University of Glasgow | School of Computing Science

Dragondrop

Level 4 Software Engineering project

Martynas Buivys, 1007100b

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Level 4 Project — March 18, 2015

Abstract

Bookmarking is a process that has high interaction costs. In the past 22 years there were no serious attempts to make bookmarking a natural everyday activity. The goal of Dragondrop project is to develop a web application that minimises the cost of bookmarking actions and provides a unified bookmark and web search experience by integrating bookmark management and sharing into a search interface. The application was evaluated in a usability study and the results indicate that the application is highly usable and provides a good user experience.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Contents

Chapter 1

Introduction

1.1 Motivation and context

Bookmarks were invented 22 years ago and have undergone several significant phase changes. In the beginning, bookmarks were simply a list of internet shortcuts. A decade later, they transformed into social sharing platforms such as Delicious and Digg that are used by millions of people.

Bookmarking has always been a separate activity that requires a certain barrier of entry. For example, in order to start using Delicious, a user must sign up on their website and learn how to use it. Similarly, Yahoo Toolbar plugin must be installed before a user can get started with using Yahoo Bookmarks [12].

In the past years the the usage of search engines such as Google and Bing became a daily activity for many internet users. Dragondrop project integrates social bookmarking into a search interface as an attempt to minimize the barrier of entry and turn it into an everyday activity.

This chapter discusses the advantages and disadvantages of existing bookmarking tools.

1.1.1 Static bookmarks

All modern browsers offer built-in bookmarking tools. A user can save pages of interest and organise them in folders in the browser's environment. Figure 1.1 illustrates a list of bookmarks in Google Chrome browser.

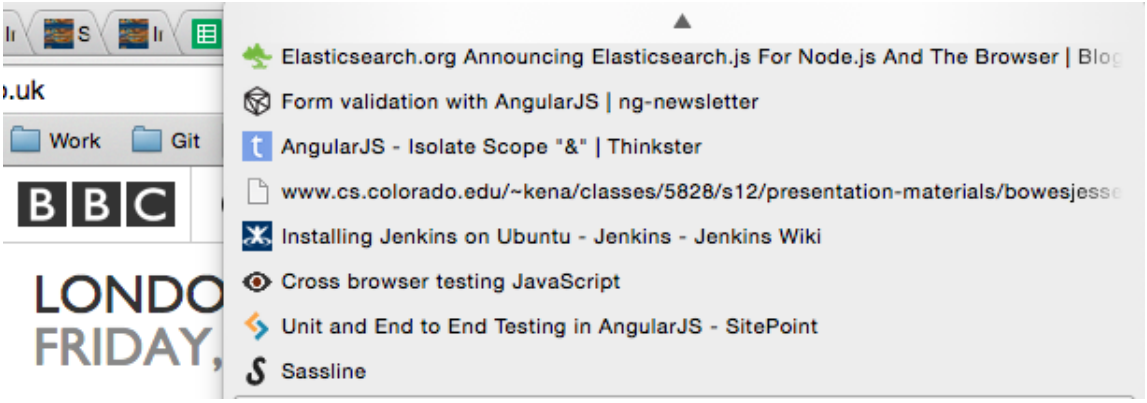

Figure 1.1: Static bookmarks in Google Chrome browser.

The problem is that all bookmarks are static and can only be accessed from the same machine and the same browser they originally were saved on. Furthermore, when the list of bookmarks grows large it becomes almost impossible to find the required bookmark. These issues have been addressed by Google Chrome Bookmark Manager.

1.1.2 Portable bookmarks

Google Chrome Bookmark Manager has a built-in search functionality that allows users to quickly retrieve relevant bookmarks. Furthermore, it associates all bookmarks in the browser's environment with the user's Google account. In this way, if multiple devices are used, the bookmarks are synchronized automatically.

On the other hand, the Bookmark Manager offers no improvement in bookmarking interaction costs. The bookmarking process is exactly the same as with static bookmarks. Moreover, there is no social sharing - all bookmarks are private.

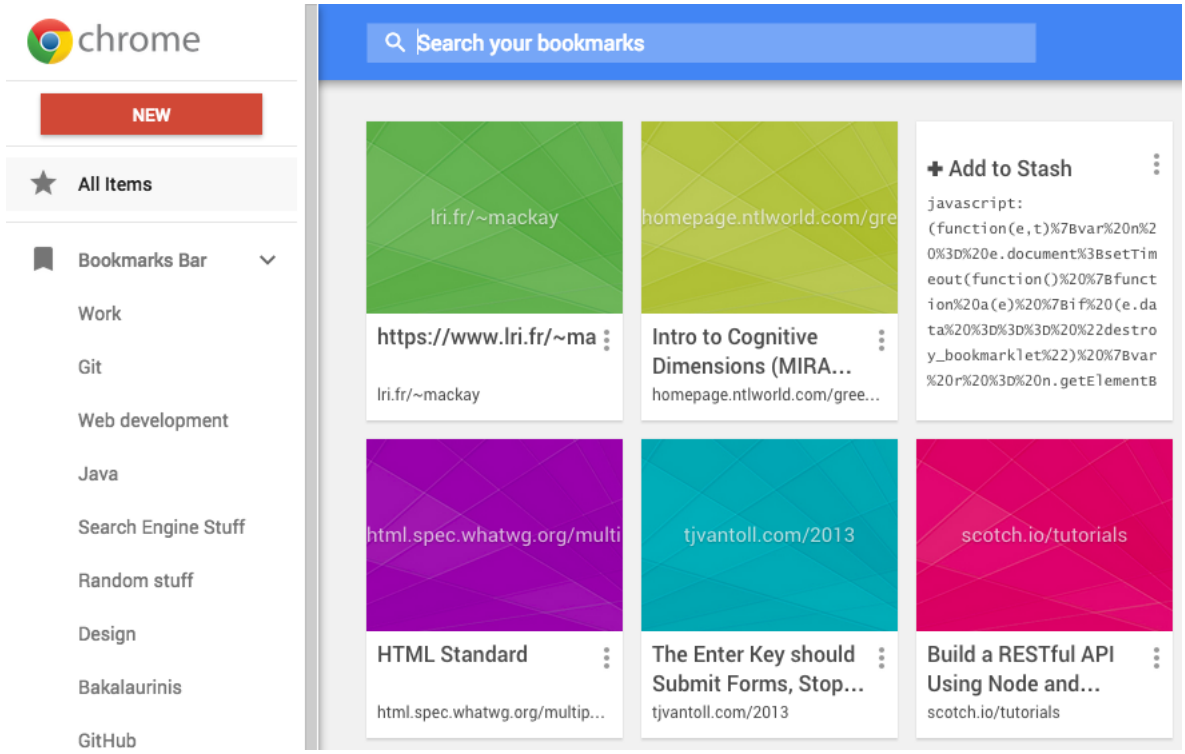

Figure 1.2: Google Chrome Bookmark Manager.

1.1.3 Social bookmarking

There is a large number of social bookmarking websites. Each of them uses social sharing aspect for different purposes. For example, Digg focuses on promoting the most interesting content on the web [3]. Delicious is a large online community that allows users to save and organize bookmarks in their platform [2].

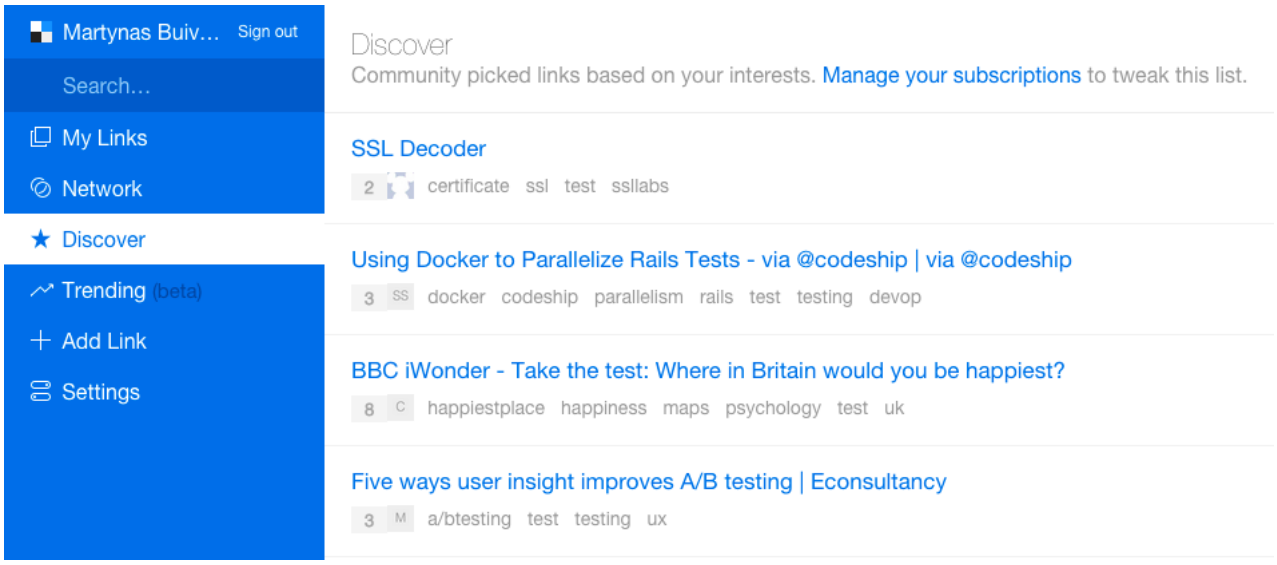

Figure 1.3: Delicious social bookmarking web service.

Delicious supports bookmark filtering at three levels: personal, network, and public. This allows the users to quickly discover what is trending on the web, as well as to follow people in their network. Delicious also provides users with a search functionality for private and public bookmarks.

Even though Delicious is a great bookmarking tool, it requires a lot of involvement from the users. In order to start using Delicious, a user needs to sign up for the service, learn how to use it, and install a browser extension.

1.2 Project description

Dragondrop project incorporates bookmarking experience into a search interface. The interface is designed to minimise the cost of users' actions and to to provide them with an enhanced search and result management capabilities. The idea is that the application should feature a minimalistic search interface and provide users with certain bookmarking capabilities.

Firstly, the users should be able to bookmark search results by dragging and dropping them on a folder. Secondly, the users should be able to make use of the search function to find previously saved bookmarks, as well as to discover other people's bookmarks. Finally, all bookmarks and bookmark folders should be easily accessible from the search interface. In addition to these features, the search interface should be able to provide folder name suggestions based on the search query. This will allow users to quickly bookmark search results under relevant folders.

The challenge of Dragondrop project is to implement these features and maintain a good user experience.

Chapter 2

Project requirements

2.1 Overview

The development of Dragondrop project was requested by Dr. Leif Azzopardi. The objective of the project is to reinvent bookmarking experience by integrating bookmarks into a search interface and evaluate its usability. This chapter identifies high level functional and non-functional requirements for the project. The functional requirements describe how an application should behave in order to support user goals and activities [20]. The non-functional requirements are concerned with the system users and their ability to effectively use the application.

2.2 Functional requirements

Account registration

The system should allow visitors to register on Dragondrop using their social media accounts.

Profile management

A user should be able to modify his profile details.

Bookmark saving

The web application should provide users with drag & drop bookmarking functionality.

Bookmark search

The application should provide users with an ability to search their existing bookmarks, as well as publicly available bookmarks that belong to other people.

Folder suggestions

The application should recommend the folders that best match a user's query. For example, given a search query ”Django”, the application should recommend a folder ”Python” if it contains bookmarks that are related to the query.

Folder management

A user should be able to create, edit, and remove bookmark folders. In addition to that, a user must have an ability to toggle the privacy setting of a folder and make it unavailable for other people to see.

Result sharing

System users should be able to share their bookmark lists with other people. This means that a user's profile and folders should have dedicated URLs.

2.3 Non-functional requirements

Availability

Dragondrop must be available to users at all times. The web application should have fast response times for users independent of their physical location.

Usability

The application must be easy to use and deliver good user experience.

Chapter 3

Design

3.1 Design strategy

Dragondrop project integrates bookmarks into a search interface. This change is a major deviation from the standards and, therefore, the application must be designed in a way that a user would be able to interact with the interface without any guidance. In other words, the web application interface should closely resemble popular search engines such as Google and Bing.

3.2 Application interface

Dragondrop website has been designed based on the idea that a good search interface design should be minimalistic and aesthetically pleasing [14]. A search engine is a medium for people to find information. When a user is looking at the search results, they are usually involved in some larger task, and do not want their flow of though interrupted by an intrusive interface [19]. Therefore, an initial design that follows the aforementioned principles was produced. Figure 3.1 illustrates the initial wireframe of Dragondrop application interface.

