tests can found in Appendix E. For each failed test an action was described to fix the fault and a score of priority. A summary of results is displayed in figure 7.2. The figure shows that most tests completed without failure. The tests that scored three or four were set with high priority (and fixed), and the tests that scored two were given lower priority.

7.1.3 Summary

The tests that were implemented were extremely useful at finding bugs that existed in the code, especially for boundary cases. It is very important to keep automated tests as up to date as possible. This means that new code can be written quickly with the assumption the old code is reliable. A lot of time was dedicated to learning the testing tools used in this project, however, to be able to complete all the tests would require more time than available. The developer has tried to implement a variety of example tests that could be implemented in furture development. It was clear as the project neared an end that the code written would need a lot of re-factoring before more additional feature could be added; so it was not a priority to write all the tests that would then need to be rewritten.

7.2 Company Feedback

A copy of the company feedback can be found in Appendix G. Overall the company was very satisfied with the progress made during the ten week internship. The company has not yet explored the idea of a plugin and this gave them an idea of what was feasible. The company was very impressed with the way the work was managed and the skills developed in the short time available. More specific questions were asked about the product and further development which can be found in the uploaded code. The answers received were evaluated and will be discussed in further detail in the next two sections.

7.3 Limitations

This section will discuss what the author found to limit the development of the project and what had not been implemented as well as it could have been.

There is one main aspect of the architecture that contributes to most of the limitations felt by the project. This aspect was that the model and business logic were not as separated as well as the MVVM model defines. This caused some of the data on the client-side to be duplicated or to be unnecessarily passed as a parameter through a chain of methods. This restricted the building of a more interactive interface, such as links to specific clips or groups, or saving the state of the page so a user can return to the same place. This limitation was not understood until later in the project when more complex features were being implemented. Due to time restrictions it was decided that the code would not be re-factored, but would be the main focus for future improvement. The fact that this was new technology can account for the late discovery of this limitation, but in reflection maybe a more effective plan of the architecture would have prevented this from having such a large effect.

To expand on the issue that the project was built using all new technology, it was felt that maybe the project scope was too large for the time available. The full stack web application and the chrome extension together create quite a large system, and larger still if you include the three *Talis* microservices. The project scope developed is quite wide and not very deep; the majority of the time was spent on the web application. It would have been possible to explore more depth and complexity if the scope only included the chrome extension; new features could have been explored that do not already exist on the market.

Another area of the project that could have been significantly improved was testing. Automated testing was left until later because of the unfamiliarity with the Javascript testing suites. There was also a large quantity of new technology being introduced at the start that made it difficult to add more. Once testing became a priority it was found to be quite complex and not enough time was reserved to complete them. In future practice tests should be prepared before the build begins.

In summary the project did suffer from being slightly too ambitious. The main limitations were caused by both having a lot of new technology and there being too much work planned for the restricted time frame. However, in reflection, given the circumstances a good base for a prototype was constructed and the many new skills were learn - the 400 hours dedicated to this project were used productively.

7.4 Further Improvements

From the knowledge and understanding gained throughout the project there are a number of improvements that would be made to the existing code before new features could be added. The first of these is re-factoring the code so that the model, controller and services are well defined on the client-side. Another would be to implement a good set of automated tests that could be run before each new build. And finally to check the security of the application, starting with a profile that is not a super-user[2].

After reflecting on the work done and the feedback from the company the following features would be candidates for the next build:

- Next generation annotations - allow users to comment on a specific point on the clip. Then analytics could be recorded on how a clip is being used.
- Adding analytics to the chrome extension, monitoring which websites are being clipped.
- Improve the feedback given to a user when a clip is captured, and allow tags to be added before being sent to the server.
- Allow the clipper to clip more than screen shots, such as grab videos or text.

[2]As this project only uses Talis' live Persona Server I was only given two test accounts, that happened to have 'su' scope. This meant they had admin authority and I couldn't test whether a normal user could access another users clipped content.

Chapter 8

Conclusion

The aims for the work described in this Report were to create a browser plugin to capture content from the web, develop an application to view and discuss the captured content, and to make use of *Talis'* microservices.

The first aim was to create a proof of concept of a tool that could be used to capture content from any web page. This was achieved by creating a Chrome extension that allowed users to select part of the page, that was then captured and sent to the web application. The extension made use of *Talis'* authentication service Persona to allow the student to log-in and the authenticate requests to both the web application and another *Talis* microservice Depot. Depot was used to store and retrieve the clips created by the student.

The second aim was to develop an application that allowed students to login, view, share and discuss the content they had captured. It was designed to be dynamic and easy to use. Persona was also used by the application to allow the students to login, maintain the privacy of the user's content and to secure the server's API. In addition to Persona it also makes use of both Depot and Babel microservices. Babel is used to store comments and create activity feeds. The application provides students the ability to create private groups which allows clips to be shared and discussed.

Overall, the system developed throughout the project and internship was a solid framework of a chrome extension and web application that could easily be extended and improved. The system supports all the high priority functional and non-functional requirements (see Section 5). Furthermore, the company agreed that the project gave them a clear picture of what features were feasible to develop in such a tool as the web clipper and responded with very encouraging feedback.

Finally, the skills developed over the project were demonstrated by the fact that the software engineering approach which was executed successfully and showed good management skills. The only area that was not planned as as efficiently as the proposed was the testing, which will need to be considered in greater detail in future work.

Appendix A

Summer Internship - Project Brief

At Talis we are helping to connect teaching and learning in better and more meaningful ways. One area we are currently exploring is around developing simple tools that allow students to collect together resources from around the web, and share and discuss that content with their peers.

At Talis we have already developed a number of Micro Services that we use to provide core capabilities across our product suite:

- Persona: A service for authentication, user identity and profiles.
- Depot: A Service for Submitting, converting and storing digital resources
- Babel: A service for storing and suggesting annotations about things. It can be used as a foundation to build a threaded discussion forum or feed, an entity autosuggestion service, a video tagging feature or anything else you can think about to do with annotations.
- Echo: A service to record events and provide analytics capabilities

The purpose of this project is to build a set of simple tools that integrate with, or make use of, the above services and allow Students to capture content on any webpage and then organise that content for their personal study goals, or share that content with their study groups. You should explore:

- Developing a browser plugin (chrome or firefox extension) that allows students to capture content on a page.
- How to snapshot and send that content to Depot for storage and retrieval, beginning with a simple use case capture content on the page as an image
- Develop an application students can log into and organise, share and discuss the content they have captured
- Capturing analytics on the frequency and the amount of time students are interacting with content.

