Making Test Case Images Searchable

BTECH 451 Final Report

Tianyun Gu
Supervisor: Dr Xinfeng Ye

Abstract

This report covers my work over the course of the year on the BTECH 451 project with Kiwiplan, which aims to produce a system to increase the ease of locating images for test cases used in the company's automated testing system. The motivation for the project is described in the first section along with an overview of the scope and goals. Functionalities required for the system are discussed, including optical character recognition for extracting text from images, and indexing images for search based on these results. The search interface is implemented as a web service to allow remote access across the network. Different options for fulfilling these requirements are considered with respect to challenges faced. The system's current implementation is described and I conclude with suggestions for possible extensions to the system in the future.

Contents

1 Introduction

This section provides context and introduces the aims for this BTECH 451 project.

1.1 Project Overview

Kiwiplan is an industry software solutions development company, which produces software for use in many areas of the supply chain in manufacturing of rigid packaging. Automated testing of their software is performed by an in-house developed system specifically for their products, named Droid. Tests are performed by checking for discrepancies between expected results in the form of a screen capture image and the screen produced from running the test case. While this approach to automated testing may not be considered the best solution today, Kiwiplan was an early adopter of an automated approach to testing, such that off the shelf solutions were not readily available at the time of Droid's development. Justification for the continued use of Droid comes from the system being built to meet the specific needs of the company, and regardless of whether other approaches may be adopted in the future, there is benefit in continuing the support of legacy applications.

One challenge currently posed by Droid is the difficulty in matching the output produced by tests results to the test cases which produced them, due to there being no quickly recognisable connecting indicators. Manually locating individual test case images from the large (and continuously growing) number of entries present in directory structures used by Droid is tedious, so in order to facilitate continued growth, a solution to perform automated search of test cases is proposed. This project aims to develop an application to address such needs.

1.2 Solution Concept

In order to address the challenges described above, the proposed solution is to develop a system which can be used to search for and retrieve base case images through the text strings which appear on the image. A user accesses a search interface in the form of a web based service and search results are to be returned in a reasonably quick time frame. A number of tasks must be performed by the application as part of this:

1) **Convert image file type:** All Kiwiplan's test case images are encoded in a proprietary file format, CBMP, which provides very efficient image compression. While this format can be used by Droid, it must be converted to a commonly useable format for processing.
2) **Extract text from image:** Text that is present on an image should be extracted and indexed for search. This is performed through optical character recognition (OCR).
3) **Images indexed with search terms:** After extracting text from images, this information needs to be stored in a method that will allow for search in an efficient manner.
4) **Communication through web based service:** Other machines on the network should be able to access the search interface to locate images stored on testing machines.

The most significant components of the project fall under tasks 2) to 4). The majority of work during the first half of the year has been focused on the second and third tasks, and the web service has been the focus of the second half. Because the project is primarily interested in achieving a functioning system at this stage, I focus on examining the use of third party packages to achieve these goals.

1.3 Related Work

To help guide implementation decisions for my projects, other systems comparable in
A similar pipeline to that identified in the goals for this project has been used to make collections of text fully searchable, such as for online libraries or document archives. I considered two examples to examine the tools utilised in achieving this, the first from the World Digital Library [1] and the second from the New Zealand Journal of History at the University of Auckland [2].

The World Digital Library created a system to facilitate search of text in digitalised books from scanned pages using the Tesseract OCR engine and Apache Solr for indexing and search. It was noted that inaccuracies in the OCR results were acceptable as the original page scans were presented to the user, rather than the OCR text itself. OCR was only used for the purpose of indexing pages, and while inaccuracies do lead to a decreased hit rate for search results, the amount of existing content which would otherwise be completely unsearchable means that the benefits outweigh the loses. In comparison to the scope of my project, incorrect OCR results may be far less desirable.

The New Zealand Journal of History is an academic journal which began publication in 1967 and offers online access to issues. Issues older than 2003 have been made available through scans of printed copies, with the content extracted through OCR. The engine utilised is ABBYY Finereader, with a reported accuracy rate of 99.5%. It is difficult to determine the significance of this value without other comparisons.

2 Optical Character Recognition

Optical character recognition, or OCR, is the process through which images of text are converted into a machine readable format, for the purposes of digitally editing or searching, for example. Images can be sourced from a variety of methods including photographs or document scans and for the purposes of this project, a capture of what would be screen display. Common applications of OCR include transcribing printed documents to digital form, data entry from records in physical form, and conversion of handwritten text.

2.1 OCR Engines

2.1.1 Evernote

Evernote is an application used for digitally creating and storing ‘notes’ with support for a range of media types, including both text and images. OCR is included in the functionalities, as part of extending the ability to perform search on content to all forms of media which may

be contained on a note. As part of the initial project concept discussion, demonstrations of using a Droid test image with Evernote without any image pre-processing appeared to be surprising successful. Further research into how Evernote performs OCR reveals a fairly complex process. The 'Evernote Indexing System' is responsible for making all content of a note searchable and is invoked each time a user upload one containing some data matching the MIME type of an image. Images then go through a series of 'passes' of OCR, each with different assumptions about how text maybe appear on the image, and each set of assumptions gives rise to different techniques used during pre-processing the image [2].