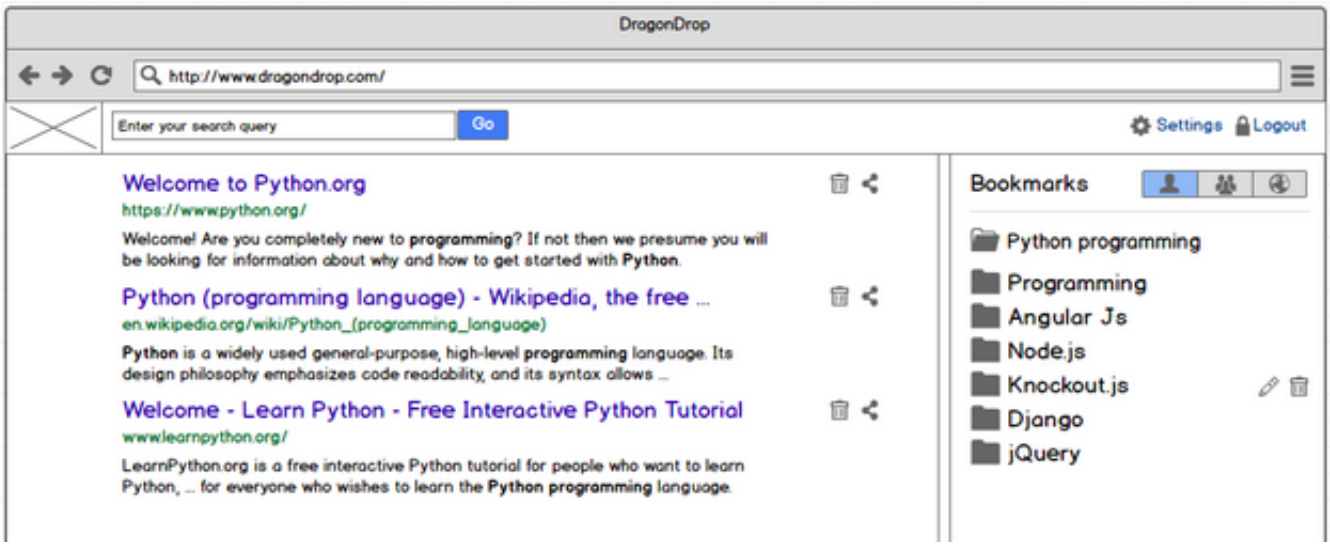

Figure 3.1: The initial wireframe of Dragondrop search interface.

3.3 Interface components

3.3.1 Header

Initially, it has been decided that the header will contain a logo, a search box, and two account navigation links. After several iterations, 3 search categories have been added to the header in order to facilitate switching between web and bookmark searches. The selected category is marked with a red underscore.

3.3.2 Sidebar

The sidebar has evolved into two sections: suggested folders and bookmark folders. Suggested folders are temporary ideas that change depending on the search query. If the search query is empty, the suggested folders are not shown. Both lists of folders are separated by descriptive headings. The headings are followed by question mark icons that show additional information to the new users. At the bottom of the sidebar there is a ”New folder” button that will invoke a popup dialog window for creating a new folder. The sidebar is illustrated in Figure 3.2 below.

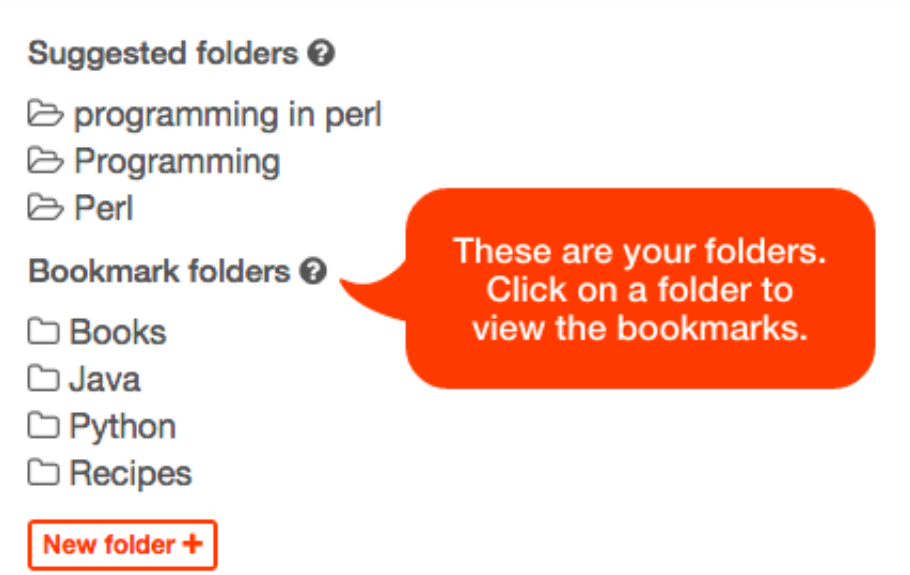

Figure 3.2: Sidebar design with folder suggestions and an activated tooltip.

3.3.3 Content

The content section displays the search results. A search result can be either a bookmark or a web page. Essentially, both are very similar and exhibit the same behaviour - they are linked with an external page. The main difference is that bookmark search results have two additional links on the right hand side of the URL. The first link points to the profile page of the bookmark owner, while the second link points to the folder the bookmark belongs to. The two different search result types are illustrated in Figures 3.3 and 3.4 below.

Figure 3.3: Web search result.

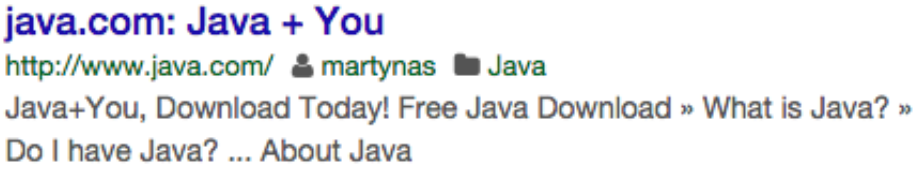

Figure 3.4: Bookmark search result.

3.3.4 Folder view

A folder view features a profile image of the folder owner, 4 control buttons, and a list of bookmarks that belong to the folder. Figure 3.5 illustrates the folder view.

Figure 3.5: Folder view.

3.3.5 Welcome message

Newly registered users are presented with a welcome message that is displayed in the content section. The message shows 3 interactive images that demonstrate the stand-out features of Dragondrop application. It can be easily switched off by clicking the button in the middle or by updating account settings.

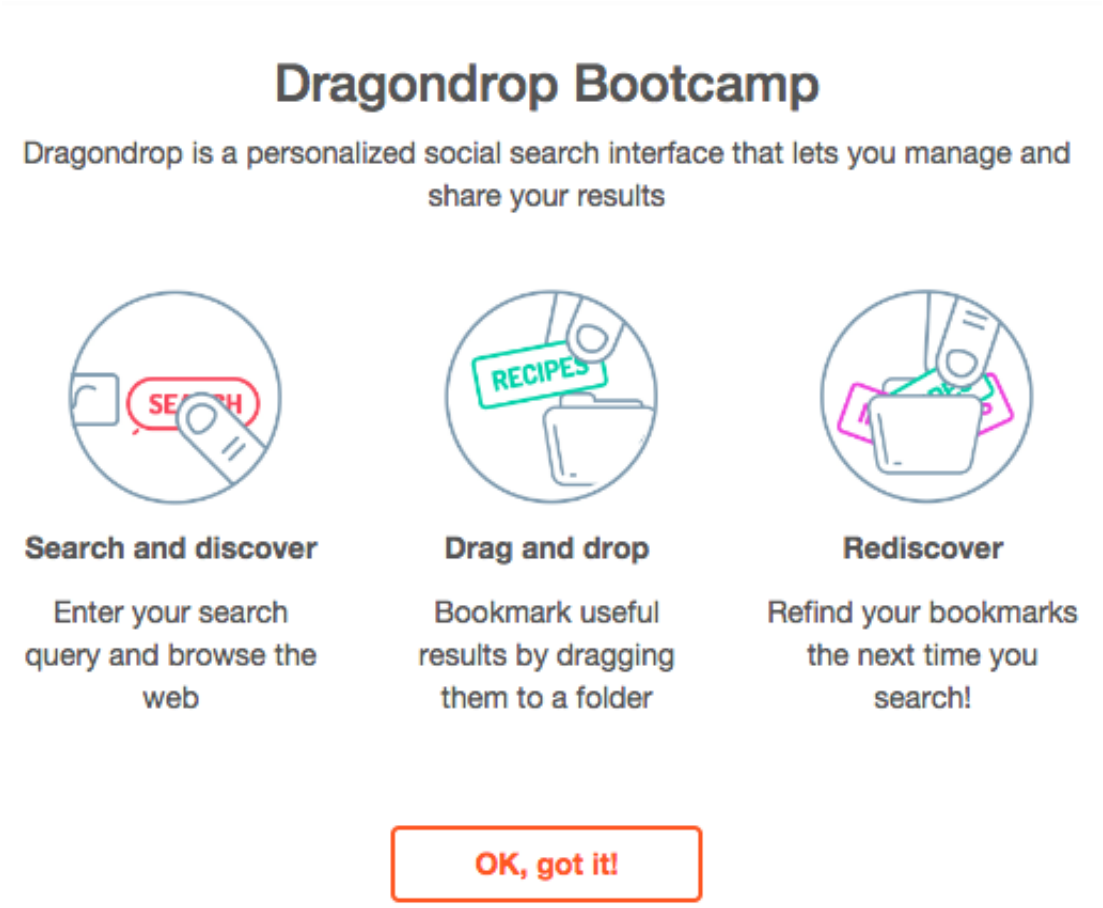

Figure 3.6: The welcome screen for new users.

3.3.6 User profile view

A profile page consists of a profile image, account handle, two action buttons, and a list of user folders. The ”Share” button invokes a popup dialog window that generates a dedicated link for the profile page. The Edit button allows the user to update their account handle. Figure 3.7 demonstrates the user profile view.

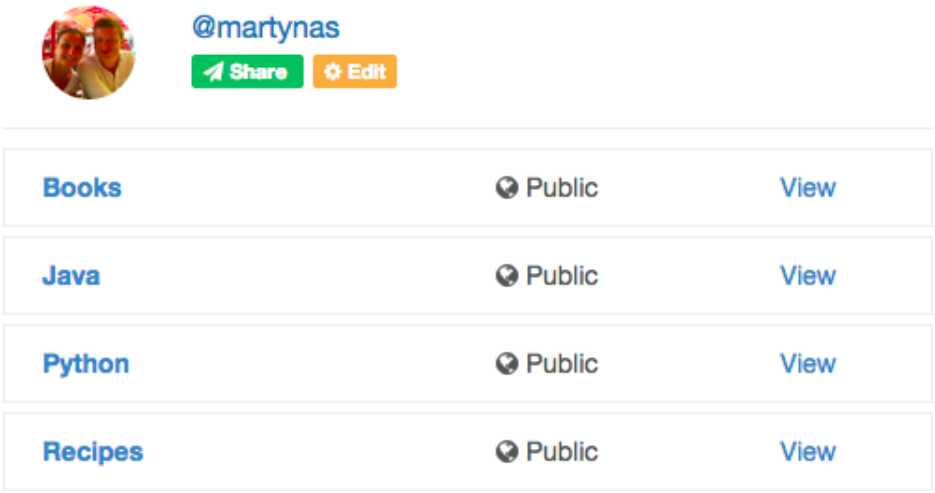

Figure 3.7: User profile view.

3.3.7 Responsive layout

Although responsive design is beyond the scope of Dragondrop project, a customized layout has been prepared for mobile and tablet device users. It supports the basic functionality of bookmark and web search. Figure 3.8 illustrates the search interface on a tablet.

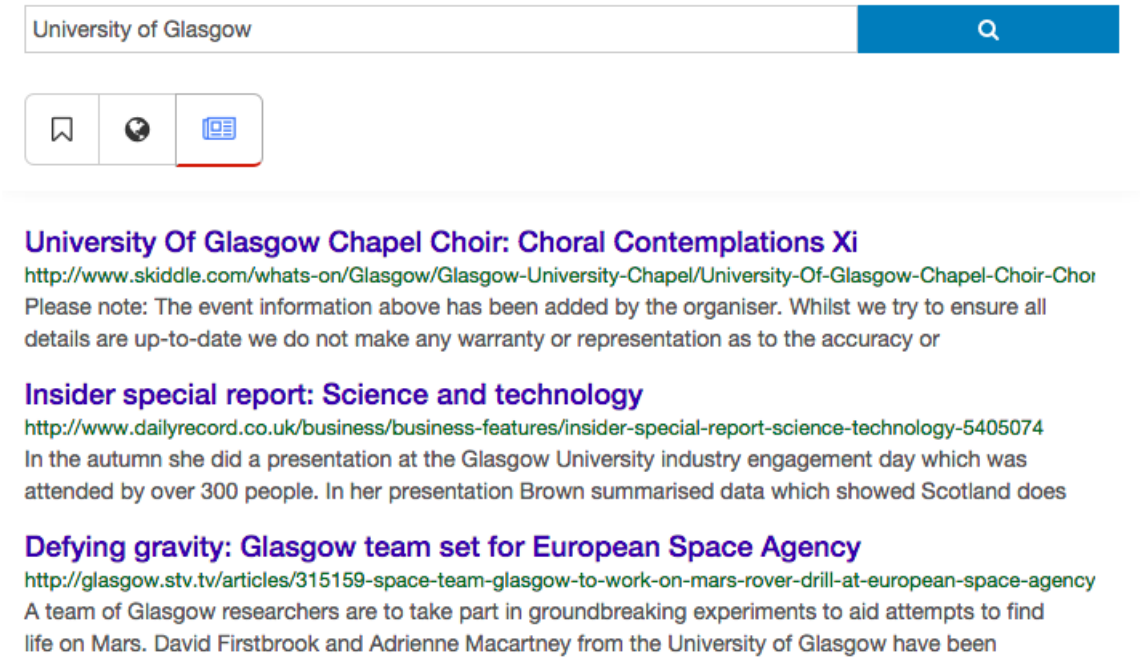

Figure 3.8: Dragondrop application in tablet vertical view.