Appendix B

Project Timeline

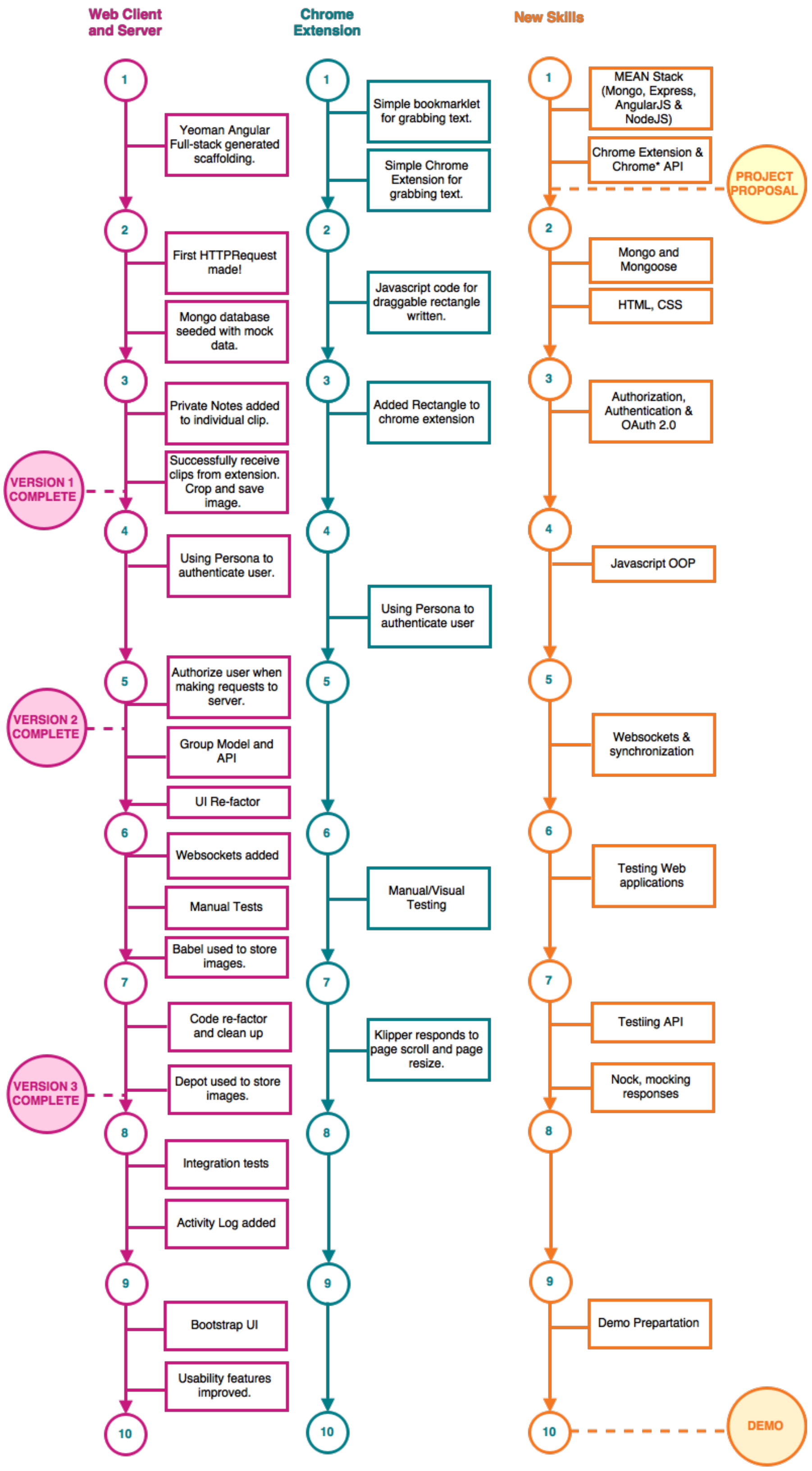

Web Client and Server
Chrome Extension
New Skills
1
Yeoman Angular Full-stack generated scaffolding.
2
First HTTPRequest made!
Mongo database seeded with mock data.
3
Private Notes added to individual clip.
Successfully receive clips from extension. Crop and save image.
VERSION 1 COMPLETE
4
Using Persona to authenticate user.
5
Authorize user when making requests to server.
VERSION 2 COMPLETE
Group Model and API
UI Re-factor
6
Websockets added
Manual Tests
Babel used to store images.
7
Code re-factor and clean up
Depot used to store images.
VERSION 3 COMPLETE
8
Integration tests
Activity Log added
9
Bootstrap UI
Usability features improved.
10
1
Simple bookmarklet for grabbing text.
Simple Chrome Extension for grabbing text.
2
Javascript code for draggable rectangle written.
3
Added Rectangle to chrome extension
4
Using Persona to authenticate user
5
6
Manual/Visual Testing
7
Klipper responds to page scroll and page resize.
8
9
10
1
MEAN Stack (Mongo, Express, AngularJS & NodeJS)
Chrome Extension & Chrome* API
PROJECT PROPOSAL
2
Mongo and Mongoose
HTML, CSS
3
Authorization, Authentication & OAuth 2.0
4
Javascript OOP
5
Websockets & synchronization
6
Testing Web applications
7
Testiing API
Nock, mocking responses
8
9
Demo Prepartation
10
DEMO