Where many traditional applications of OCR place high value on the accuracy of results, the end goal from this is to extract as many text strings as possible from the content of an image such that indexing and search will provide comprehensive cover. The end result from this process is a set of possible alternatives for each word identified on the image, each associated with a weight value, w, representing the confidence in its correctness. This information is embedded in the metadata of an image-containing note such that when a user performs a search for a phrase, each alternative listed for all text identified on an image is checked for a match [3]. In this way, Evernote's approach to OCR introduces a high level of fault tolerance as it does not rely on the decisive results from a single pass of character recognition.

Evernote's API is publically available however the user license agreement forbids the exclusive use of the OCR services. For the purposes of this project, building a system of this complexity is unrealistic due to time and resource constraints. One factor which may be of interest to consider if time permits in the latter part of the project is the use of a probabilistic approach when indexing images, with differing techniques used during pre-processing to produce alternative character recognitions.

2.1.2 ABBYY Finereader

ABBYY Finereader software is geared towards converting scanned images into an editable form and includes capabilities to recognise and maintain document formatting during the conversion. While this may be useful if we are interested in keeping the integrity of the document content, such as in [2], this extra information is not useful for the purposes of this project, where we are only interested in using OCR for indexing. Furthermore, because the images to be worked with are not scanned documents, I cannot be certain that this would produce the same level of accuracy as described earlier. It is likely that such an OCR solution which is aimed to be used with scanned documents will not produce the best results when applied to screen images, as will be illustrated in section 2.2. Another drawback of this system is the limited technical documentation available, due to the software falling under a proprietary license, so it overall its usefulness in this project seems limited.

2.1.3 Tesseract OCR

The Tesseract OCR engine was original developed by Hewlett Packard and is currently open source. One key feature in the way Tesseract performs OCR is the use of an 'adaptive classifier', which actively learns from the content as it is processed, such as estimating character widths, to improve accuracy of information which is later processed. Character recognition is performed in two steps. The first pass sends successfully classified characters

to the adaptive classifier as training data and the second pass will attempt to reclassify words which were uncertain on the first pass [6].

Another feature of Tesseract OCR is its ability to detect and adapt to the baseline of text. Because Tesseract is purely an OCR engine and does not perform any pre-processing on images before character recognition, this capability may be useful if text on images do not follow a perfect vertical or horizontal alignment. This feature is not very valuable for the purposes of this project however, as input images will not suffer from such problems. The lack of pre-processing performed by Tesseract OCR engine does provide one benefit in that there is greater control over how images are handled before character recognition is performed, which proves to be useful due to the unconventional nature of the images used. This is elaborated on in the next section.

One drawback of Tesseract OCR is the use of polygon approximations for characters identified in an image. The typeface of text in Droid images is a uniform thickness of one pixel, with serifs, as shown in Fig. 1. Because even slight alterations to characters in this style may have a very large impact on the character apparently appearing on the image, any inaccuracies introduced by approximations of the shape of characters may compromise the accuracy of OCR results. All things considered, the flexibility provided by this engine seems the most appropriate for use with this project.

2.2 Challenges

Several factors influence the accuracy of OCR results, aside from the algorithms used. These come from qualities of the image content, more specifically, the integrity of the text which OCR intends to capture. In common applications of OCR, such as with scanned documents or photographs of text, factors which may reduce the quality of OCR results include the presence of noise or artifacts in the image, distortions to the text [5]. To achieve better results, pre-processing and image before OCR aims to correct these issues.

Because images used by Droid are originally digital, many of these factors will not be present. However, other challenges are encountered due to the nature of such images. Most OCR packages are not created to be used with images generated from on screen display. For example, a minimum resolution of 300dpi is recommended for images used in OCR, compared to the standard 72dpi used on computer monitors. This means that although images used by Droid contained perfectly clean text, performing OCR on these images without any pre-processing will still produce very poor results.

Fig. 1 is an example of an image used by Droid before performing any pre-processing, and Fig. 2 shows the output given by Tesseract OCR from this image. Despite the text contained in this image being very easily readable for a human, OCR results are extremely inaccurate. Other issues demonstrated here include:

- Presence of undesirable words which appear on the interface.
- False positives caused by interface elements.
- Image border interfering with character recognition (“Num” interpreted as “um” due to the left border).

Evidently, there is a strong need for pre-processing to improve accuracy to a usable level.