Chapter 4

System architecture

4.1 Acronym reference

- HTML - HyperText Markup Language
- DOM - Document Object Model
- API - Application Program Interface
- Front-end - Web application component
- Back-end - API component
- CRUD - Create, Read, Update, Delete
- RQ - Redis Queue
- OAuth - Open Authentication
- CSS - Cascading Style Sheets
- AJAX - asynchronous JavaScript and XML
- YUI - Yahoo User Interface
- JSON - JavaScript Object Notation

4.2 High level overview

Dragondrop is a component-based system consisting of four main components.

Web application component

The graphical user interface that users interact with.

API component

The core system component that controls the communication between the user and the data.

Search component

A set of functions that support search in 3 verticals: web, news, and bookmarks. Although the functions are physically located in the API component, at a conceptual level they can be looked at as a separate entity. Therefore, the search functions will be referred to as the Search component.

Database component

A source-of-truth database that contains information about all bookmarks, folders, and users.

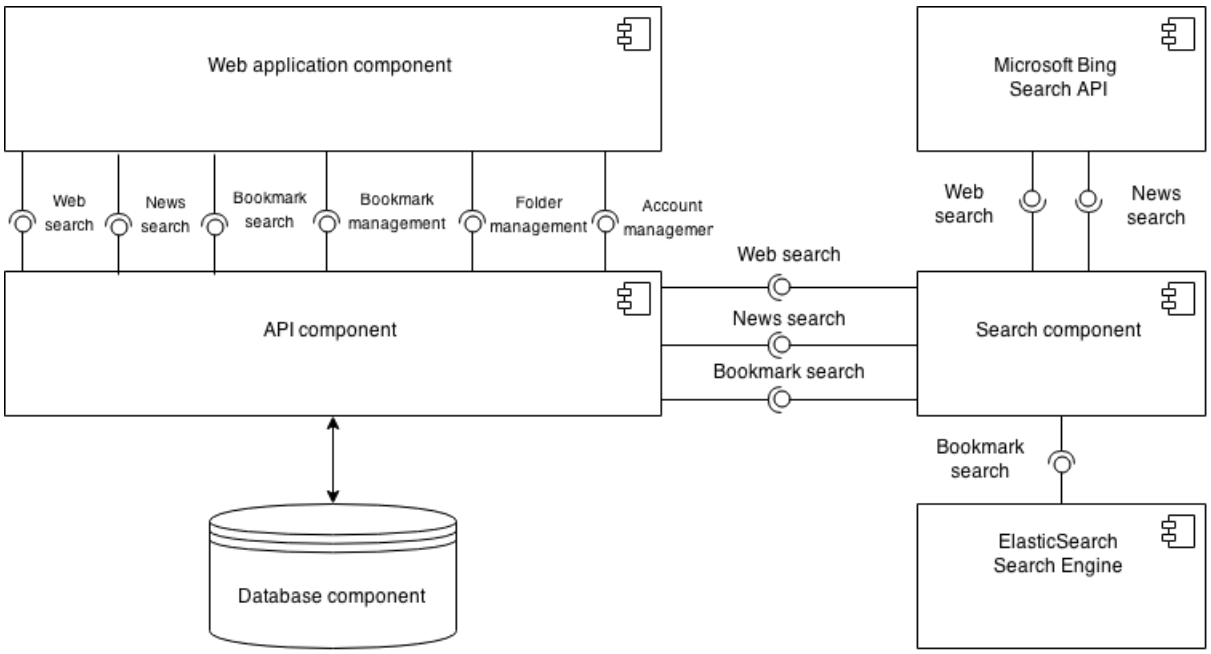

Figure 4.1: High level overview of components in Dragondrop system.

4.3 Web application component

The Web application component is the presentation layer for the data. It provides the users with with a set of HTML components (e.g. forms) that can be used to invoke search and bookmarking functions in the API component. Essentially, it is the search interface that users interact with.

4.3.1 Routing

The Web application component is a single-page website built using a JavaScript framework. The main advantage it offers is that pages (otherwise known as states) can be loaded without the need to reload the entire application. In other words, the application only retrieves the content it requires without the need to download static parts such as the header or the footer. In this way, the application provides the user with a seamless browsing experience.

The only disadvantage is that all of the project's code (HTML, CSS, and JavaScript) must be downloaded during the initial page load which might be noticeably slower.

The application has 4 different sates: Login, Search results, User profile, and Folder view. A user is in the Login state when there is no active session between the client and the server. After successfully logging in, a new session is registered and the user gets redirected to the Search results state. From this point, the user is able to navigate to any other state with a single click of a button.

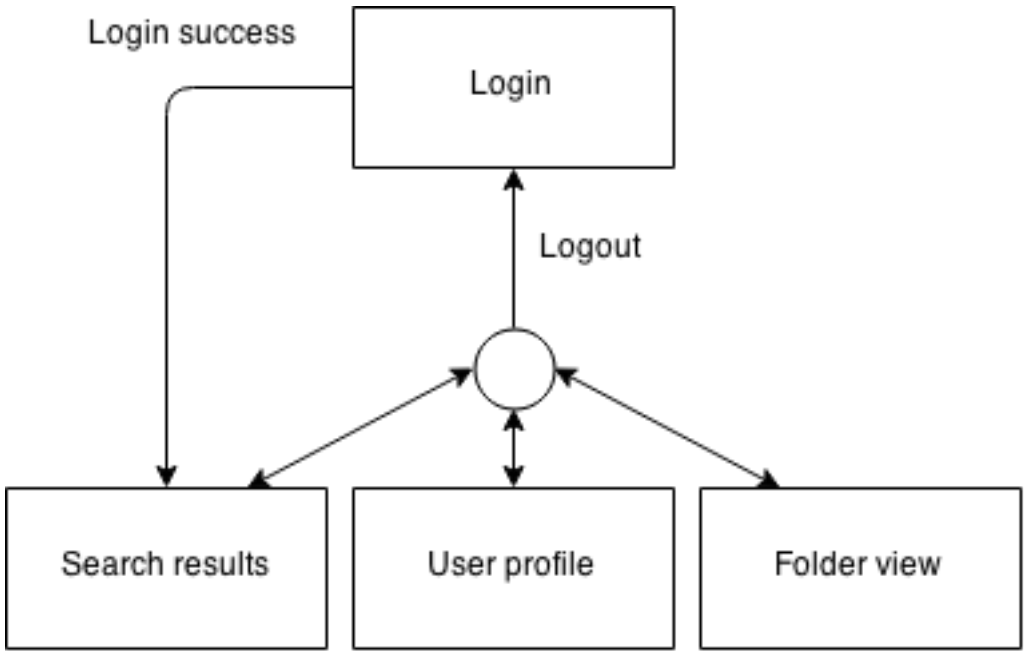

Figure 4.2: Possible state change directions in the Web application component.

4.3.2 JavaScript framework

When developing a web application from scratch, it is very important to choose the right technology from the very beginning. JavaScript code is easy to write, however it is difficult to write well. It is naive to rely solely on DOM manipulation libraries such as jQuery or YUI [13] because as soon as the application's logic becomes a little more complex, the developer ends up with a lot of poorly organised spaghetti code. Therefore, it is vital to use a JavaScript framework that will bring a clear structure to the project and reinforce good development practices. The trending frameworks in 2015 are Angular JS, Backbone.js, Ember.js, and Knockout.js. After a careful examination, Angular JS has been chosen for Dragondrop project because of its unique features, available support, and an easy-to-use template system. This section highlights key considerations that were made during the assessment process.

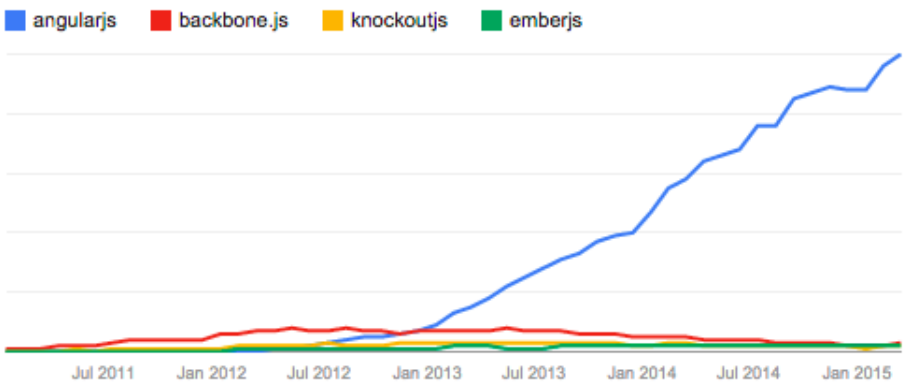

Figure 4.3: JavaScript framework trends. Source: Google Trends.

Angular JS is developed and continuously supported by Google. Its primary goal is to simplify the dynamic web application development through innovative concepts such as directives and two-way data binding. Directives are one of the most powerful features in Angular JS framework. They allow web developers to declare new reusable HTML tags. In addition to that, two-way bindings can save a lot of time because the developer is not required to produce boilerplate code that regularly synchronises the view with the model and the model with the view [1]. However, this can be achieved only by scanning all objects and comparing each property with the last known value. This can be a bottleneck for large applications. Therefore, Angular JS is mostly suitable for small and medium-sized projects.

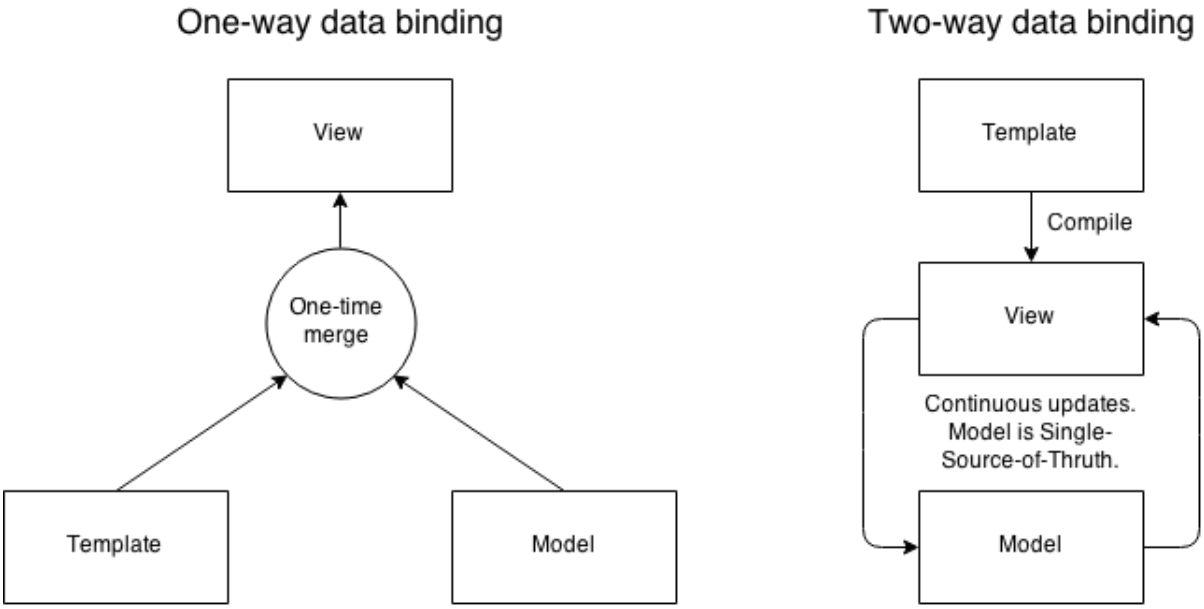

Figure 4.4: Classical one-way binding vs. two-way data binding in Angular JS

Community size and available support such as online tutorials is one of the most important aspects when choosing a tool for any project. The 4 candidate JavaScript frameworks have been evaluated in terms of developer involvement, available plugins, and active questions on StackOverflow. Angular JS stands out as the clear winner with the largest online developer community.

Angular JS and Knockout.js have a built-in templating mechanism while Ember.js and Backbone.js rely on

Criterion	Angular JS	Backbone.js	Ember.js	Knockout.js
Contributors	1205	252	464	50
Number of commits	6600	2800	8900	1300
Active pull requests	293	22	75	84
Stars	36355	21055	13087	6159
Plugins available	1285	257	740	62
StackOverflow questions	83204	17587	14360	13150

external string-based templating libraries. Angular JS templates are plain HTML containing data-* attributes that are combined with information from the model and controller to render a dynamic view. Ember.js makes use of Handlebars.js library that compiles string templates. String-based templates can be a great advantage as they can be compiled on the server side, thus significantly reducing application startup time. Also, Angular JS is required to traverse the entire DOM during loading time to aggregate rendering instructions and can be a problem for some large applications. On the other hand, Dragondrop is a lightweight application and would not be negatively impacted by the limitations mentioned earlier. Therefore, Angular JS and Knockout.js style template system would be most suitable for the project because of the simplicity they offer. The differences of each template system are highlighted below. Given an array ”people” consisting of 2 JavaScript objects, an unordered list will be produced.