Figure B.1: Project Timeline

Appendix C

Detailed Component Diagram

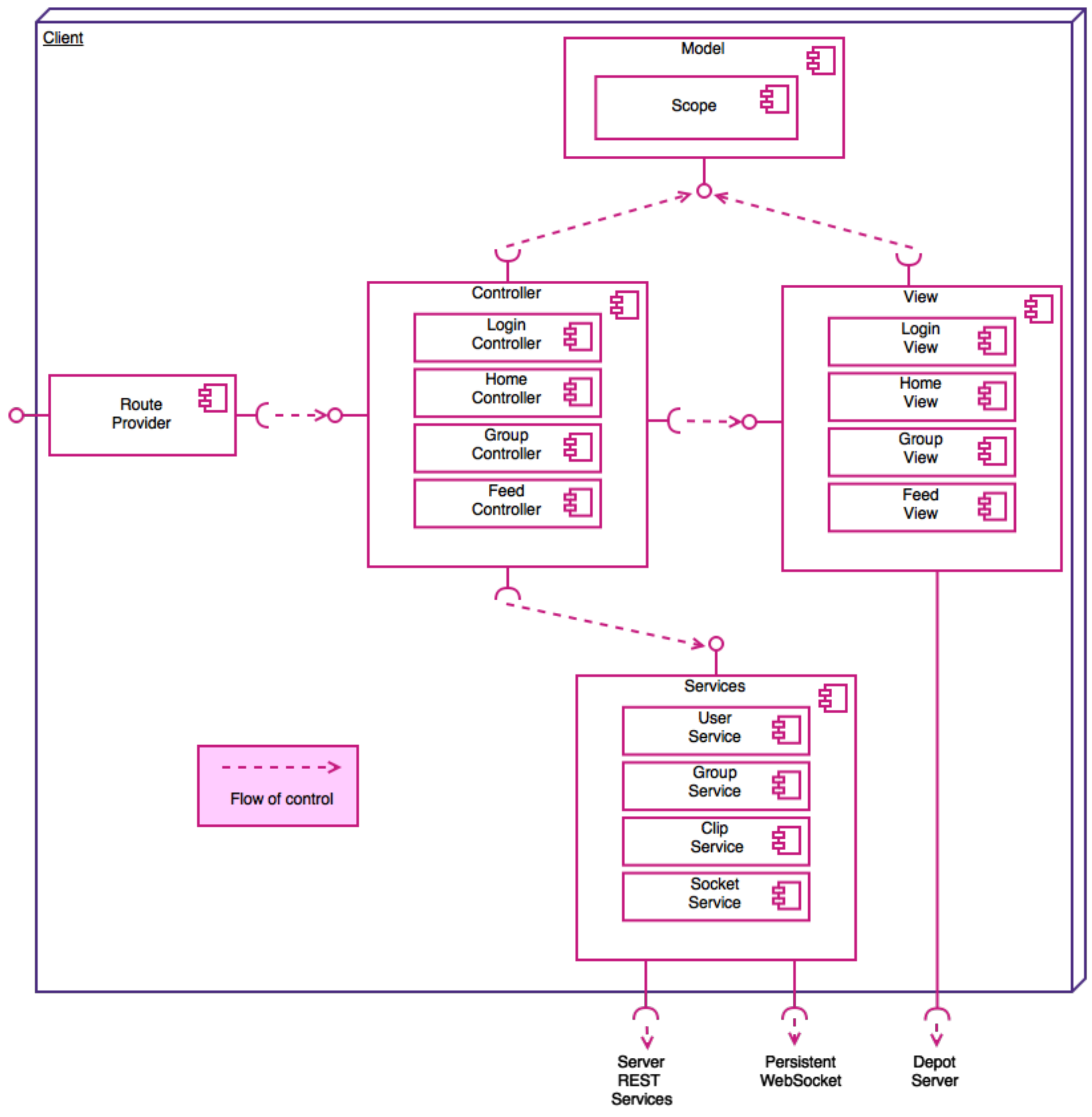


Figure C.1: Client Component Diagram

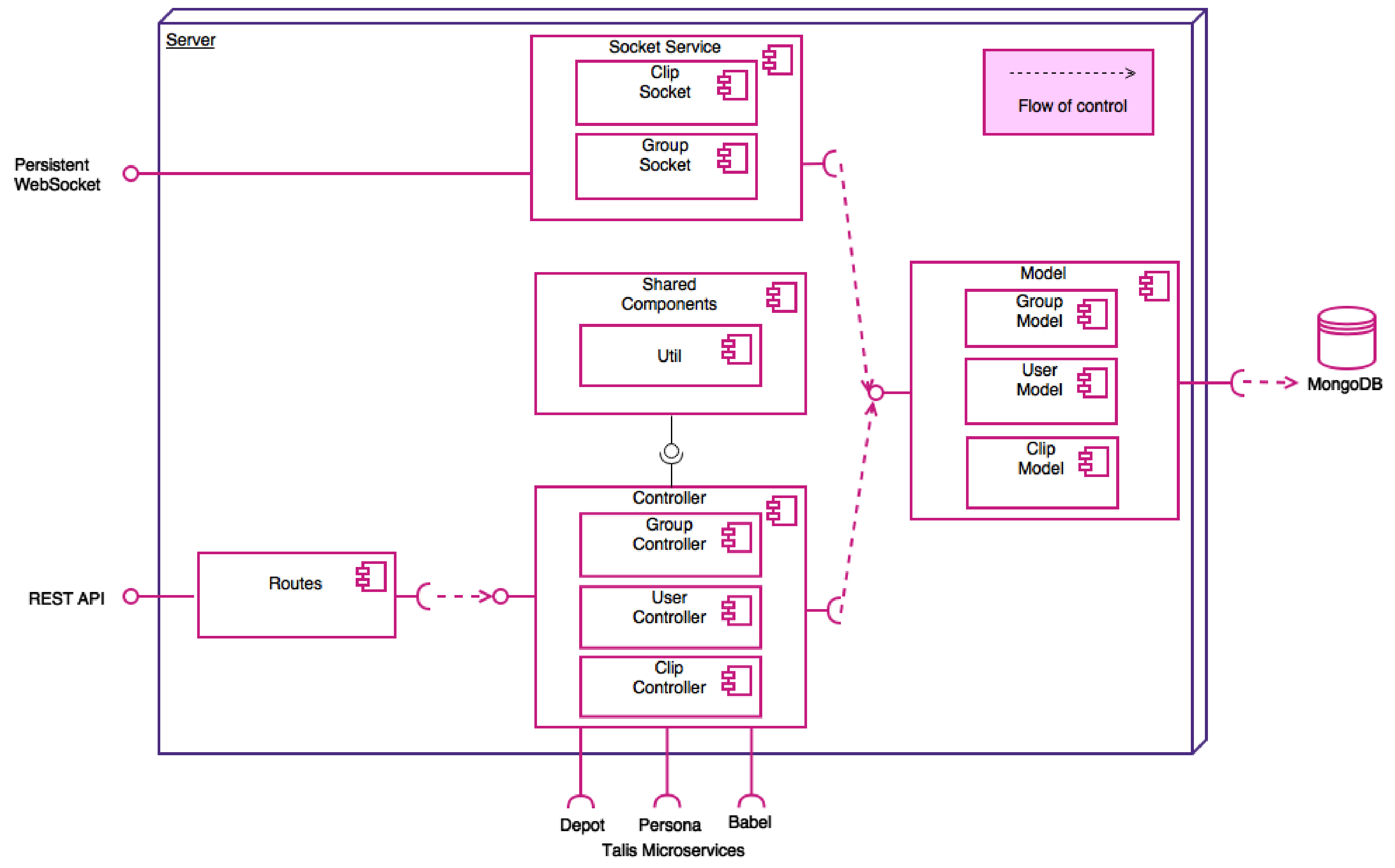


Figure C.2: Server Component Diagram

Appendix D

Sequence Diagrams

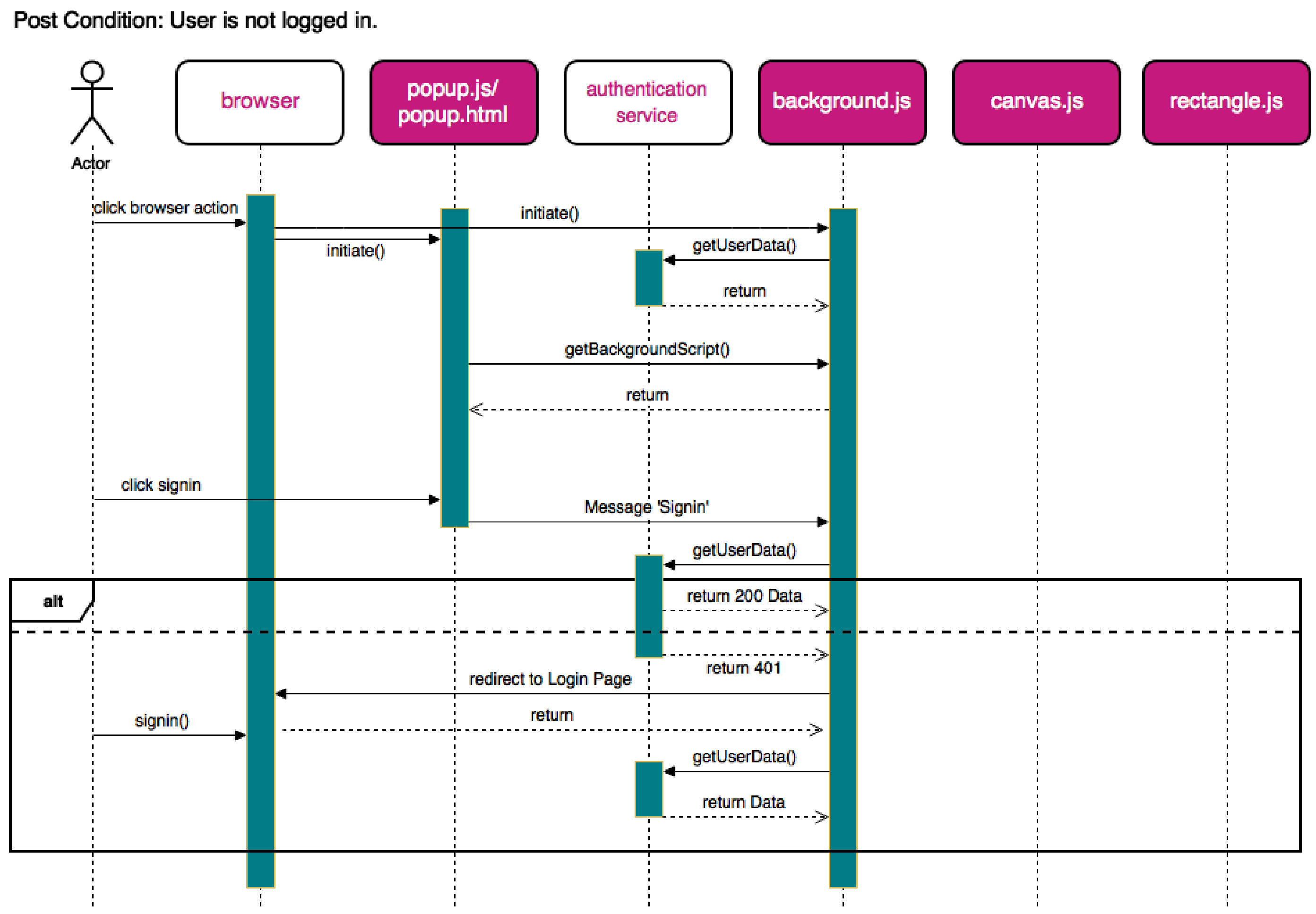


Figure D.1: Sequence Diagram for Chrome Extension Login

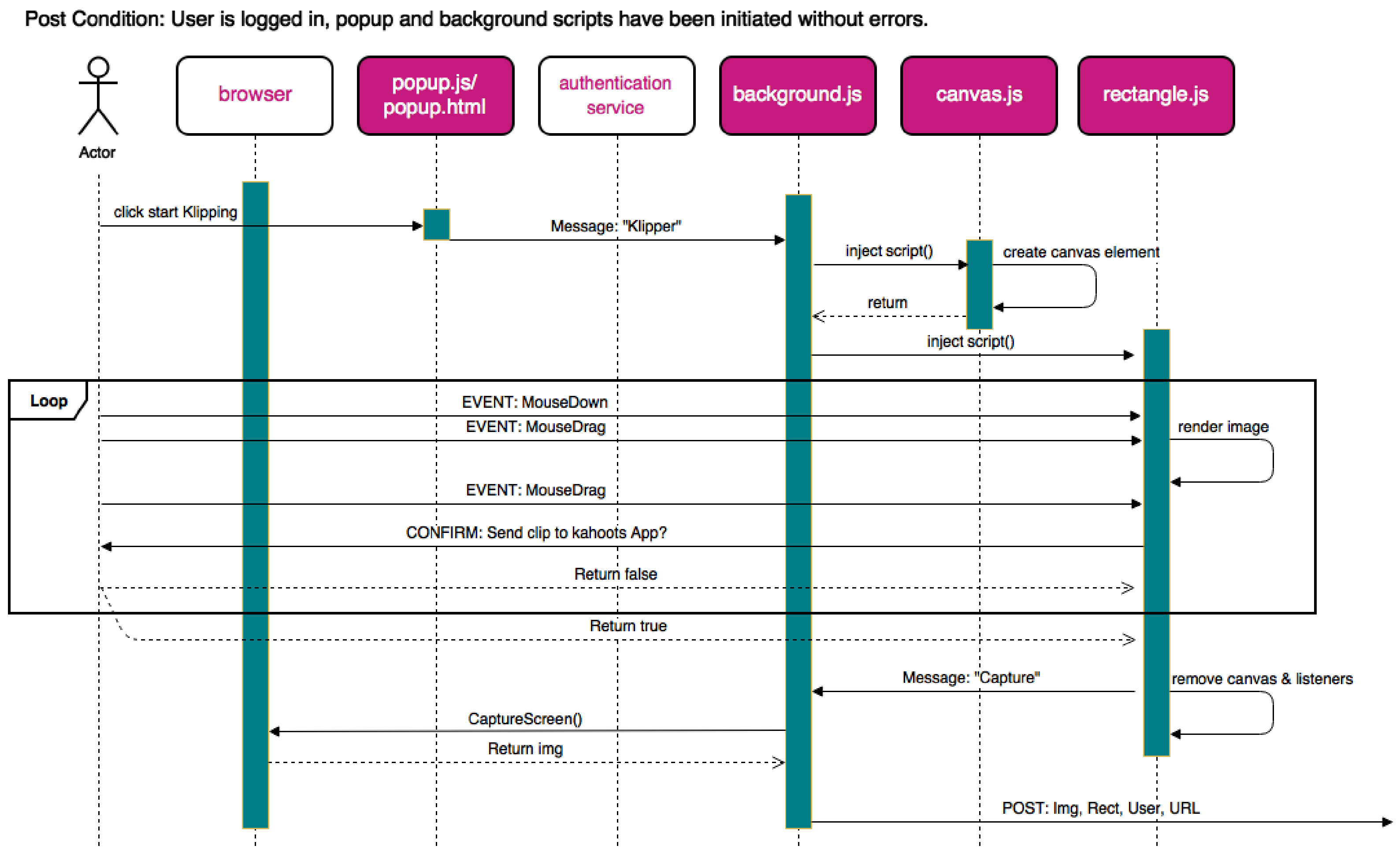


Figure D.2: Sequence Diagram for Chrome Extension Clip Function

Appendix E

Manual Browser Test Results

These two tables show an example of browser test results conducted by the evaluator during development. The following scores reflect the significance of the problem:

0 - Evaluator believes there is not a problem.

1 - Not significant, cosmetic only.

2 - Minor problem, doesnt affect usability too much. Low priority.

3 - Major problem, important to fixed. High priority.

4 - Usability catastrophe![15]

Precondition for tests: Babel, Depot and Persona Services are running correctly.

Table E.1: Home-page Cognitive Walkthrough Test Results

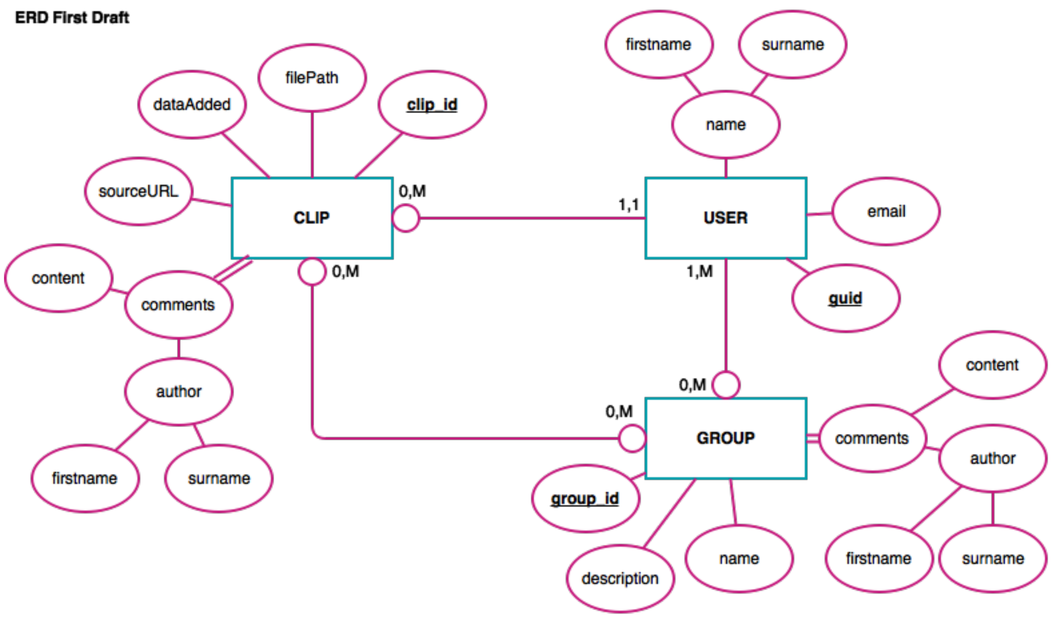
No.	Scenario	Result	Action
1.	Log-in to app, then use extension to capture content.	Clip appeared on homepage within seconds. The clip becomes first in the list, and becomes the active clip.	None.(0)
2.	Do not Log-in to app, then use extension to capture content.	Once logged in the Clip appeared on homepage within seconds. The clip becomes first in the list, and becomes the active clip.	None.(0)
3.	Add a private comment to a clip.	Comment appeared in text area above within seconds.	None.(0)
4.	Add a comment that will cause the text area to overflow.	Text area does not expand and a scroll bar appears, but does not scroll to most recent message.	4.1. Make scroll stick to most recent comment. (2)
5.	Select a new active clip.	Active clip is updated, comments are updated, link to resource correct.	None.(0)
6.	Delete selected clip.	If clip has been shared with the current active group it is not removed from group it appears to still be there. Clip successfully deleted from user's homepage.	6.1. Swap hard delete to soft delete so records are retrievable from Depot.(3) 6.2. Group page needs to synchronize with deletion.(4)
7.	Share clip with existing group.	Clip has not been shared.	7.1 Fix share button.(4)
8.	Remove all clips and refresh page.	Displays 'no clips available' image.	None.(0)