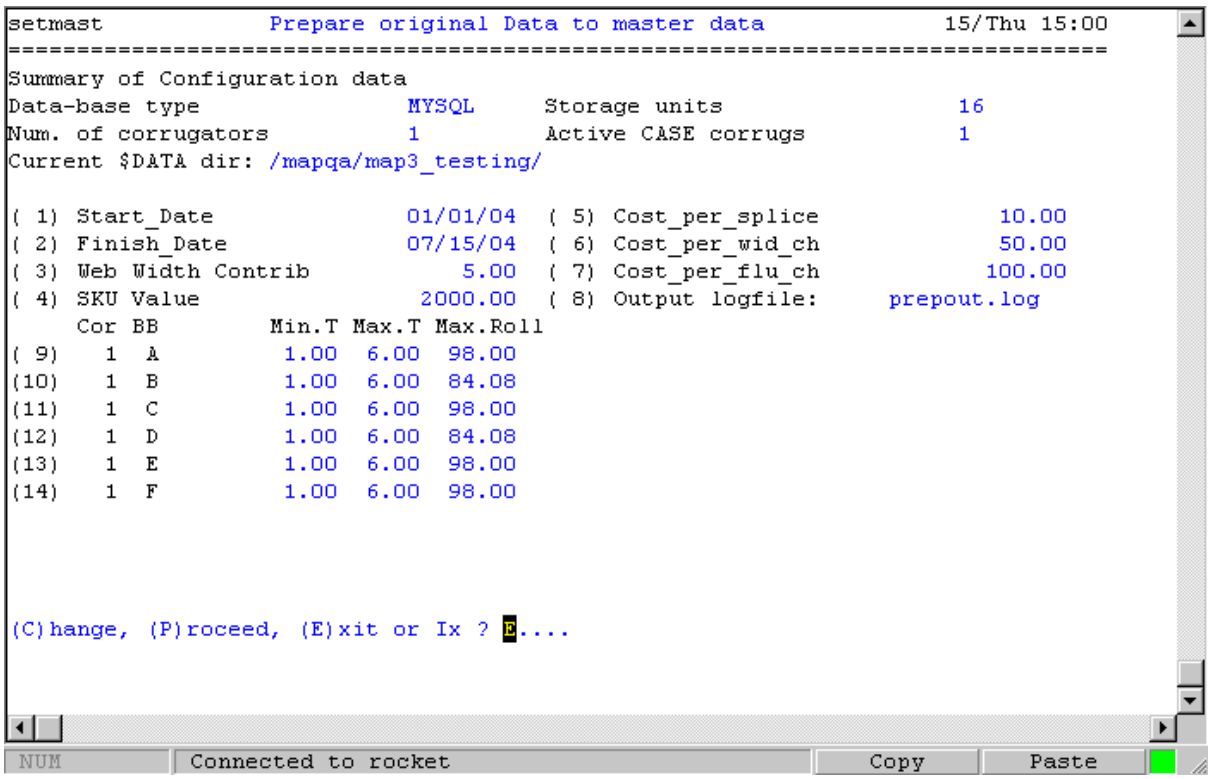

Figure 1. Example image before any pre-processing.

```
10
11  secmas: PrEpare urlglna) Data to master data JS/Thu 15 an
12  Summary of Configuraclun data
13  Dacarhase type HYSQL Storage unlcs 1
14  um. of corrugacurs 1 Acclve case corrugs 1
15  Current snara a1 /mapfia/map37cesc)ng
16  L 1L Scarcibace u1/u1/u4 L 5L Cusciperispllce 1u.uu
17  L 2L Flnlshibace u7/15/u4 L 5L Cusciperiwldich 5n.uu
18  L 3L "eh "ldch Concrlh 5.uu L 7L Cusciperiiluich 1uu.uu
19  L 4L sxu Va1ua zuuu.uu L 5L Output Juginle. paapaaa.1ag
20  CD: as u1m.r Hax.T Hax.Rull
21  L 9L 1 a 1.uu s.uu 95.nu
22  L1uL 1 a 1.uu s.uu 54.ua
23  L11L 1 c 1.uu s.uu 95.nu
24  L12L 1 n 1.uu s.uu 54.ua
25  L13L 1 a 1.uu s.uu 95.nu
26  L14L 1 r 1.uu s.uu 95.nu
27  (CyhangEy LPLIDceed, LELx): or IX 7 a
28  4 »
29
30  17mm y—cammaaaaa ma tucker. Copy Paste l 1
31
```

Figure 2. OCR Result from Fig. 1.

2.3 Pre-processing

The most obvious first step to address the issues introduced above is to alter the resolution of the image, and the simplest solution to this is scaling up. Through experimentation I arrived at enlarging the image by around 400% to be the limit of accuracy gains from simple scaling. Although this now creates blurriness in the image, this alone significantly increases accuracy over the original image, shown in Fig. 3 (complete illustration of OCR results from this is given in Appendix A). Cropping the image to remove the borders and interface also solves a number of the previously mentioned issues.

This result is not perfect. In particular, certain characters appear to be consistently interpreted incorrectly, such as 'M' and bracket characters. Another noteworthy observation from this output is the differences in text formatting in comparison to Fig. 2. However, because white space will not be considered when indexing images, this is not significant to consider and issue.

Beyond these there are no obvious indicators for how improvements to the OCR result could be achieved through editing the image during pre-processing, and it maybe be a matter of conducting continued experimentation to achieve the most accurate results. This level of accuracy is deemed acceptable in the interest of advancing project progress. Future refinements can be made to the pre-processing performed on images to achieve higher accuracy.

```
37
38  Start_Date 01X01X04
39  Finish_Date 07X15X04
40  Heb Uidth Contrib 5.00
41  SKU Value 2000.00
42  Cor BB Hin.T Hax.T Hax.Roll
43
44  1 A 1.00 6.00 98.00
45
46  1 B 1.00 6.00 84.08
47
48  1 C 1.00 6.00 98.00
49
50  1 D 1.00 6.00 84.08
51
52  1 E 1.00 6.00 98.00
53
54  1 F 1.00 6.00 98.00
55
56  (Clhange, (PJroceed, (Eint or Ix ? I....
57
58  Cost_per_splice
59  Cost_per_wid_ch
60  Cost_per_flu_ch
61  Output logfile:
62
```

Figure 3. A sample of OCR output from image scaled up to 400%.