Data	**Expected output**

`var people = [`
`  {name: "John Smith"},`
`  {name: "Jane Smith"}`
`];`
	`<ul>`
	`  <li>John Smith</li>`
	`  <li>Jane Smith</li>`
	`</ul>`

Angular JS	 <li data-ng-repeat="person in people"> {{person.name}}
Knockout.js	<ul data-bind="foreach: people"> <li data-bind="text : $data.name">

Handlebars
 	
used by Ember.js
	Handlebars.registerHelper('list',
	function(items, options) {
	var out = " ";
	for(var i=0, l=items.length; i<l; i++) {
	out += " ";
	out += options.fn(items[i]);
	out += " ";
	}
	return out + " ";
	});

Underscore.js used by Backbone	 <% _.each(people, function(person) { %> <%= person.name %> <% }); %>

4.3.3 Additional JavaScript libraries

Dragondrop web application depends on 4 additional JavaScript libraries that provide the interface with various interactive capabilities.

jQuery a versatile JavaScript library that offers a simple and easy-to-use API for DOM traversal, manipulation, and event management.

jQuery UI a rich collection of user interface widgets and interactions built on top of jQuery. Dragondrop makes use of Draggable and Droppable interactions to allow the user to drag and drop search results to the bookmark folders.

UI Bootstrap a set of useful Bootstrap components adapted for Angular JS framework. Dragondrop uses Modal component for creating custom popup dialogs.

UI Router an alternative routing library for Angular JS applications. Its main advantage over the native routing mechanism is the ability to have nested views.

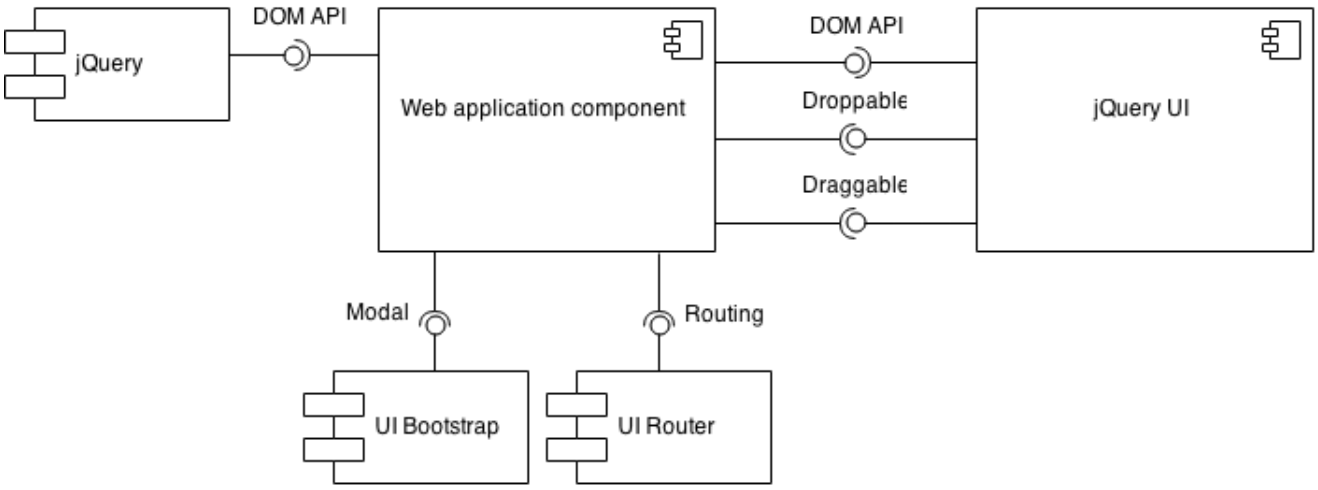

Figure 4.5: JavaScript dependencies in the Web application component

4.3.4 CSS framework

The Web application component uses Bootstrap CSS framework library. It offers a very well documented 12 column grid system that allows rapid development of responsive websites. Furthermore, the library makes websites look more consistent by providing many built-in CSS rules for the most common HTML elements such as forms, buttons, labels, and tables. Therefore, Bootstrap is an ideal CSS framework for Dragondrop project.

4.4 API component

The API component provides an interface for the Web application component to communicate with the Database and the Search component through a JSON API [5]. The API supports all standard CRUD (Create, Read, Update, Delete) operations for accessing and updating bookmarks and folders. It also provides specific endpoints for searching the web, news, and bookmark verticals, as well as retrieving and updating user account information. The list of API endpoints is summarized in the table below.

Table 4.1: API endpoints

Endpoint	Description
/folder/all/	Retrieve all user's folders
/folder/create/	Create a new folder
/folder/read/	Retrieve a folder with all bookmarks in it
/folder/update/	Update folder details
/folder/delete/	Delete a folder
/folder/togglevisibility/	Toggle public visibility of a folder and its bookmarks
/bookmark/all/	Retrieve all user's bookmarks
/bookmark/create/	Create a new bookmark
/bookmark/read/	Retrieve a single bookmark
/bookmark/update/	Update a bookmark
/bookmark/delete/	Delete a bookmark
/search/bookmarks/	Search the bookmarks vertical
/search/web/	Search the web vertical
/search/news/	Search the news vertical
/user/read/	Retrieve user profile information
/me/handle/	Change account handle
/me/toggletutorial/	Switch the tutorial on or off

4.4.1 Authentication layer

The API implements a proxy design pattern by controlling the communication between the Web application component and the data store. It acts as an authorisation layer for all requests to the Database and the Search component. The API handles all GET and POST requests in a way so that the client only has access to the information that he can read or modify. For example, a user must not be able to delete a folder that belongs to another user. Furthermore, the API must orchestrate bookmark search requests so that no private bookmarks that belong to other users are returned. This is achieved by wrapping the original query with a custom Lucene search query.

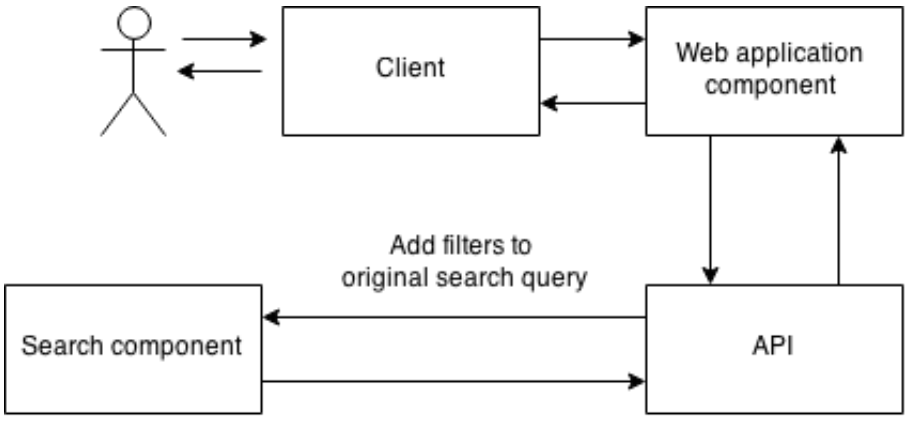

Figure 4.6: API adds additional query parameters to take into account users' privacy settings.

4.4.2 Bookmark indexing

Furthermore, the API is responsible for indexing all searchable bookmark information. This means that for each bookmark in the database, the API is responsible for maintaining a corresponding searchable record in the Search

component. This feature introduces two new problems. Firstly, the API needs to process the requests in as little time as possible. Secondly, it must ensure data consistency between the Database and the Search component. Two ways of approaching this problem were considered.

The first approach involves using Redis Queue [8] library for enqueueing data synchronization jobs. For example, add each incoming bookmark modification request to the queue and allow a background worker to process it on a separate thread. This is a preferred implementation in the long run, however it introduces new dependencies that are an unnecessary overhead in the early stages of development.

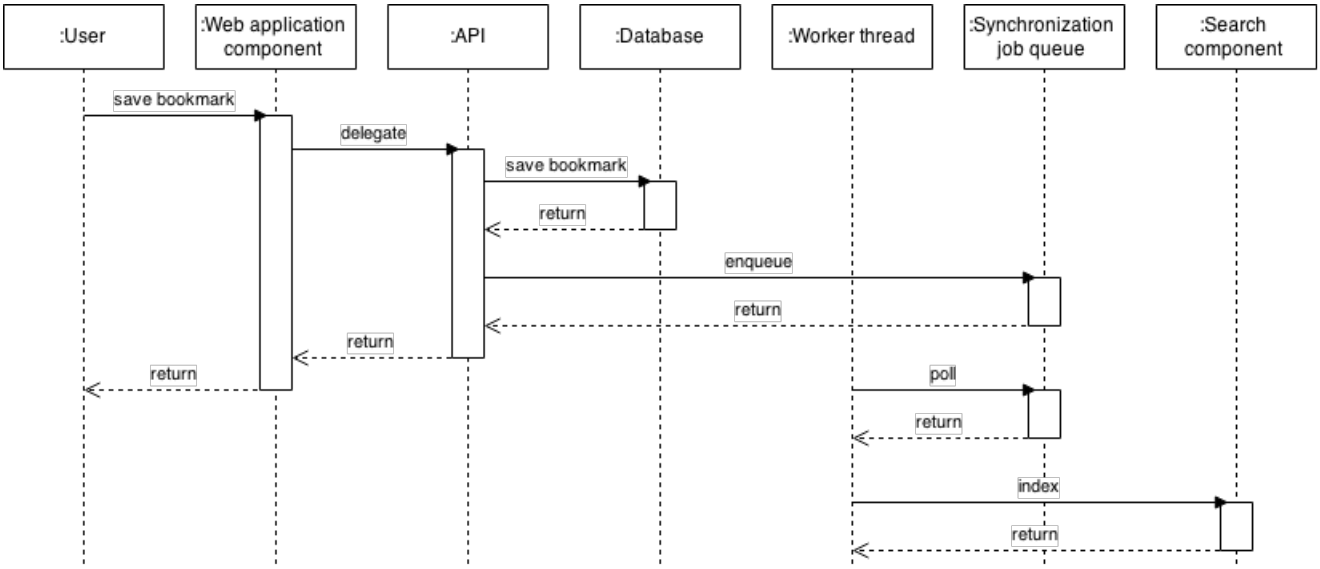

Figure 4.7: A sequence diagram illustrating the complete bookmark saving and indexing procedure using Redis Queue.

The second approach is very straightforward and does not require any additional libraries. As soon as a bookmark modification request is received, the handler function executes a blocking update call to the Search component. Although this increases the response time of the API, the delay is insignificant and will not be a problem until the application is used by thousands of users. Therefore, the feature was implemented using the second approach because of its suitability in the early stages of the application.

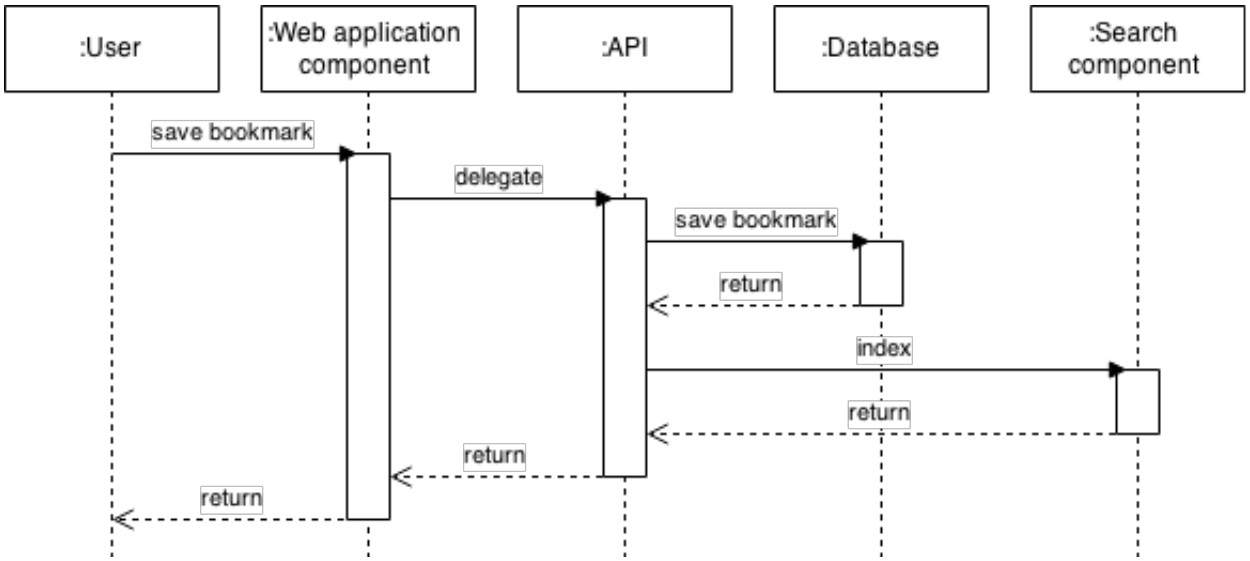

Figure 4.8: A sequence diagram representing the complete bookmark saving and indexing procedure using blocking calls.

4.5 Database component

The Database component in Dragondrop application stores information about all user accounts and their bookmarks. It is a PostgreSQL database that runs in Amazon cloud.

The Profile entity stores the information about the social media account that was used during the account registration process using OAuth2 [6]. For example, when a Google account is used to sign up for the application, the user's account handle, identifier, and profile image are automatically retrieved from Google's database. This information is then used for constructing a user profile page on Web application component. The Folder entity contains information about each folder's meta information such as folder name, folder slug, and privacy setting (either publicly visible or not). The Bookmark entity collects information about the bookmarked page: the URL, page title, and meta description. Each bookmark also has a reference to the folder they belong to. These entities and their relationships are illustrated in Figure 4.9.