Table E.2: Group-page Cognitive Walkthrough Test Results

No.	Scenario	Result	Action
9.	Initialise page with no groups.	Displays 'You have no groups' alert. Displays 'no clips available' image. Hides active clip view.	9.1 Do not show 'no clips available' image.(2) 9.2 Disable 'add user' button.(4)
10.	Initialise page with one group with no clips.	Displays 'no clips available' image. Hides active clip view.	None.(0)
11.	Initialise page with many groups with many clips.	Displays active clip and all clips in list below. Comments loaded correctly.	11.1 Clip's group tabs will be removed in this build.(4)
12.	Share clip in one group with another group.	Clip shared successfully. Displays 'clip has been shared' popup.	None.(0)
13.	Remove last clip from group.	Removes clip and page state is resets to 'no clips available'.	None.(0)
14.	Remove first of many clips from group.	Removes clip from group. Sets the next clip to active clip.	None.(0)
15.	Remove a clip from the middle of list.	Removes clip from group. Sets the first clip in list to active clip.	None.(0)
16.	Click on a group tab to make it active.	List of clips is updated. First clip in list becomes active. Highlights active group tab.	16.1 If you refresh the page the active group will return to the first group. It would be ideal if the state was saved.(2)
17.	Add shared comment to a clip.	Comment is posted in correct group and clip.	None.(0)
18.	Initialise with one group, then leave group.	Resets state of page and displays 'You have no groups' alert.	None.(0)
19.	Initialise with many groups, then leave first group in list.	Works correctly sporadically. Sometimes new active group is not set.	19.1 Investigate and fix bug.(3)

Appendix F

Original Entity Relational Diagram



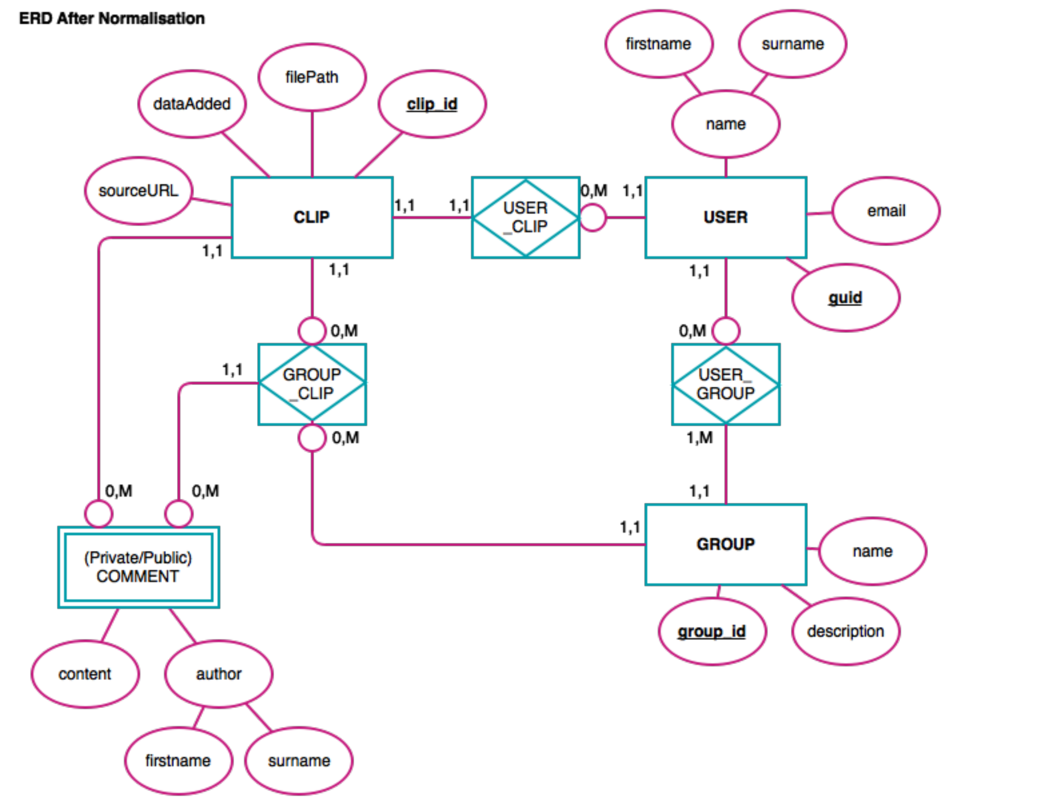


Figure F.1: a) Un-normalised ERD b) Normalised ERD

Appendix G

Summer Internship 2015 - Lauren Lewis - Project Feedback

G.1 Feedback from Nadeem Shabir (Talis Intern Manager)

The project that Lauren was assigned was to build a set of simple tools that integrate with, or make use of, a number of Talis micro-services, in order to allow Students to capture content on any webpage and then organise that content for their personal study goals, or share that content with their study groups. The project required Lauren to specifically explore the following:

- developing a browser plugin (chrome or firefox extension) that allows students to capture content on a page.
- how to snapshot and send that content to Depot for storage and retrieval, beginning with a simple use case capture content on the page as an image
- develop an application students can log into and organise, share and discuss the content they have captured
- time permitting: capture analytics on the frequency and the amount of time students are interacting with content.

Overall, we feel that Lauren achieved what we asked to her, and in the process impressed all of us.

The technology stack that she used was, in large part, prescribed by Talis, and consequently was completely new to Lauren. We knew that this would be a challenge but are pleased to say that she was able to rapidly get to grips with these technologies. This is particularly impressive given her level of knowledge and experience prior to starting her internship. We are all very impressed with how she managed to pick up so many new things in a short amount of time, and produce a working prototype of both the Chrome extension and the Student facing webapp.

She was also able to to integrate the prototype with an number of our own Micro Services, each of which is essentially a REST based API. Some of the concepts around REST were also very new to her, as were the nuances of some of our services and the standards that they implement, for example OAuth. Lauren demonstrated a great deal of initiative and took a keen interest in really understanding the pros and cons / strengths and limitations of these technologies, and how they could be applied to this project.

Lauren fitted in well with the rest of the team, and showed had a definite passion for Software Development. We hope her experience has given her encouragement for the future when she finishes her degree.

This has been a very successful project, and we would like to thank Lauren for all her hard work.

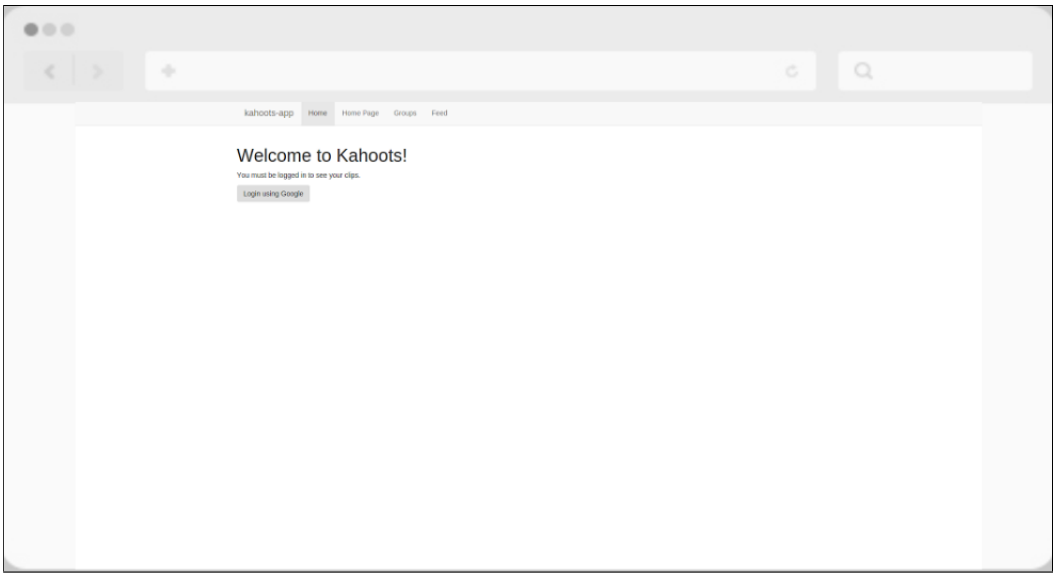
In terms of your questions: **Anything that could be improved?** It's a bit of a loaded question, when I consider what we asked you to explore and the constraints we placed around that i.e. time, tech stack (which was very new to you), I think what you created was really good. If I look at the app / extension and ignore that context and think of it just in terms the functionality it offers then there are some things that might improve it:

- more focus on UX, really bottom out how to build an experience that really encourages students to collect and share these resources
- create more feedback loops within the application based on analytics, tell a student how many times his/her peers have looked at content she Klipped and shared with them.
- Notifications, so I know when someone has commented on one of my Klips and I can engage in a discussion with them.
- With the Chrome extension, perhaps explore more complex Klipping, grabbing video, or actual text

Is the prototype useful to Talis? I think so, it explored some areas we hadn't particularly around chrome extensions, so that is useful. And I think we might very well consider implementing our own browser plugins in the future. Its an area we haven't invested a lot of time in but you've demonstrated some of whats possible and it gives us more food for thought.

Appendix H

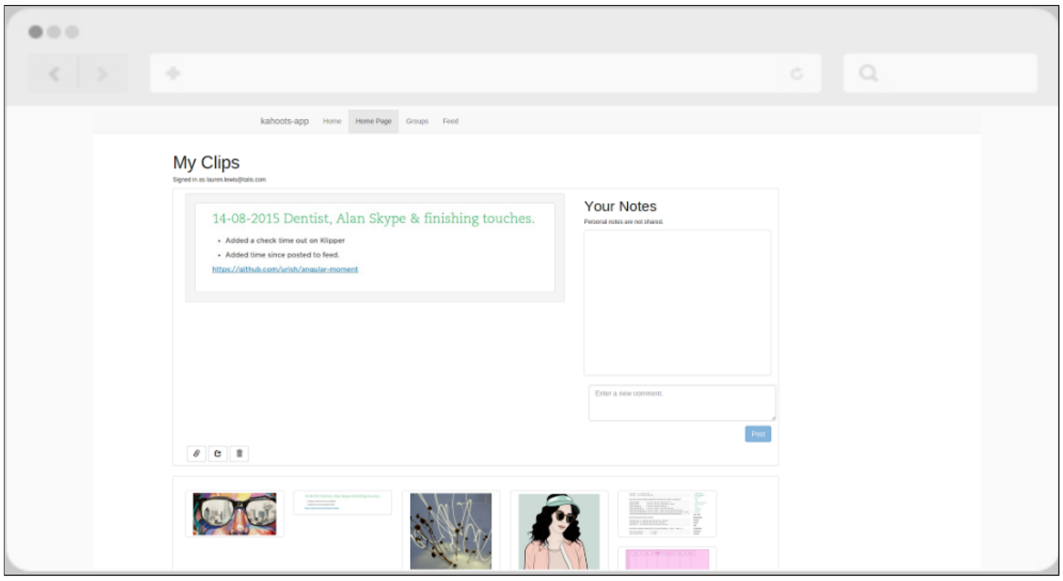
Software Screenshots



Welcome to Kahoots!

You must be logged in to see your clips.

Figure H.1: *Kahoots web application landing page.* When the sign-in with Google button is clicked the user is redirected to Google to sign in, which then redirects back to the Kahoots homepage with the access token appended to the URL.



My Clips

Signed in as lauren.lewis@talis.com

14-08-2015 Dentist, Alan Skype & finishing touches.

- Added a check time out on Klipper
- Added time since posted to feed.

https://github.com/urish/angular-moment

Your Notes

Personal notes are not shared.

Enter a new comment.

Figure H.2: *Kahoots web application home page.* This is where a user can view all the content they have captured and make private comments.

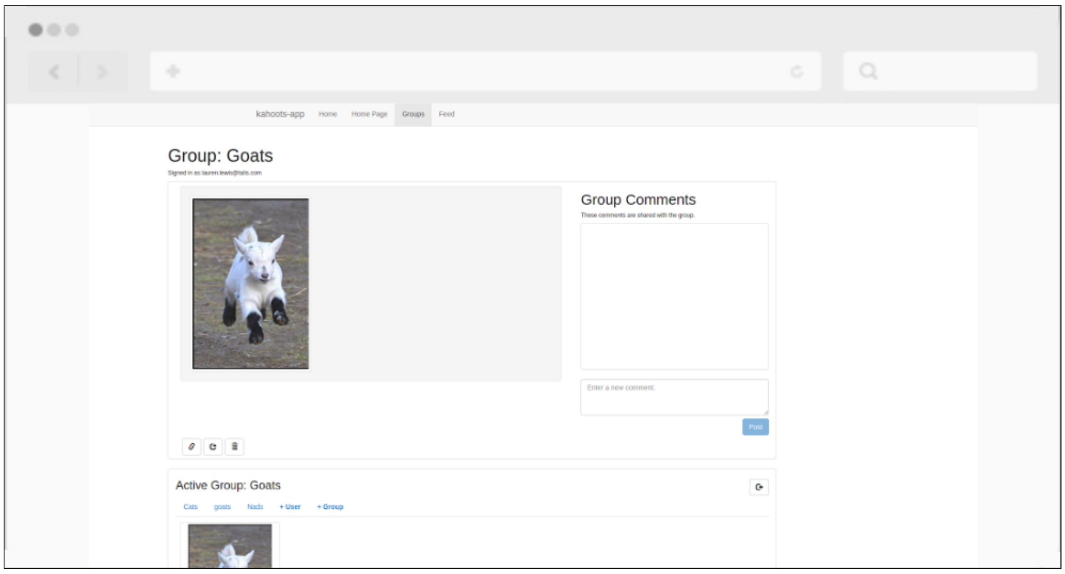
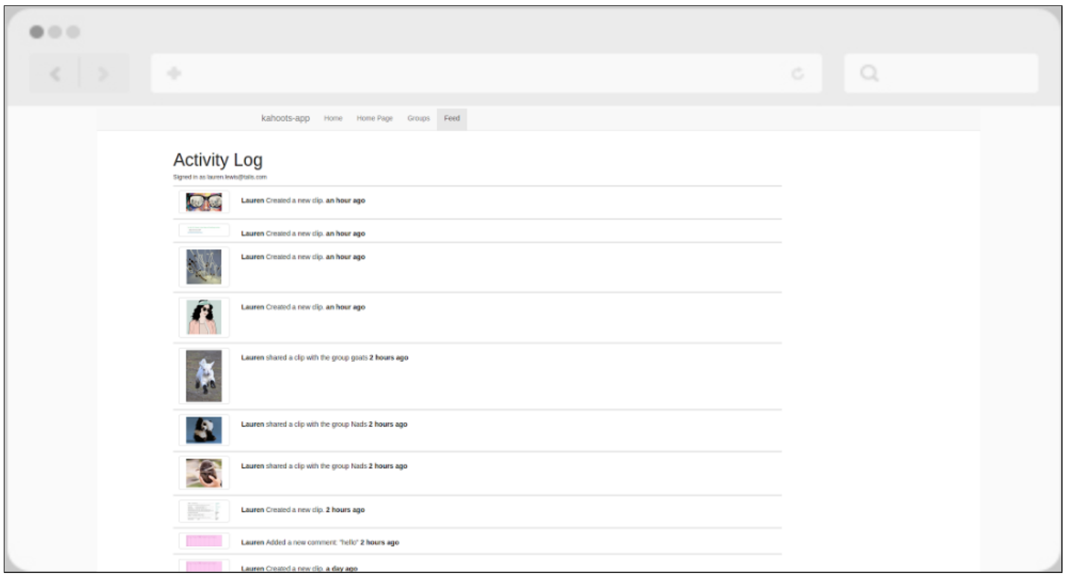


Figure H.3: *Kahoots web application group page.* This is where a user can access their groups, view group clips, add a new group, or leave a group.