3 Indexing and Search

This phase of my project investigates the storing of OCR results for search. An important consideration is compatibility with Kiwiplan's existing systems, which utilise MySQL databases. Although many indexing tools offer their own databasing options, this is not considered ideal due to the need to keep systems consistent for in-house applications, for ease of maintenance. The tools used for indexing and search are often combined, so

3.1 Xapian

The first option considered was Xapian, an open source search engine library which supports parallel communication with multiple databases. The most attractive feature of Xapian is the capability for probabilistic search [7], which would be useful for implementing a system with similarities to Evernote's indexing of images. Xapian performs indexing by defining a set of documents, $\{D_1, D_2, \dots, D_i\}$, and index terms, $\{t_1, t_2, \dots, t_j\}$ [8]. Each index term has an associated list of documents indexed by it so that when a search is performed using a specific term, the results are the entries in this list. This is an efficient method of storing index mappings so queries can be processed in a fairly quick amount of time.

One drawback of Xapian is the lack of a Java interface. Although Java bindings exist, the use of JNI removes platform independency. Furthermore, because there are no dependencies available online, to use Xapian would require the entire library be included with the application and potentially rebuilt where ever it is to be run. This, along with the lack of complete documentation for its use, may cause future issues if used.

3.2 Hibernate

A pure Java implementation provides the most convenience for both building the system and future maintenance. Hibernate Search, build on Hibernate ORM, allows Java objects to be indexed for search using object properties as index terms, as defined by the developer through either annotations throughout the code or XML configuration files. Configuration informs Hibernate of how objects should be mapped to the database, as well as providing database connection information. These two configurations are separated into Hibernate configuration and object configuration.

To facilitate search, Java objects are first converted to database entries through ORM. One advantage of this is that there is direct control over the structure of the relations used to store indexes, as the method for creating mappings are defined by the Java application, along with which object properties are indexed.

3.2.1 ORM

Working with a system which uses data represented in both an object orientated form and relational data may be problematic, as there may be no exact mapping of object information into relations. This is further confounded by the need to ensure changes made to the data in either of these forms must maintain consistency. Object-Relation Mapping, or ORM, is the conversion of data objects in an object-orientated paradigm to relations in the relational model for data. This technique is often used to increase the ease of working with such cross-paradigm system. ORM will add additional computation overheads to the system, however

this a trade-off for removing the need to consider additional data structures when sending and retrieving information from the system and database.

Even with the use of ORM, some consideration must be given to the structure of Java objects used to ensure mappings are performed correctly. The use of Hibernate ORM will often denormalise data, so the intended structure of relations must also be considered to reduce redundancy in the resulting relations. Because the objects used by the system are few and simple; images and index terms with a single many-to-many relationship between them, this did not pose too many problems. The data structures implemented by this system are discussed in further detail in section 4.

Hibernate categorizes objects into two types when converting to a relational model, entity and attribute value types [10]. Entity typed objects correspond to rows on a table, while value types are owned by entities. Value typed objects are further divided into three categories; basic (single valued data types), embeddable (a more complex data structure, such that it has a number values of its own) and collection. These categories are used by Hibernate in determining how conversion to a relational data structure should be performed on Java objects.

3.2.2 Hibernate Search

Hibernate Search extends the core functions of Hibernate ORM by enabling full text search on objects stored through Hibernate. Searchable object attributes are specified through code annotations, and search can be enabled on both primitive fields or nested objects. Hibernate Search operates using Lucene indexes, and indexing is performed automatically each time an object is processed through ORM, whether through persisting, updating or removal from the database. Where a database of existing entries is used and objects have not been previously processed through ORM, a Lucene index must be created for the existing data in order to enable it for search.

One advantage of using Hibernate search to make database queries from the Java application, rather than querying the database directly, is the removal of dependency on vendor specific SQL dialects. Queries made from the application are created as a Hibernate Query, and transparently translated to the configured SQL dialect in the background. This means that migrating from one database implementation to another will not require changes to be made to the Java code.

4 System Architecture Overview

Fig 4. illustrates the implemented system architecture and interactions between components.

4.1 Image Pre-processing

When the system receives an image to be indexed to the database, the first step is to decompress the CBMP file. Because this file type was developed by Kiwiplan to be used with Droid, there is no common method for decompression and a conversion program is called by the system as a new process to perform the conversion. This image is then read by the system and passed on to ImageMagick [9] to continue pre-processing, as described in section 2.3. The image is scaled up and cropped to remove undesired edge pixels.