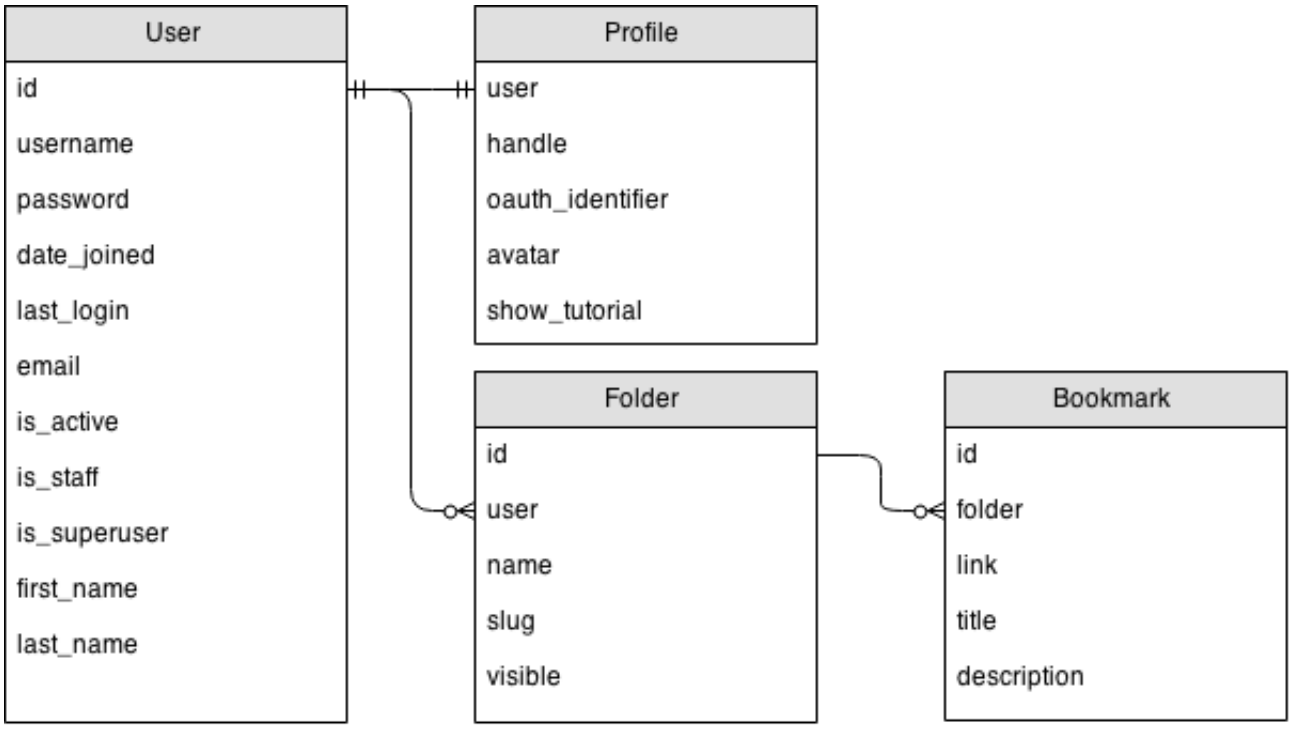

Figure 4.9: Entity-relationship diagram for Dragondrop application.

4.6 Search component

The Search component is responsible for retrieving documents based on keywords in a given search query. More importantly, the documents must be retrieved quickly and satisfy the user's information need. Furthermore, the Search component is required to support searches in 3 different verticals: web, news, and bookmarks. All web search and news search requests are delegated to Microsoft Bing Search API. Therefore, the main challenge for Search component is to provide a bookmark search functionality. Basic string matching using SQL queries is slow, ineffective, and does not scale. Therefore, a search engine must be used to overcome these limitations.

ElasticSearch search engine was chosen for Dragondrop project. It offers an architecture designed for creating modern realtime search applications [26]. Furthermore, it is highly scalable, well documented, and easy to set up. This section overviews the initial ElasticSearch server configurations.

ElasticSearch can be configured to use a specific similarity module for scoring documents that match a query. The default similarity module makes use of DefaultSimilarity algorithm in Lucene; however, other popular algorithms such as BM25 are available and can be easily enabled at a later point in the future.

Documents are provided by the API component and the Search component only needs take care of indexing and search. A bookmark document is represented by the same fields as the corresponding bookmark in the database: link, title, description, and folder. However, the importance of the terms that appear in these fields is different. For example, terms that appear in the title of a bookmark document tend to better describe the content of the linked page. Therefore, the relevancy of the title field is set to be two times greater than the relevancy of the description field.

Each bookmark record also has 3 additional meta information fields: ”id”, ”visible”, and ”user”. The ”id” field is a unique bookmark document reference in the Search component and it is exactly the same as the ”id” of the corresponding bookmark in the Database component. It is used by the API during document creation and

Table 4.2: Document mappings summary

Field	Type	Boost factor
Title	string	4
Description	string	2
Link	string	1
Folder	string	1

update operations. The ”visible” and ”user” fields are needed to support more complex queries that need to take into account the user's privacy settings. For example, the query below will retrieve all user's bookmarks that are relevant to search query ”Glasgow”, as well as other relevant and publicly available bookmarks.

```
((user:1) AND ("Glasgow")) OR ("Glasgow" AND visible:true AND (NOT user:1))
```

The search engine makes use of two analysers. The first analyser is a simple Whitespace Analyser built using a Whitespace Tokenizer that splits strings at whitespace characters. It is used for search query parsing. The second analyser is a custom n-gram analyser for the index. It takes a string and produces a series of close variants. The example below demonstrates the analyser in action given a query ”joe”.

```
> curl
> -XGET 'http://localhost:9200/index/_analyze?analyzer=nGram_analyzer'
> -d 'joe'
{
    "tokens": [{
        "token": "jo",
        "start_offset": 0,
        "end_offset": 3,
        "type": "word",
        "position": 1
    }, {
        "token": "joe",
        "start_offset": 0,
        "end_offset": 3,
        "type": "word",
        "position": 1
    }, {
        "token": "oe",
        "start_offset": 0,
        "end_offset": 3,
        "type": "word",
        "position": 1
    }]
} %
```

Chapter 5

Application testing

5.1 Strategy

Testing plays a significant role in the software development life cycle. It is intended to demonstrate what a program does and to discover program defects before it is put into use [25]. Dragondrop application tests the core functionality of the system using two methodologies: unit testing and end-to-end testing. The system has 3 testable elements: Angular JS controllers and services, the graphical user interface (the website), and the API. The API endpoints are not covered by unit tests because of an upcoming migration to a RESTful API (see Future Work section in the final chapter). All test cases are oriented towards low implementation cost and high gain.

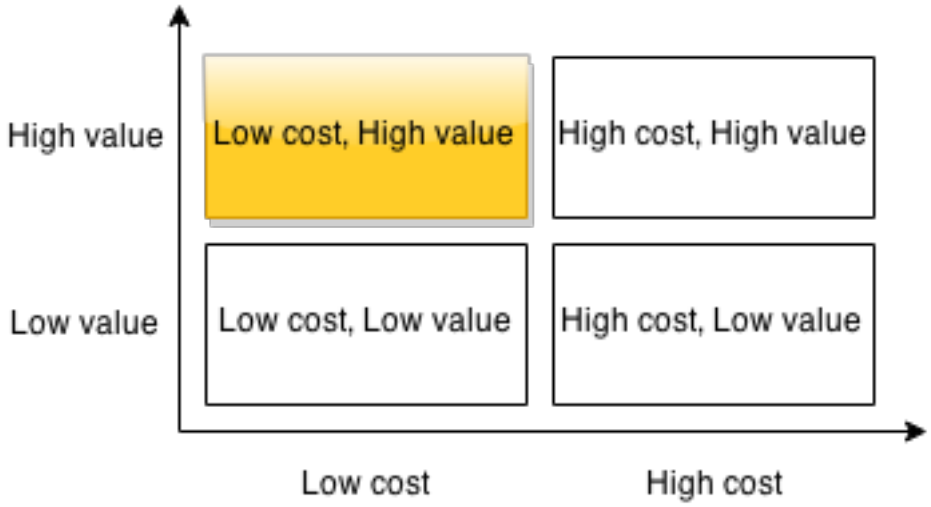

Figure 5.1: Testing strategy in Dragondrop application.

5.2 Approach

End-to-end testing (or integration testing) focuses on realistic end user scenarios. The goal is to test whether the entire application works as intended. Dragondrop application has various scenarios that check if all system components are integrated correctly and users see the elements they are supposed to see. For example, when a user types in a search query and searches the web, a list of search results should be displayed on the screen.

All end-to-end tests are run in Google Chrome browser and they interact with Dragondrop application the same way as a user would. While this is acceptable in the application development stage, the tests in production mode should be run in the cloud using a provider like Sauce Labs [9]. The reason for this is that browsers interpret HTML, CSS, and JavaScript code differently. For example, the search results might be visible in Safari, Mozilla Firefox, and Chrome browsers but invisible in Internet Explorer. Sauce Labs offer automated cross browser Selenium [10] testing. With this idea in mind, Protractor [7] end-to-end testing framework has been chosen for the project. It is designed for Angular JS applications and can be easily integrated with any cloud testing service provider.

Protractor is easy to set up and has a well documented API. It is built using Node.js and, therefore, uses Jasmine, a behaviour-driven development framework for testing JavaScript code. Jasmine features a clean and easily comprehensible syntax. An example test suite is demonstrated below.

```
describe("A suite", function() {
  it("contains spec with an expectation", function() {
    expect(true).toBe(true);
  });
});
```

The end-to-end tests in Dragondrop application replicate various end user scenarios ranging from account registration to folder creation and bookmarking interaction simulation. Although integration tests are great for assuring that the system is working as expected, they are unsuitable for addressing edge cases. Therefore, the end-to-end tests must be backed unit tests.

Unit testing focuses on testing the functionality of objects and methods [25]. Dragondrop application tests Angular JS controllers and services using Karma test runner. It supports Jasmine testing framework that is used in unit test specifications.

Chapter 6

Studies and analysis

6.1 Heuristic evaluation

During the application development stage there was not enough time and resources available to conduct a usability study and recruit users. Instead, heuristic evaluations were conducted on a regular basis. For example, the application was demonstrated to experts at Google in order to gain an insight from people who are experts in the field but are not involved in the project. Furthermore, the user interface was presented to the project supervisor on a weekly basis to collect feedback about newly implemented features. The reviews are conducted informally in the form of one-on-one interviews. This approach is very cost and time effective and enables rapid development.

6.2 Usability study

A formative usability test was conducted at the University of Glasgow. The main goals were to evaluate the usability of the web application interface and to find as well as quantify usability problems. Twelve people were invited to take part in the experiment in order to ensure the majority of usability problems are discovered in a single experiment [23]. The participants in this study were 9 male and 3 female students in Computing Science.

The usability study employed an iterative approach recommended by J. Nielsen [22]. The experiment was broken down into two smaller tests and the participants were divided into two groups of six people. The most obvious usability issues were fixed immediately after the feedback from the first group was collected and analysed. This enabled the second group of participants to point out new usability problems. Furthermore, the comparison of the performance between the two groups allowed to see if the fixes after the first experiment made a positive impact to the usability of the system.

6.2.1 Procedure

The experiment consists of 8 simple tasks that cover different aspects of Dragondrop web application. After completing a task, each participant was asked to fill out a questionnaire to evaluate the usability of a particular feature of the system. After the test the participants were asked to complete a 10-item System Usability Scale (SUS) questionnaire that provides a view of perceived usability [24]. The task scenarios are summarized in the list below.

Task 1 Account registration: create an account and sign in to Dragondrop web application.

Task 2 Customize user profile: update account handle and switch off the tutorial.

Task 3 Web search: search the web and news verticals for latest political events.

Task 4 Bookmarking (1): find 3 pizza recipes and bookmark them.

Task 5 Folder organization: change folder name and update folder's privacy setting.

Task 6 User space: navigate to user profile, obtain the sharing link, and visit it.

Task 7 Bookmarking (2): find 3 new pizza recipes and bookmark them.

Task 8 Bookmark search: use the search box to re-find the bookmarks saved in tasks 4 and 7.

6.2.2 Iteration 1

The first iteration went successfully and various usability issues were uncovered. The problems and actions taken are summarized in the list below. Figure 6.1 illustrates the number of times each problem was encountered.

Problem 1 Participant selected "News" vertical before typing in a search query.

The first problem occurs when a user selects "Web" or "News" vertical before typing in a search query. Dragondrop search interface has been designed to automatically switch to "Bookmarks" vertical when a search query changes. Therefore, as soon as a user starts typing, the search vertical settings get automatically reset. Unfortunately, there was not enough time to implement the fixes before the second iteration. However, the most straightforward solution to the problem would be to make search verticals "sticky". This change would reduce the flexibility of the search function but make the interface much more predictable.

Problem 2 Participant lost the search results after clicking on a result.

The second problem emerges when a user clicks on a search result, visits the linked page, and clicks the "Back" button. The search results are loaded through an AJAX request and they do not have a linkable state (explicit URL). Thus, a user returns to a blank search result page. This problem was solved by altering URL target attribute for all search results to open linked pages in a new tab or window.

Problem 3 Participant created a folder with a name that is not suitable for the long-term usage.