Lauren Created a new clip. **an hour ago**
Lauren Created a new clip. **an hour ago**
Lauren Created a new clip. **an hour ago**
Lauren Created a new clip. **an hour ago**
Lauren shared a clip with the group goats **2 hours ago**
Lauren shared a clip with the group Nads **2 hours ago**
Lauren shared a clip with the group Nads **2 hours ago**
Lauren Created a new clip. **2 hours ago**
Lauren Added a new comment: "hello" **2 hours ago**
Lauren Created a new clip. **a day ago**

Figure H.4: *Kahoots web application activity log page.* This is where a user can view all their and their groups activity.

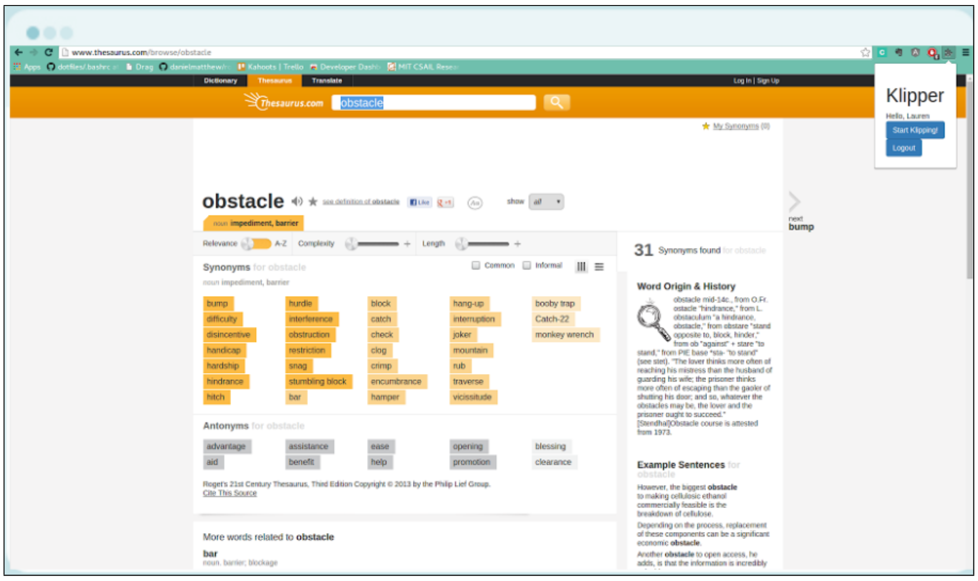


Figure H.5: *Kahoots Klipper Chrome Extension browser-action popup.* In the top right corner of the browser you can see the popup that appears when the user click the browser action button.

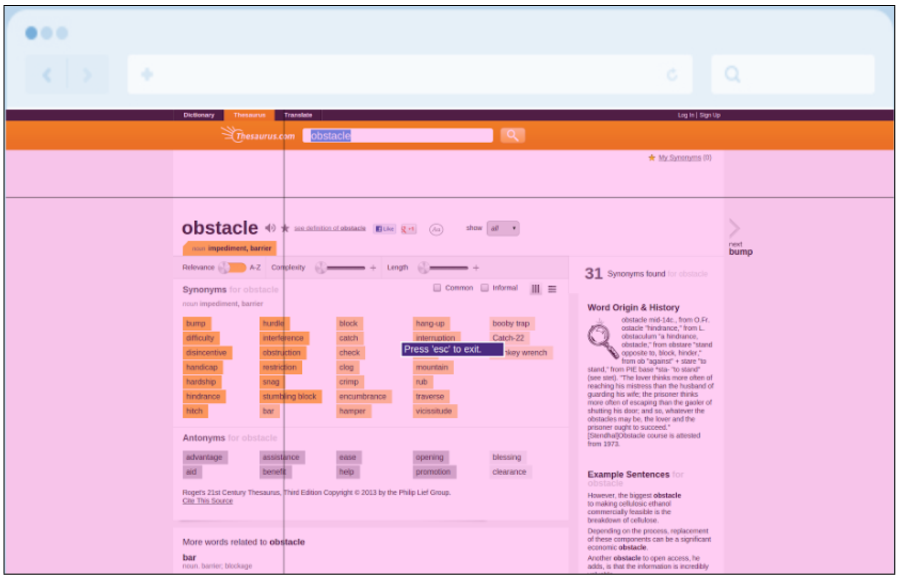
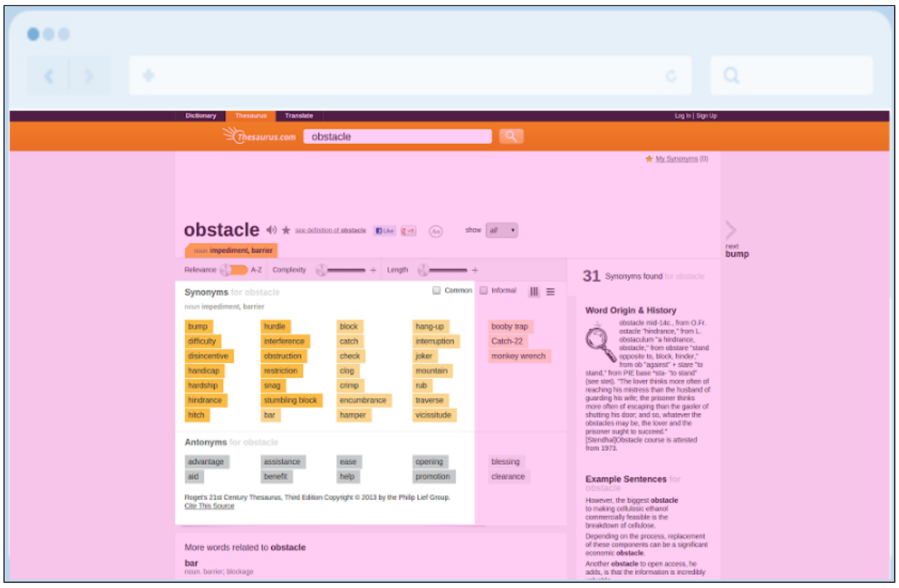


Figure H.6: *Kahoots Klipper Chrome Extension start clipping.* Once the user has signed in and then clicked the start clipping button the page the clipping tool script is injected into the current tab's web page. The cross that is shown in this diagram follows the cursor to allow users to select an area to capture.