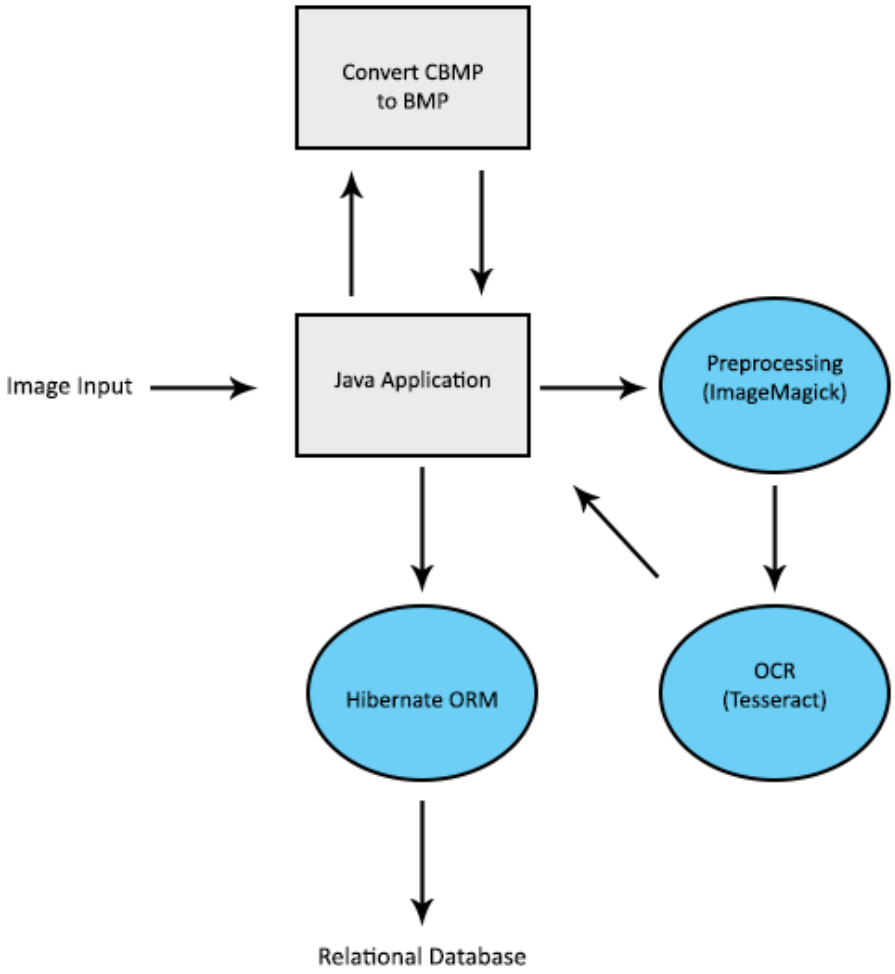

Figure 4. Image processing architecture.

4.2 Data Representation

The Java application defines two classes for representation of image data and associated index terms, which are mapped to three tables in the database. Each image stores information about its file name and file path, specifying its location on the test machine, along with an associated set of index terms containing the output produced by Tesseract OCR. Every unique index term is itself represented as an object, which is necessary for the many-to-many relationship which exists between terms and images.

Hibernate ORM converts these objects into relational form to be stored in an SQL database, in accordance with Hibernate's configuration. Configuration is provided in the form of XML documents, which is the standard practise at Kiwiplan. The Hibernate configuration for TestImage is given in Appendix B to illustrate an example.

4.2.1 TestImage Class

A TestImage object is created for every image processed through the system and stored in the database. An instance of TestImage stores the following information:

- **imageId:** A variable of type int, used as the primary key for an entry in the database. Although providing no significant information about the image, this field is required by Hibernate ORM. imageId numbers are generated incrementally by Hibernate as images are persisted, and are not set directly by the system.
- **imageName:** A variable of type String, containing the image file name.
- **imagePath:** A variable of type String, containing the file location of the image. This informs the user of where the image can be located on the test machine.
- **imageTags:** A Java HashSet of ImageTag instances, listing all index terms of an image, and generated from the result of performing OCR.

4.2.2 ImageTag Class

An ImageTag object is created for every unique term which appears on images, and stored in the database as their own table. One instance of ImageTag stores the following information:

- **tagId:** A variable of type int, denoting the primary key for the unique term. Used by Hibernate ORM for database storage.
- **tagText:** The String value of the term.
- **images:** A Java HashSet of TestImage instances, referencing each image that the term appears in. This creates a reverse index structure, and although this is unnecessary for the system logic, Hibernate uses this to complete the many-to-many relationship when data is translated for the relational database.

4.2.3 Database Design

The chosen database design represents the many-to-many relationship between test images and index terms as three tables. TestImage objects are mapped to the TESTIMAGE table and ImageTag objects are mapped to the IMAGETAG table. A third table, TAGMAP, stores a listing of every ImageTag that exists for each TestImage. Although the TAGMAP table may grow to contain a very large number of rows, this database design stores data in a normalised form and is the most space efficient, as redundancy is reduced and integer values are less costly to store compared to strings. The structure of these tables is provided below with example data.

TESTIMAGE

ImageID	ImageName	ImagePath
1	image1.png	/home/tianyun.gu/img1/image1.png
2	Image2.png	/home/tianyun.gu/img1/image2.png

IMAGETAG

TagID	TagText
5	num
6	sum
7	web

TAGMAP

ImageID(FK)	TagID(FK)
1	5
1	6
2	6
2	7

5 Web Service

For the database of images to be searchable from remote machines, a search interface must be developed for a web based service. There are a number of options for implementing this, whether through a pure Java implementation or utilising a cross platform approach.