The third issue was encountered by 4 participants out of 6. It occurs when a user searches the "Web" or "News" verticals and prepares to save a search result. At that moment in time, a user sees 10-50 search results, a folder recommendation section and an empty list of folders. The participants did not have any bookmarks, therefore the folder recommendation section could only suggest one folder which was equivalent to the name of the query. This means that if the search query is "Margherita pizza recipes", the only suggested folder available would be "Margherita pizza recipes". In most cases, this folder name would not be suitable for a user's long-term bookmarking needs and he would be forced to rename it to something more generic. A simple solution to this problem was to split the search query string at space and suggest individual terms as folder names. For example, given the aforementioned search query, the list of suggested folders would be populated with 3 additional items: "Margherita", "Pizza", and "Recipes".

Problem 4 Participant attempted to create a folder before making a search query.

Problem 4 demonstrated that some users would prefer creating a folder before issuing a search query. For example, one participant tried double-clicking the sidebar to create a new folder. At that time, the only way to create a folder was through a search interface. Therefore, in order to address this limitation, a ”Create new folder” button was added to the sidebar.

Problem 5 Participant tried submitting a form with an ”Enter” key press.

Dragondrop web application uses popup dialogs to collect user input when editing account handle, renaming a folder or creating a new folder. Problem 5 emerged when 2 participants changed their account handle and hit ”Enter” key to submit the form and close the popup dialog window. Implicit form submission by hitting ”Enter” key is a web standard [4]. Therefore, this feature was incorporated into Dragondrop application after the first iteration.

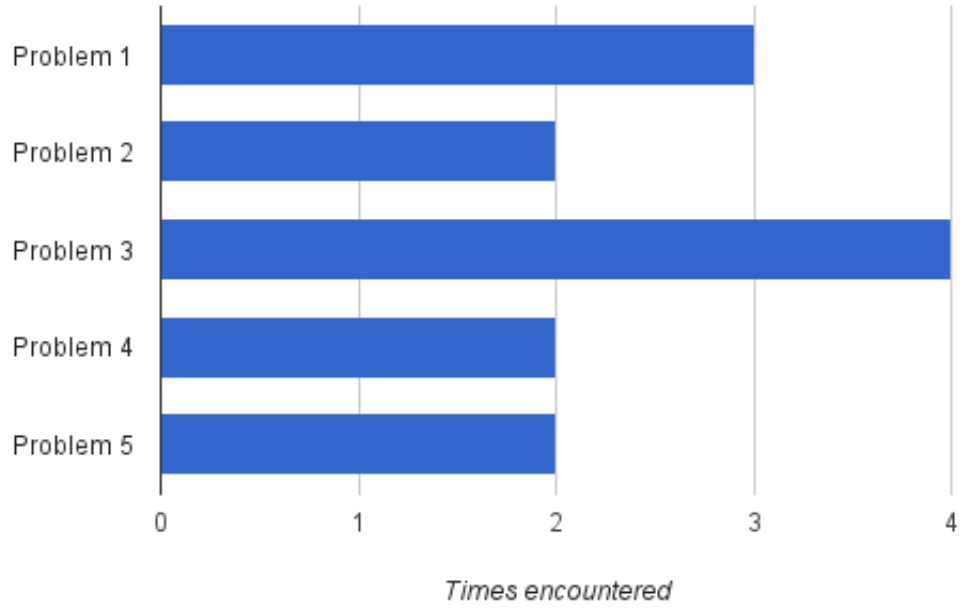

Figure 6.1: Number of times each problem was encountered in Iteration 1.

6.2.3 Iteration 2

The second iteration went well and a few new user interface issues were discovered. The 5 most frequently encountered problems and planned solutions are described in the list below. Figure 6.2 shows the number of times each problem was encountered in Iteration 2.

Problem 1 Participant could not identify action buttons in Folder state.

Dragondrop web application allows users to make changes to their bookmarks with the help of action buttons in the Folder state. There are four action buttons available to users: ”Privacy”, ”Share”, ”Edit”, ”Delete”. One participant said that the action buttons look like labels, rather than clickable folder controls. Other two participants complained that the buttons are too small and hard to notice. Therefore, the size of the buttons will be increased.

Problem 2 Participant selected ”News” vertical before typing in a search query.

This is exactly the same problem that emerged for 3 participants in the first iteration. It was encountered by 2 out of 6 people in Iteration 2. The search interactions will be redesigned as discussed in Iteration 1.

Problem 3 Participant created a folder with a name that is not suitable for the long-term usage.

The participants created a folder called ”Margherita pizza recipes” that was supposed to be given a generic name such as ”Pizza recipes” or ”Recipes”. Although this problem is the same as in Iteration 1, the circumstances are completely different. The problem occurred when two participants failed to follow the rules in Task 4 and started taking steps before fully understanding the task completion criterion. Therefore, this problem is ignored and no action will be taken.

Problem 4 Participant tried to save a search result into a new folder by dragging it on the ”Create new folder” button.

Problem 4 was caused by the ”Create new folder” button that fixed a usability problem after Iteration 1. Two participants tried dragging a search result on the button, expecting to create a new folder and save the bookmark. This functionality is currently not supported and the users had to bookmark the page the standard way. However, this feature could be a useful addition and will be implemented in the near future.

Problem 5 Participant tried to click on one of the suggested folders.

First-time Dragondrop users struggled to distinguish Suggested Folders from Bookmark Folders. They thought that the Suggested Folders are actually their Bookmark Folders and tried clicking on them. The only visual difference between the two lists is that they appear under different headings. Therefore, the icons for the suggested folders will be changed to eliminate the confusion.

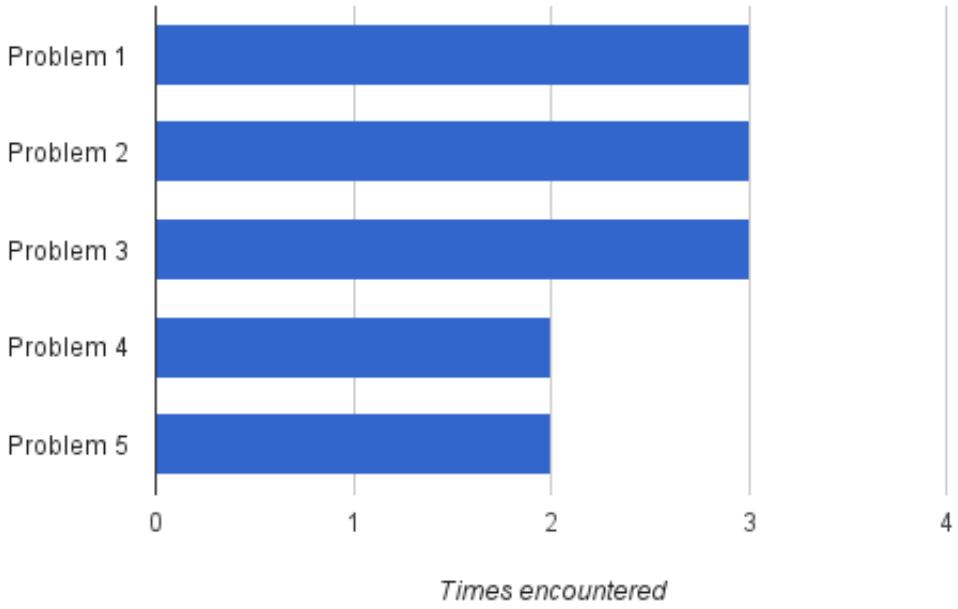

Figure 6.2: Number of times each problem was encountered in Iteration 2.

6.2.4 Task performance

Task performance results have been gathered, analysed, and compared. Table 6.1 summarizes the average task completion times in both iterations. The sample sizes were very small and, therefore, the performance differences are insignificant for the majority of tasks. However, since the changes after the first iteration were mainly oriented towards improving the bookmarking experience and simplifying folder creation, the task completion times for Tasks 4 and 7 demonstrate a significant improvement. For example, on average, Task 4 in the second iteration was completed 71 seconds faster. Similarly, the mean task completion time for Task 7 improved by 13 seconds.

Table 6.1: Average task completion times in seconds

	Iteration	Mean	σ	95% Conf. interval
Task 1	1	22.67	14.53	[11.04, 34.29]
	2	12.50	6.98	[6.92, 18.08]
Task 2	1	38.33	14.47	[26.75, 49.91]
	2	41.50	24.84	[21.62, 61.38]
Task 3	1	37.83	7.96	[31.46, 44.20]
	2	35.33	12.64	[25.22, 45.45]
Task 4	1	152.67	52.40	[110.74, 194.59]
	2	81.33	25.48	[60.94, 101.72]
Task 5	1	17.33	9.03	[10.11, 24.56]
	2	18.67	3.50	[15.86, 21.47]
Task 6	1	25.50	7.45	[19.54, 31.46]
	2	24.50	10.43	[16.16, 32.84]
Task 7	1	50.50	30.69	[25.94, 75.06]
	2	37.50	13.35	[26.82, 48.18]
Task 8	1	19.50	13.92	[8.36, 30.64]
	2	22.67	14.72	[10.89, 34.44]

6.2.5 Learnability

The task scenarios in the experiment were designed so that the learnability aspect of Dragondrop application could be recorded. Tasks 4 and 7 required the participants to bookmark 3 pizza recipes. By comparing the task completion times for the repetitive tasks, it was observed that the mean task completion times decreased by 67% and 54% for Iterations 1 and 2 respectively. The data is summarised in Table 6.2 and the difference in task completion times is illustrated in Figure 6.3.

Table 6.2: Average task completion times in seconds

	Task 4	Task 7	Decrease
Iteration 1	152.67	50.50	67%
Iteration 2	81.33	37.50	54%

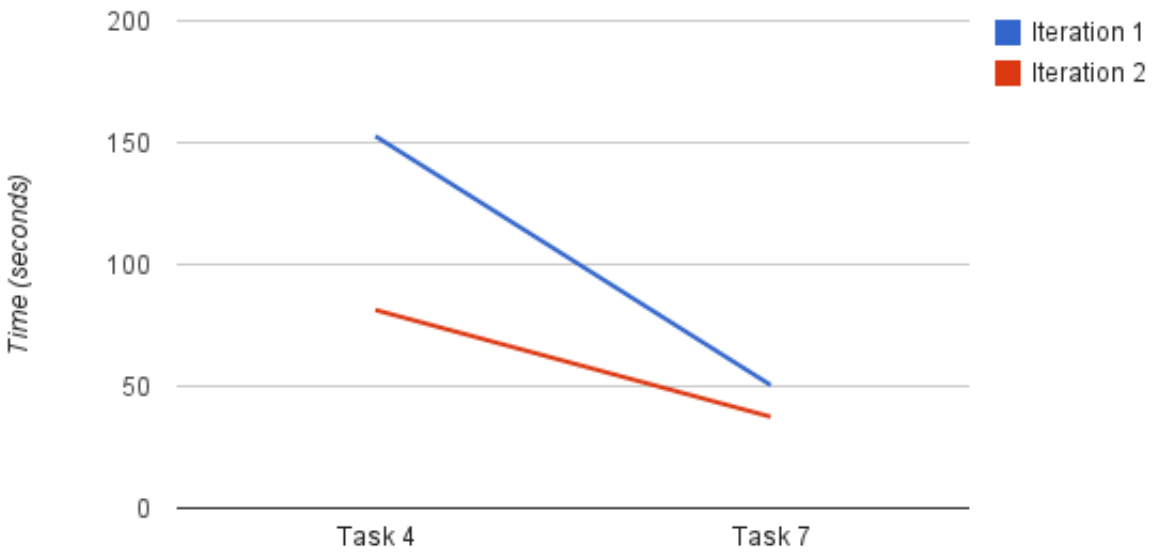

Figure 6.3: Average task completion times for repetitive tasks.

6.2.6 System Usability Scale scores

Post-test questionnaires are a great way for gathering information about participant's experience with the application as a whole [24]. System Usability Scale (SUS) is the most widely used post-test questionnaire based on Likert Scale. It contains 10 carefully designed questions that force the respondents to read each item and think whether they agree or disagree with it [16]. The results can be transformed into a score ranging from 0 to 100. The average SUS scores are summarised in Table 6.3

Table 6.3: SUS scores summary

Iteration	Mean (score)	σ	95% Confidence interval
1	87.50	13.69	[76.54, 98.46]
2	92.08	7.14	[86.37, 97.80]

It was estimated that, based on 500 studies, the average SUS score is 68. Dragondrop application scored 87 and 92 in iterations 1 and 2 respectively. These results indicate that the perceived usability of the system is well above average and participants enjoyed using the web application. On the other hand, all test participants were Computing Science students or researchers in Information Retrieval field. Their experience in usability experiments and familiarity with search engine interfaces is significantly higher than of a regular user. Therefore, these results do not represent the entire population and a more representative study needs to be conducted.

Figure 6.4 illustrates the differences between SUS in both iterations. The highest and the mean SUS scores are slightly better in the second iteration. However, the first iteration has much more consistent scores and one

outlier that negatively affects the mean SUS score. Therefore, there is not enough evidence that the second iteration provided the test participants with a significantly better user experience.