Dictionary Thesaurus Translate

Log In | Sign Up

Thesaurus.com obstacle

My Synonyms (0)

obstacle see definition of obstacle show all

noun impediment, barrier

next bump

Relevance A-Z Complexity Length

31 Synonyms found for obstacle

Synonyms for obstacle

Common Informal

noun impediment, barrier

bump	hurdle	block	hang-up	booby trap
difficulty	interference	catch	interruption	Catch-22
disincentive	obstruction	check	joker	monkey wrench
handicap	restriction	clog	mountain	
hardship	snag	crimp	rub	
hindrance	stumbling block	encumbrance	traverse	
hitch	bar	hamper	vicissitude	

Antonyms for obstacle

advantage	assistance	ease	opening	blessing
aid	benefit	help	promotion	clearance

Roget's 21st Century Thesaurus, Third Edition Copyright © 2013 by the Philip Lief Group.
Cite This Source

More words related to **obstacle**

bar
noun. barrier; blockage

Word Origin & History

obstacle mid-14c., from O.Fr. ostacle "hindrance," from L. obstaculum "a hindrance, obstacle," from obstare "stand opposite to, block, hinder," from ob "against" + stare "to stand," from PIE base *sta- "to stand" (see stet). "The lover thinks more often of reaching his mistress than the husband of guarding his wife; the prisoner thinks more often of escaping than the gaoler of shutting his door; and so, whatever the obstacles may be, the lover and the prisoner ought to succeed." [Stendhal]Obstacle course is attested from 1973.

Example Sentences for obstacle

However, the biggest **obstacle** to making cellulosic ethanol commercially feasible is the breakdown of cellulose.

Depending on the process, replacement of these components can be a significant economic **obstacle**.

Another **obstacle** to open access, he adds, is that the information is incredibly

Figure H.7: *Kahoots Klipper Chrome Extension finish clipping.* Once the user has dragged out a rectangle shape over the content the extension application will confirm with the user before sending the web application.

Appendix I

Kahoots README

I.1 Kahoots Web App and Kahoots Web Klipper

Kahoots Web App and the chrome extension Kahoots Klipper were developed as part of a 10 week internship at Talis. The purpose of this project was to build a simple tool that allowed students to capture content from the web and then organise and share then content for their personal study goals, or share that content with their study groups.

Install Prerequisites Before installing Kahoots, you will need to set up your dev environment. Instructions can be found here (Step 1 only):

```
http://yeoman.io/codelab/setup.html
```

Install Kahoots Web App To install, grab a copy of the repository:

```
git clone https://github.com/talis/kahoots.git
```

Install all of the dependencies:

```
npm install
bower install
```

The app requires mongo. For mongo, you may need to create db directory:

```
sudo mkdir /data/db
```

Make sure mongo is running:

```
mongod
mongo
```

for more information on installing mongo, visit:

`http://docs.mongodb.org/manual/installation/`

For this project you will also need have your own babel server, information on how to install this can be found here: (This is a private repository, only available to developers in the Talis organisation) If you do not add this step none of the comments will work, or the feed.

`https://github.com/talis/babel-server`

To being the application, use the grunt command:

```
cd kahoots/kahootsApp
grunt serve
```

I.1.1 Adding Kahoots Klipper to Chrome Browser

1. On your chrome browser, click menu.
2. Select More tools ¿ Extensions.
3. Click on the button Load unpacked extension...
4. Select kahoots/kahootsKlipper/app and click open.
5. You should see a little jigsaw puzzle icon in the top right corner of your browser, click this to start the Klipper.

I.1.2 Running tests for the web application

To begin the tests, use the grunt command:

```
cd kahoots/kahootsApp
grunt test:server
```

Bibliography

[1] *Snip suggest.* http://www.snipit.org/about/.

[2] *Socket.IO.* http://socket.io/.

[3] *Open Annotation Data Model*, 2013. http://www.openannotation.org/spec/core/20130208/index.html.

[4] *Single-page Application*, 2015. https://en.wikipedia.org/wiki/Single-page_application.

[5] *Social Bookmarking*, 2015. https://en.wikipedia.org/wiki/Social_bookmarking.

[6] Maarten van Steen Andrew S. Tanenbaum. *Distributed systems : principles and paradigms.* Prentice Hall, 2002.

[7] S. Bradner. Key words for use in rfcs to indicate requirement levels. 1997.

[8] ISTQB Exam Certification. *What is Incremental Model- Advantages, Disadvantages and When to use it*, 2015. http://istqbexamcertification.com/what-is-incremental-model-advantages-disadvantages-and-when-to-use-it/.

[9] Stephan Chapman. *Ajax: Why Aysnchronous is Almost Always Better*, 2015. http://javascript.about.com/od/ajax/a/ajaxasyn.htm.

[10] Chrome. *Content Scripts.* https://developer.chrome.com/extensions/content_scripts.

[11] Chrome. *Cross-Origin XMLHttprequests.* https://developer.chrome.com/extensions/xhr.

[12] Clipular. *Clipular! Screen capture reinvented. Screenshots Collected.* https://www.clipular.com.

[13] Victor R. Basili Craig Larman. Iterative and incremental development: A brief history. *IEEE Computer Society*, 36(6):47–56, 2003.

[14] Ruth Cohen David Boud and Jane Sampson. *Peer Learning in Higher Education.* Kogan Page, 2001.

[15] Finlay J. Abowd G. D. Beale R. Dix, A. *Human-Computer Interaction (3rd ed.).* Harlow, England: Pearson Education Limited., 2004.

[16] Evernote. *Evernote - Web Clipper.* `https://evernote.com/webclipper/`.

[17] Evernote. *Evernote Web. The Workspace for Your Life's Work.* `https://evernote.com/`.

[18] Martin Fowler. *Testing Strategies in a Microservice Architecture*, 2014. `http://martinfowler.com/articles/microservice-testing/`.

[19] Matt from Google. *Overview of Google Docs, Sheets, and Slides*, 2015. `https://support.google.com/docs/answer/49008`.

[20] Github. *Github.* `https://github.com/`.

[21] Google. *The activeTab permission*, 2015. `https://developer.chrome.com/extensions/activeTab`.

[22] Google. *Chrome.tabs*, 2015. https://developer.chrome.com/extensions/tabs.

[23] William Jones. *Keeping Found Things Found.* Morgan Kaufmann, 2008.

[24] Jeremy Likness. *Model-View-ViewModel MVVM Explained*, 2010. `http://www.codeproject.com/Articles/100175/Model-View-ViewModel-MVVM-Explained`.

[25] Eiji Kitamura Malte Ubl. *Introducing Websockets*, 2010. `http://www.html5rocks.com/en/tutorials/websockets/basics/`.

[26] MongoDB. Mongodb crud operations. 2015. MongoDB Documentation Project, Release 3.0.6.

[27] mongodb. *Top 5 Considerations When Evaluating NoSQL Databases*, 2015. `https://s3.amazonaws.com/info-mongodb-com/10gen_Top_5_NoSQL_Considerations.pdf`.

[28] Jakob Neilson. Usability inspection methods. pages 47–49.

[29] Sam Newman. *Building Microservices.* O'Reilly, 2015.

[30] Peter Bacon Darwin Pawel Kozlowski. *Mastering Web Application Development with AngularJS.* Packt, 2013.

[31] Carla S. Redden. Social bookmarking in academic libraries: Trends and applications. *The Journal of Academic Librarianship*, 36(3):219–227, 2010.

[32] Rebecca Riordan. *Head First Ajax.* O'Reilly, 2008.

[33] Talis. *Lighthouse-A Talis Project*, 2015. `http://talis.com/lighthouse/`.

[34] Pedro Teixeira. *Nock. HTTP Server mocking for Node.js*, 2015. `https://www.npmjs.com/package/nock`.

[35] Trello. *Trello.* `https://trello.com/`.

[36] Santiago L. Valdarrama. *Is a Single-Page Application what you really need?*, 2014. `https://blog.svpino.com/2014/10/15/is-a-single-page-application-what-you-really-need`.

[37] Matt West. *Improving Your Development Workflow with Yeoman*, 2014. `http://blog.teamtreehouse.com/improving-development-workflow-yeoman`.

[38] Jaime Teevan William Jones. *Personal Information Management.* University of Washington Press, 2007.

[39] Yeoman. *The Web's Scaffolding Tool for Modern Webapps.* `http://yeoman.io/`.