5.1 Javascript

One method of implementing the web interface involves the combination of and HTML web pages for the user interface, utilising Javascript to retrieve and format raw object data from the service back end, deployed independently. An example of this may retrieve JSON data from a web service and dynamically update the content of an HTML page with some predefined function.

With this approach, the web service and user interface are kept separate, and processing of data received from the service is entirely the responsibility of the receiver. This means that the web service does not need to consider how the information will be displayed, and allows for a number of different interfaces to potentially interact with the service, each handling the data uniquely. However, this does add an additional layer of processing, where Java objects must be converted between formats. Because this web service will only see in house use, the benefits of potential reusability for other applications do not seem significant enough to justify such an approach.

Furthermore, because Kiwiplan has adopted Java as their primary language for use, in the interests of maintenance a Java focused approach is more desirable, as this will reduce the learning curve required for somebody who is introduced to the system implementation. The advantage of separating the user interface from application logic can still be achieved with a more integrated approach.

5.2 JSP

JavaServer Pages (JSP), is a Java Enterprise Edition technology which allows for the separation of business logic from the visual representation of a page. The layout and structure of a JSP's content may be defined through HTML or XML, while dynamic page content can be generated through embedded Java code, comparable to PHP and Microsoft's ASP [11]. This means that static portions of a web page can be written in a traditional method, while Java used to create dynamically generated content may be separately developed. Changes to the implementation of the Java backend do not require updates to be made to the JSP, as the two are kept

independent. Likewise, the static elements of a page, including the visual representation, can be altered without needing to make changes to the Java code.

When a browser requests a web page implemented as a JSP from a web server, the server recognizes this request by the extension of the URL and the request is passed to the JSP engine. The engine loads the JSP page from disc and performs translation to a servlet. The JSP engine then compiles the servlet into an executable class. The web server then loads and executes the servlet class. HTML is produced as output and given back to server, which is forwarded to the client. The client browser is able to handle this result as a usual, static HTML page.

JSP also replaces the need for a programmer to create servlets for a service, as JSPs are translated into servlets when the page is loaded. The life cycle of the servlet is then managed by the servlet container and servlets are only created as needed. Multiple servlets are not created every time a page is loaded, as the servlet container keeps track of the servlets already running and reuses them as required.

5.3 Spring MVC

Spring framework is used for inversion of control for Java applications. The framework consists of the core module, which provides a Spring container for the management of Spring beans, as well as a collection of various extensions to the core. Spring MVC is an extension of the Spring core intended for sure with web applications, to provide separation of the application into *model-view-controller* layers. A Spring MVC application handles processing at each layer and allows for the implementation of a web service purely through Java, with the entire service contained in one package, including web pages and system logic.

One advantage of this approach is that there is no need for the handling of data objects on the client-side, as all data processing functions are performed by the servlet, written with Java code. The Spring controller handles the mapping of requests to classes or handler methods, which perform data access and application logic, which can then be resolved to *views* to provide a visual representation of the data [12]. Spring supports JSP as one such method of providing *views*, such that HTTP requests are mapped to the appropriate JSP file on the disk. This means that HTTP requests do not request JSP files directly, rather that Spring handles how JSPs are returned in response to incoming queries.

Furthermore, dependency injection provided by Spring allows for resources to be managed without the programmer's explicit specification in the code. Necessary resources will be set up and destroyed automatically, so the programmer need not account for these things when creating the application logic.

6 Service Architecture

This section describes the implementation of the web service portion of the system, supporting search.

6.1 Service Implementation

The Java application is to be deployed on an Apache Tomcat 8.5.4, an open source implementation of a Java servlet container. Spring MVC facilitates separation of concerns, so

changes to the data objects from the earlier part of the system are not necessary. A data access layer is implemented of top of this, in the form of data access objects (DAO), which contain the logic required to facilitate search with Hibernate and retrieve the appropriate information from the database.

Resource handling is also managed by Spring through dependency injection, such as the set up and management of Hibernate sessions used to communicate with the database. The session is never explicitly created or closed by the application code, and Hibernate session configuration is provided through an ApplicationContext file.

A controller class is implemented to map request URLs to the appropriate handling methods to generate dynamic JSP content. Spring allows for mappings to be specified through the use of annotations. The following requests are mapped:

- **/search:** This request, without any extensions rebuilds the database index. Rather than the *view* for this being resolved to a JSP file, the user is redirected to the index page.
- **/search/{terms}:** "terms" is a path variable, presenting the entire search String. This may consist of one or more words, and does not need to be tokenized before creating a database query. The search result data is resolved to *results.jsp*.

6.2 User Interface

The user facing portion of the web service is delivered through JSPs, which are translated into HTML pages on the client-side. Each URL request is mapped to a unique JSP, which provide the page contents, while CSS is used to apply the visual layout and page design. The main page consists of a simple search field and search button. The function to rebuild the Lucene index is also invoked from this page, although there is no unique page associated with this (the user will remain on the index page). Upon performing a search, the user is directed to another page which lists the search results in a table, showing both image name and file path. There are two JSP files in total:

- **index.jsp:** Acts as the home page of the web interface. This page allows the user to perform a search and rebuild the search index. Fig. 5 illustrates this page.
- **results.jsp:** Displays the search results as a table (pictured in Fig. 6). The user is automatically directed to this page after performing a search.