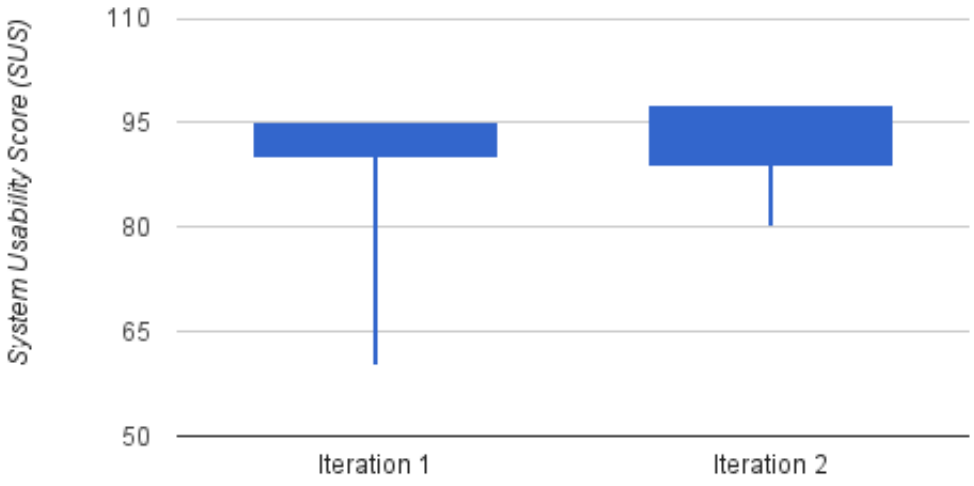

Figure 6.4: SUS scores in iterations 1 and 2.

6.3 Usage information

6.3.1 Logging

The usage statistics have been collected in the Web application component. The advantage of using client side logging over server side logging is that it can provide a much richer data about how user interacts with a search interface [15]. For example, it would be impossible to observe how long it takes for a user to drag a search result to a folder using server side logging. However, it can be easily done in the client side using JavaScript.

The current implementation uses a stack of logs. When a function that contains logging code is invoked, it pushes a new event to the stack. An event contains three fields: time, action, and meta information. The action field is a string representation of the action. The meta information field may contain some additional about the event. The stack may hold up to 7 events. As soon as the the stack is full, it pushes the events to the API component. In order to avoid losing events when the stack is partially filled and the user closes the application, the stack is automatically cleared every few seconds.

While the current implementation is great during application development and testing stage, it does not provide any security and a malicious user would be able to push fake logs to the server. Furthermore, the current logging service would lose the logs if the API is unable to handle the request. Therefore, the security and reliability aspects of the logging service would need to be enhanced in the future.

6.3.2 Usage statistics

Usage data was gathered during the application testing stage that lasted 3 weeks. In total, 20 people created an account on Dragondrop application, created 37 folders, and bookmarked 121 pages. 3611 search queries were issued: 3310 bookmark searches, 243 web searches, 58 news searches. The average length of a web and news search queries was 8 characters. The length of bookmark search queries was not be tracked because bookmark search requests are submitted with every key press.

Users	Folders	Bookmarks
20	37	121

Bookmark searches	Web searches	News searches
3310	243	58

The bookmarking interactions have been recorded and analysed. In total, 145 ”drag” actions and 98 ”drop” actions have been logged. This implies that there are 1.5 drag actions for each drop action. Furthermore, it was observed that it takes between 1.1 to 2 seconds to drag and drop a bookmark to a folder. This information is illustrated in Figure 6.5 below.

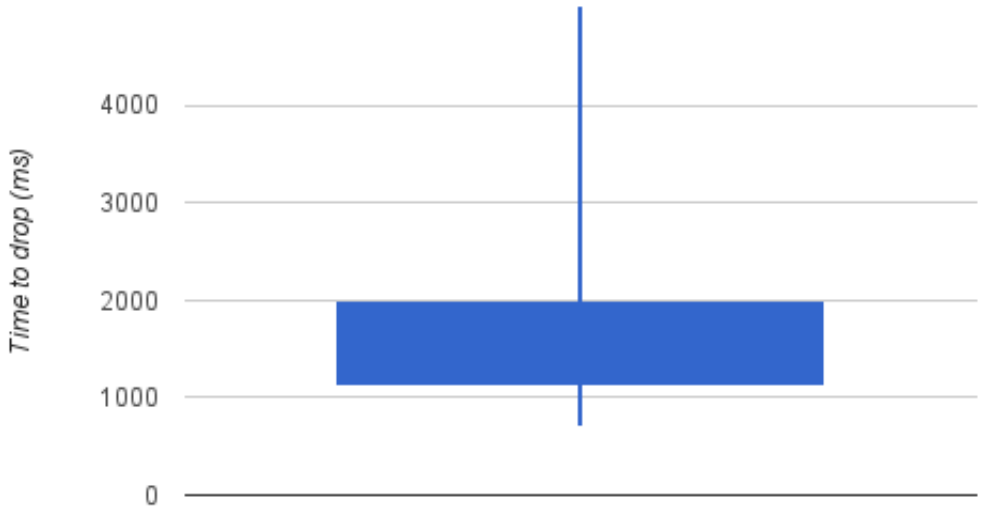

Figure 6.5: Time taken to drag and drop a search result.

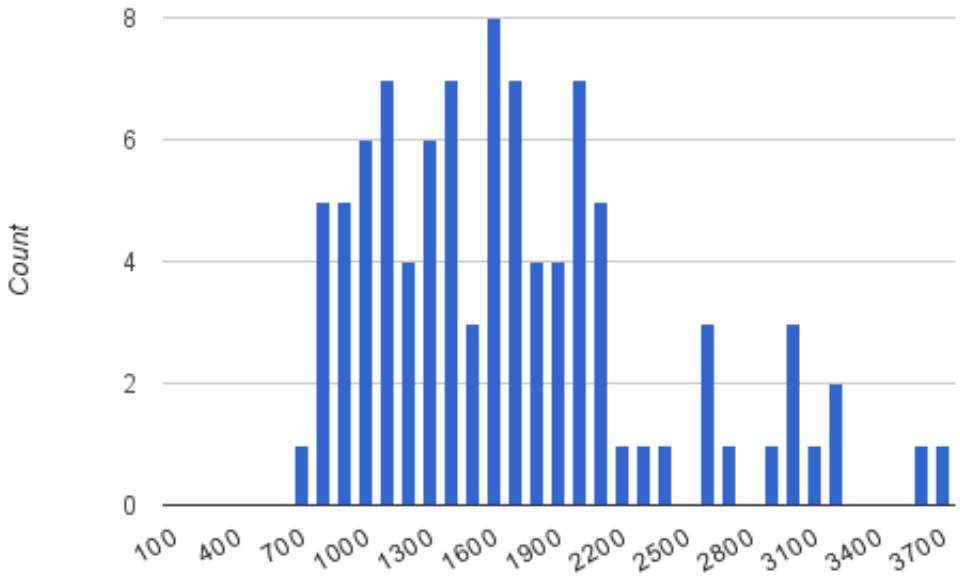

Figure 6.6: The distribution of time to drop values.

6.4 Future work

The expert reviews, usability study, and usage data provided a lot of insight into current usability problems in Dragondrop web application. This sections summarises the changes that will be introduced in addition to the scheduled work discussed in Iteration 2 section.

6.4.1 Result representation

Currently, bookmark search results appear exactly the same as web search results. The only visual difference is that bookmark search results have two additional links next to the URL field. The first link points to the user profile of the bookmark owner, while the second link points to the folder it belongs to. This caused some confusion between the test participants as they were struggling to distinguish the bookmark results from the web results. Therefore, the account handle and the folder name will be moved above the title of the bookmark search result.

6.4.2 Result separation

During expert reviews it was pointed out that all bookmark search results are listed in the same block. At the moment, all personal bookmarks are listed first and there is no separation between personal and public bookmarks. Therefore, public bookmarks will be displayed under a heading labelled ”Bookmark results from other people”.

6.4.3 Secondary notation

According to Cognitive Dimensions Framework, secondary notation can make information comprehension easier [18]. By allowing a folder to have a simple customisable structure such as categories, users would be able to group the bookmarks based on their personal needs. For example, a ”Python” folder could be divided into 4 bookmark categories: ”Tutorials”, ”Books”, ”Django”, and ”Scripts”. This change would give users more control over arranging their bookmarks.

6.4.4 Hidden dependencies

Dragondrop web application has number of hidden dependencies. It means that important links between entities in the system are not visible [18]. For example, it is not obvious to the user that when a folder is renamed and there is another folder with exact same name, they will be automatically merged. It is important to note that the actual problem is the fact that the user is not made aware of what is going to happen. Therefore, an information message will be displayed in all parts of the system that have hidden dependencies.

6.4.5 Visibility of system status

According to J. Nielsen, ”The system should always keep users informed about what is going on, through appropriate feedback within reasonable time” [21]. Unfortunately, Dragondrop application has no success or failure indicators for certain actions in the system. For example, when a user attempts to change his account handle in the preferences dialog, the system does not notify him whether the change was successful or not. This a major flaw in the current design and will be addressed with high priority.

6.4.6 Bookmarking process

The usage statistics demonstrated that 1.5 ”drag” actions are required to save 1 bookmark. Although further and more detailed analysis is required, this information suggests a high failure rate of the bookmarking action. Therefore, the bookmarks will be moved closer towards the folder sidebar. Also, the size of folder icons will be slightly increased.

Chapter 7

Conclusion

7.1 Summary

The objective of this project was to develop a search interface that supports bookmarking functionality and offers good user experience. The final version of the product meets all project requirements set out in Chapter 2 of this report. The usability study demonstrated that the application is highly usable and provides a good user experience. Although a lot of further work is required, Dragondrop could be turned into a real product that has potential to introduce a new phase in the history of bookmarking.

7.2 Problems faced and lessons learned

7.2.1 Testing

Testing is a crucial part of a software development life cycle. However, it is often forgotten that software testing should begin as early as possible. Unfortunately, Dragondrop application testing started at a very late stage in the life cycle. Consequently, it was very difficult to derive edge cases and write tests for features that were no longer fresh on the mind.

7.2.2 Tangled and scattered code

The separation of concerns is one of the main principles in software engineering. It implies that the code should be organized in a way that each element in the system is responsible for doing only one thing [25]. Although Dragondrop application is well modularized, it still has 4 cross-cutting concerns.

In Dragondrop application all business functions are tangled with support functions. For example, folder create function must authorize the user, validate the request, validate request data, execute the request, and construct a response. The same procedure is repeated for all API calls. Every time a major change is introduced to Dragondrop application, all API functions must be rewritten. This often leads to inconsistencies and defects in the system.

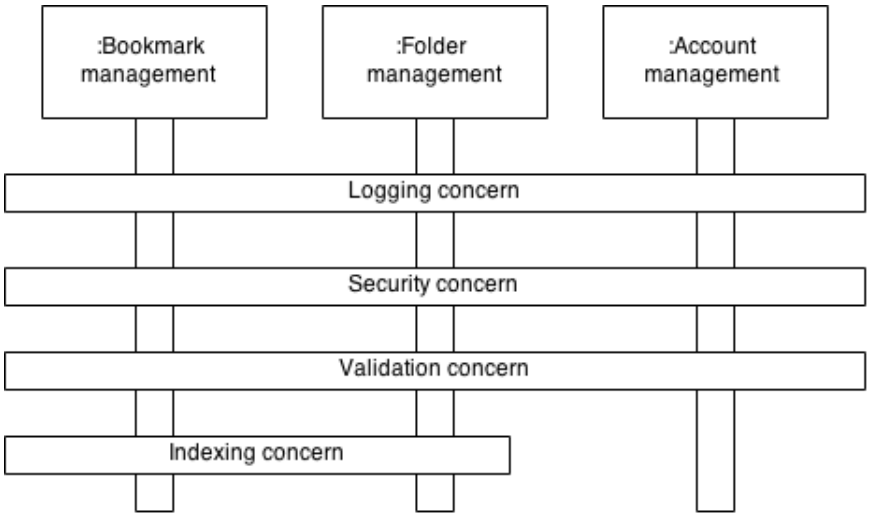

Figure 7.1: Cross-cutting concerns in Dragondrop application.

The way to deal with the tangled and scattered code is to introduce an aspect oriented programming framework to the application. It would be responsible managing different aspects of the system: security, logging, validation, and indexing. The business functions will no longer require to include common wrapper code that distracts developers from the main objective.

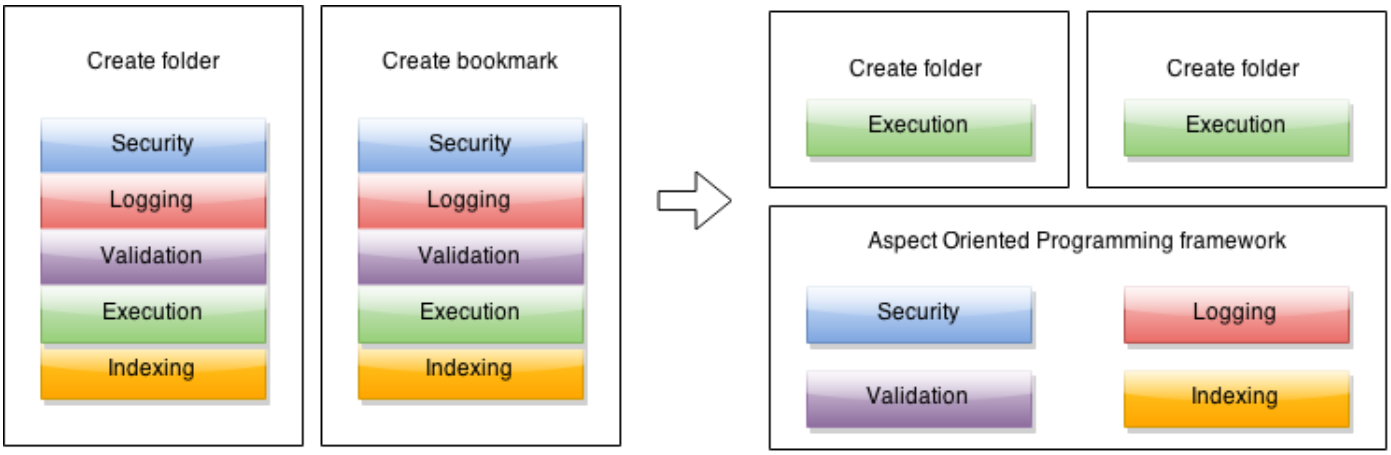

Figure 7.2: Using Aspect Oriented Programming to increase the maintainability and reusability of the code.

7.3 Future work

Dragondrop is an exciting new project with many future development opportunities. This section discusses what further steps should be taken to improve the software development life cycle, system architecture, and user experience of Dragondrop application.

7.3.1 Continuous integration

Continuous integration is a common software development technique that allows developers to integrate their work frequently and discover any integration problems as early as possible [17]. Online services such as Travis

CI offer continuous integration for GitHub projects [11]. It works by observing the changes made to the project. Every time a GitHub pull request is received, Travis builds the project using the code for the new feature. If any of the project's tests fail, the pull request cannot be merged into the master branch. Although the use of continuous integration techniques does not guarantee that new features will not introduce defects, it certainly reduces the risk. Furthermore, this change in the software development life cycle reinforces other good software engineering practices such as test driven development. For example, all new features must be committed along with unit tests. Therefore, Dragondrop application should be configured for Travis continuous integration platform in the near future.

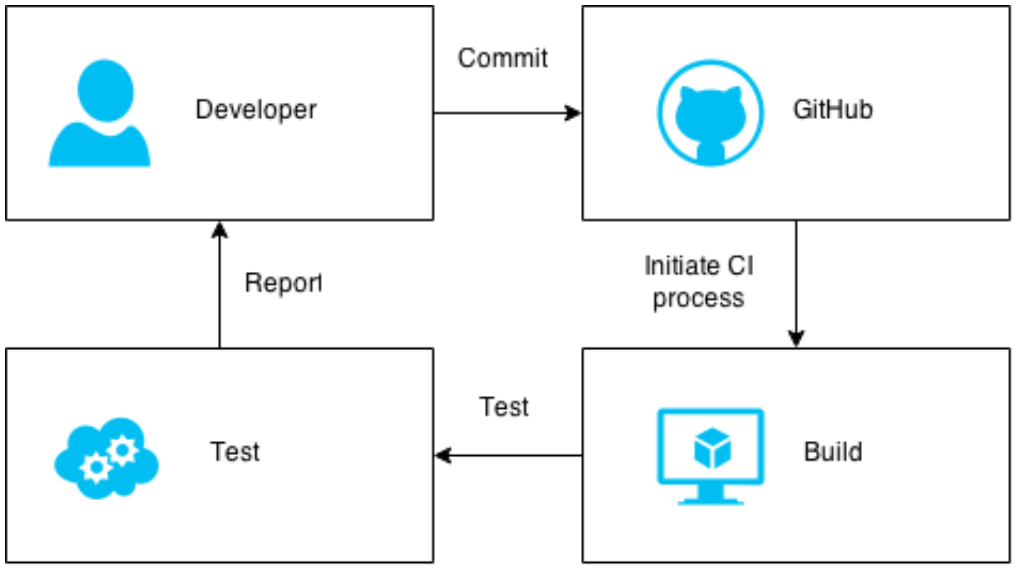

Figure 7.3: Continuous integration process.

7.3.2 RESTful API

The API component in Dragondrop project needs to be improved. One of the weakest parts in current implementation is that the data is stored in the session and that makes the whole project stateful. Stateful sessions are often a performance bottleneck for websites with heavy traffic. Therefore, Dragondrop should lean towards more scalable solutions. The API should be reimplemented in a stateless manner (following REST paradigm) so that each client's request would contain all information required by the server. The benefits of this change would be tremendous. First of all, the API would become much more scalable. Secondly, the development of various widgets that consume the API would be facilitated. Finally, the Web application component and the API component would become completely uncoupled. In other words, the website would become just another application that makes use of Dragondrop's public API.

7.3.3 Dragondrop in your browser

Dragondrop application only supports bookmarking inside the search interface. This means that the user is unable to bookmark any pages unless he is logged in to his account on Dragondrop website. One of the ways overcome this limitation is to implement a browser extension that allows the user to bookmark the page on any website with a single click of a button. A good example is Pocket extension for Google Chrome - an application that allows users to save articles they would like to read later. Figure 4.9 demonstrates how a user can save a page to Pocket with a single click of a button. An equivalent browser extension for Dragondrop project would simplify the bookmarking process.

Another useful addition would be to offer Dragondrop users to change their default browser's search engine to Dragondrop. All queries typed in the URL box would be redirected the user to Dragondrop instead of Google. This would greatly increase the search volume and increase the engagement with the application.

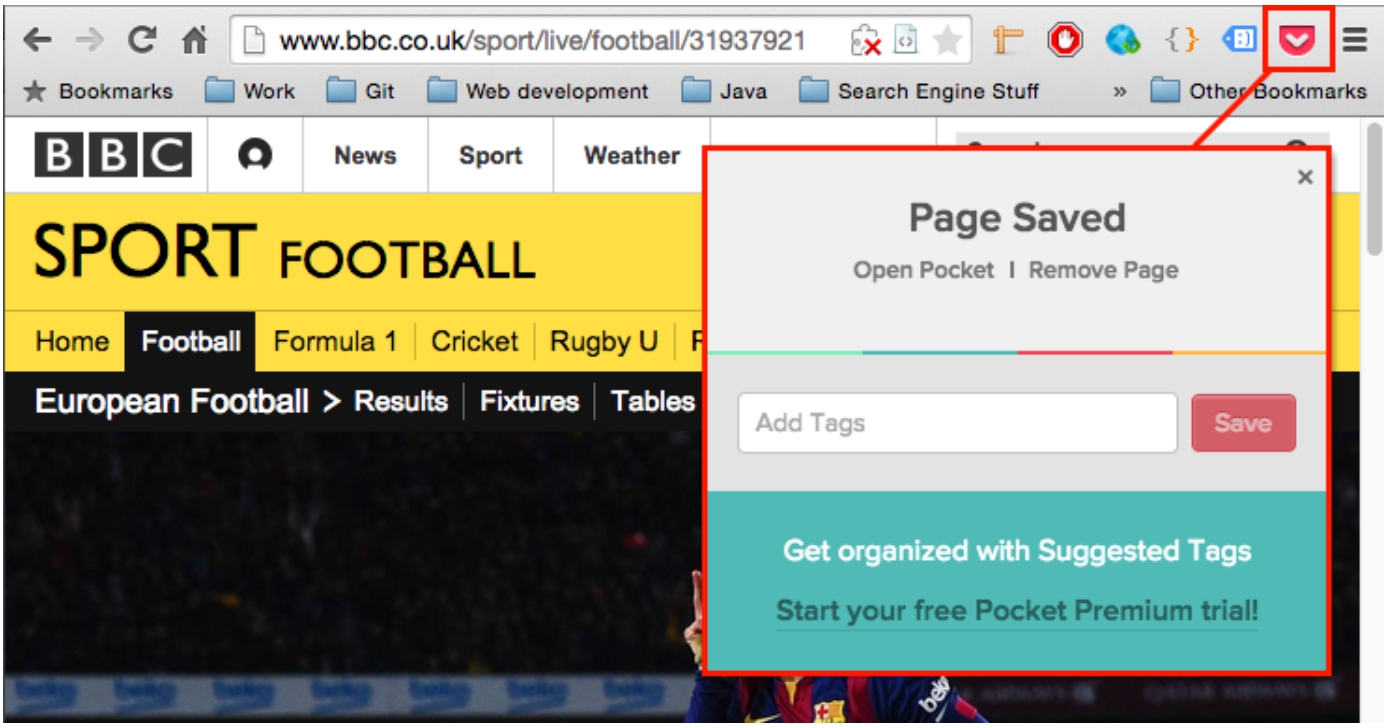

Figure 7.4: Pocket - a simple browser extension that allows users to save articles they would like to read later.

7.3.4 Import existing bookmarks

One of the most common mistakes in terms of product usage experience that online product developers make is to greet newly registered users with an empty screen. In other words, when a user signs up on the website he should be presented with various opportunities to engage with the system. For example, instead of saying "You have no bookmarks.", the user could be presented with an option to import existing bookmarks from his browser. This simple feature would greatly contribute towards a better user experience and allow the user to get started with using the application more quickly.

7.3.5 Bookmarked page content indexing

The Search component in Dragondrop application indexes each bookmark's title, description, url, and the name of the folder that it belongs to. The search functionality works great when a user searches for a specific website. For example, the search query must include either the name of the website or keywords that appear in the bookmark's title or description. Unfortunately, the search will fail if a user is unable to recall the specific bookmark that contains the required content. Therefore, the content of the bookmarked pages must be crawled and indexed as well. This change would present the users with results that better represent their information need.

7.4 Acknowledgements

- Dr. Leif Azzopardi for his invaluable guidance and passionate involvement in the project.

- All participants in the usability study for their time and honest feedback

Appendices

Appendix A

ElasticSearch mappings

```
{
"settings": {
      "analysis": {
         "filter": {
            "nGram_filter": {
               "type": "nGram",
               "min_gram": 2,
               "max_gram": 20,
               "token_chars": [
                  "letter",
                  "digit",
                  "punctuation",
                  "symbol"
               ]
            }
         },
         "analyzer": {
            "nGram_analyzer": {
               "type": "custom",
               "tokenizer": "whitespace",
               "filter": [
                  "lowercase",
                  "asciifolding",
                  "nGram_filter"
               ]
            },
            "whitespace_analyzer": {
               "type": "custom",
               "tokenizer": "whitespace",
               "filter": [
                  "lowercase",
                  "asciifolding"
               ]
            }
         }
      }
```

```
   },
"mappings" : {
"bookmark" : {
          "_all": {
             "index_analyzer": "nGram_analyzer",
             "search_analyzer": "whitespace_analyzer"
          },
        "properties" : {
         "folder" : {"type" : "string"},
         "link" : {"type" : "string"},
         "title" : {"type" : "string", "boost" : 4},
         "description" : {"type" : "string", "boost" : 2},
            "user": {"type": "integer", "index": "not_analyzed"},
            "visible": {"type": "boolean", "index": "not_analyzed"}
        }
     }
}
}
```

Bibliography

[1] Data binding in angular js. https://docs.angularjs.org/guide/databinding/.

[2] Delicious social bookmarking. https://delicious.com/martynasb.

[3] Digg social bookmarking. http://digg.com/.

[4] Html living standard. https://html.spec.whatwg.org/multipage/forms.html#implicit-submission.

[5] The javascript object notation (json) data interchange format. https://tools.ietf.org/html/rfc7159.

[6] The oauth 2.0 authorization framework. http://tools.ietf.org/html/rfc6749.

[7] Protractor end to end testing for angular js. http://angular.github.io/protractor/.

[8] Redis queue: Simple job queues for python. http://python-rq.org/.

[9] Sauce labs. http://saucelabs.com/.

[10] Selenium browser automation tools. http://www.seleniumhq.org/.

[11] Travis continuous integration. https://travis-ci.com/.

[12] Yahoo bookmarks. https://uk.toolbar.yahoo.com/bookmarks.

[13] Yahoo user interface library. http://yuilibrary.com/.

[14] A. Shtub A. Parush, R. Nadir. Evaluating the layout of graphical user interface screens: Validation of a numerical computerized model. In *International Journal of Human-Computer Interaction*, pages 343–360, 1998.

[15] Leif Azzopardi, Myles Doolan, and Richard Glassey. Alf: A client side logger and server for capturing user interactions in web applications. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '12, pages 1003–1003, New York, NY, USA, 2012. ACM.

[16] Joohn Brooke. Sus - a quick and dirty usability scale. http://hell.meiert.org/core/pdf/sus.pdf.

[17] Martin Fowler. Continuous integration. http://martinfowler.com/articles/continuousIntegration.html, 2006.

[18] Thomas Green. Cognitive dimensions of notations. http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/, 1998.

[19] Marti A. Hearst. *Search User Interfaces*. Cambridge University Press, 1st edition, 2009.

[20] Ruth Malan and Dana Bredemeyer. Defining non-functional requirements. http://www.bredemeyer.com/pdf_files/NonFunctReq.PDF, 1995.

[21] Jakob Nielsen. 10 usability heuristics for user interface design. http://www.nngroup.com/articles/ten-usability-heuristics/, 1995.

[22] Jakob Nielsen. Why you only need to test with 5 users. http://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/, 2000.

[23] Jakob Nielsen and Thomas K. Landauer. A mathematical model of the finding of usability problems. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 206–213, New York, NY, USA, 1993. ACM.

[24] Jeff Sauro. *A Practical Guide to Measuring Usability*. Measuring Usability LLC, 2010.

[25] Ian Sommerville. *Software Engineering*. London : Pearson, 9th edition, 2011.

[26] Ryan Sonnek. Realtime search: Solr vs elasticsearch. http://blog.socialcast.com/realtime-search-solr-vs-elasticsearch/, 2011.