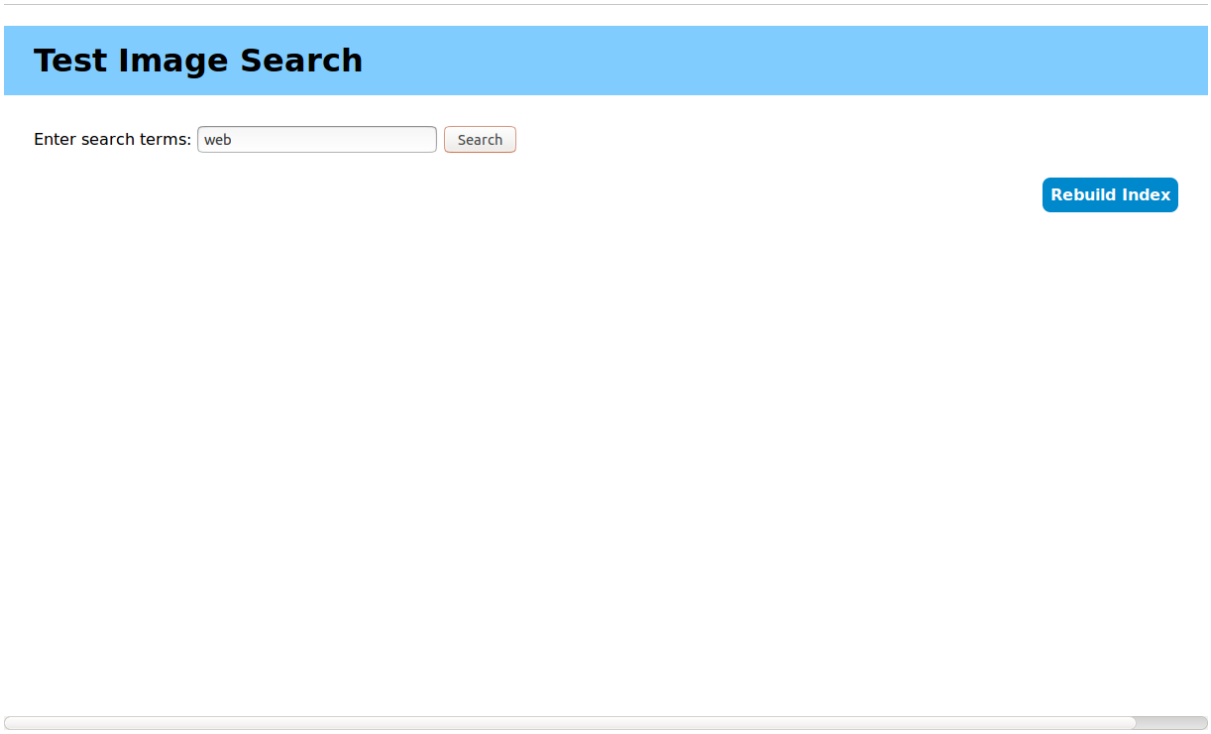

Figure 5. Appearance of the index page.

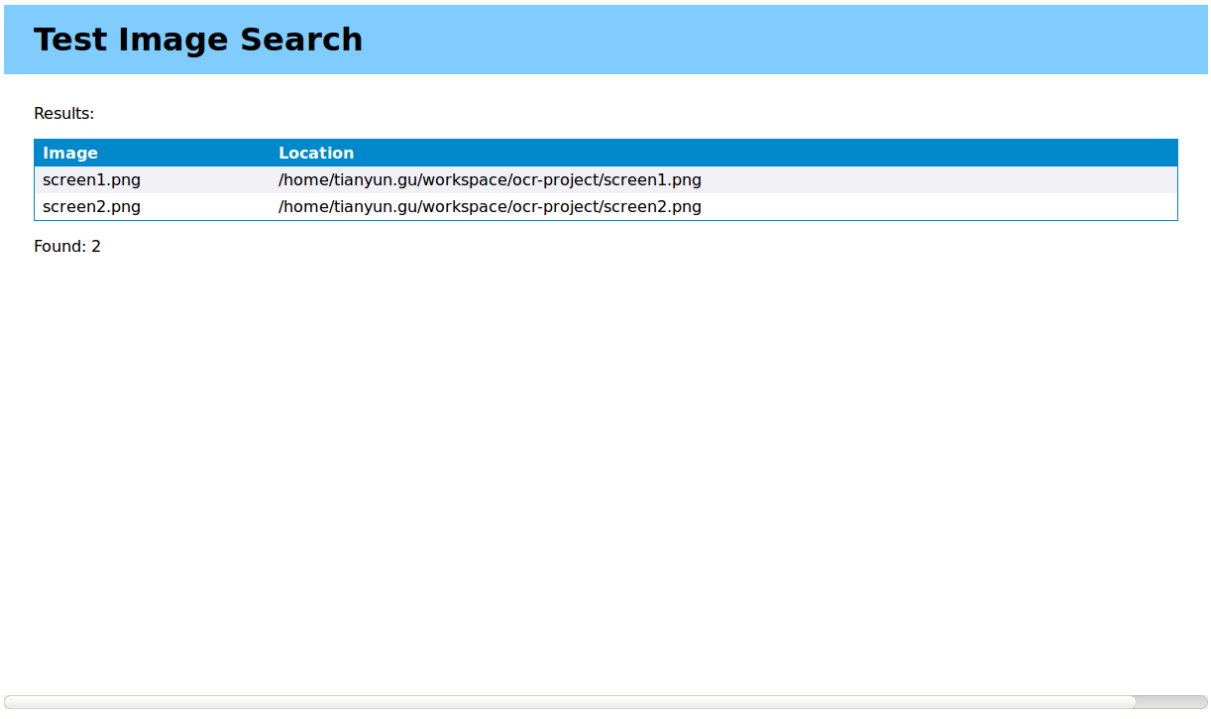

Figure 6. Appearance of the results page.

7 Future Work

My work over the year on this project has been to successfully implement a system which indexes images for search based on the text present. The index terms are extracted through OCR to what has been deemed an acceptable level of accuracy for this fairly simple implementation. I propose number of possible extensions to this work for the future.

Firstly, the accuracy of OCR with Tesseract could be improved through exploring additional fine tuning steps in the image pre-processing stage. The current steps taken for pre-processing are fairly basic and in depth research into this was not carried out for the project, in the interest of advancing progress. Furthermore, Tesseract OCR engine provides training data, which allows a user to train Tesseracts character recognition to be more refined according to styles of text found for the user's specific purposes. This could likely solve the problem described in section 2.3 where certain characters are consistently identified incorrectly.

The search features available to the user is another possible area for development. In the system's current implementation, searches performed by a user containing multiple search terms will only display results containing the entire list terms. This can be changed through configuration of the Hibernate session, and options to customise how results are qualified could be a useful extension to the existing system. Full text search also supports options such as wildcard searches, partial matching, and word stemming, which have not been considered for the sake of simplicity.

Finally, the user interface could be further enhanced, as the current implementation only demonstrates the basic functionalities of the search system. For user convenience, it may be desirable to add the ability to retrieve the image data of search results through the search interface, or otherwise present to the user some visual representation of the image. Search results are not currently paged. The entire list of results is retrieved all at once, which may create scalability issues if the number of results is very large. Navigation through pages could also be improved.

8 Appendixes

Appendix A

```
Prepare original Data to master data

15XThu 15:00

Summary of Configuration data
Data-base type

Num.

of corrugators

HYSQL

1

Current $DATA dir: Xmapqafmap3_testing/

I 11
I 21
```

```
I 31
I 41

I 91
I101
I111
I121
I131
I141

Storage units
Active CASE corrugs

I 51
I 61
I 71
I 8)

Start_Date 01X01X04
Finish_Date 07X15X04
Heb Uidth Contrib 5.00
SKU Value 2000.00
Cor BB Hin.T Hax.T Hax.Roll

1 A 1.00 6.00 98.00

1 B 1.00 6.00 84.08

1 C 1.00 6.00 98.00

1 D 1.00 6.00 84.08

1 E 1.00 6.00 98.00

1 F 1.00 6.00 98.00

(Clhange, (PJroceed, (Eint or Ix ? I....

Cost_per_splice
Cost_per_wid_ch
Cost_per_flu_ch
Output logfile:

16
1
10.00
50.00
100.00

prepout.log


```

Appendix B

```
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://www.hibernate.org/dtd/hibernate-mapping">

<hibernate-mapping package="kiwiplan.ocrproject">

  <class name="TestImage" table="IMAGE">
          <id name="imageId" column="ImageID">
               <generator class="increment"/>
          </id>
    <property name="imageName" column="ImageName" />
    <property name="imagePath" column="ImagePath" />
          <set name="imageTags" table="TAGMAP" cascade='all' inverse="false">
      <key>
        <column name="ImageID" not-null="true"/>
      </key>
      <many-to-many entity-name="kiwiplan.ocrproject.ImageTag">
        <column name="TagID" not-null="true"/>
      </many-to-many>
    </set>
     </class>

</hibernate-mapping>
```

9 References

[1] C. Adams, "Making Scanned Content Accessible Using Full-text Search and OCR", *Blogs.loc.gov*, 2014. [Online]. Available: https://blogs.loc.gov/digitalpreservation/2014/08/making-scanned-content-accessible-using-full-text-search-and-ocr/.

[2] "New Zealand Journal of History", *nzjh.auckland.ac.nz*, 2016. [Online]. Available: http://www.nzjh.auckland.ac.nz/.

[3] A. Pashintsev, "Evernote Indexing System", *Evernote Blog*, 2011. [Online]. Available: https://blog.evernote.com/tech/2011/09/30/evernote-indexing-system/.

[4] B. Kelly, "How Evernote's Image Recognition Works", *Evernote Blog*, 2013. [Online]. Available: https://blog.evernote.com/tech/2013/07/18/how-evernotes-image-recognition-works/.

[5] J. White and G. Rohrer, "Image Thresholding for Optical Character Recognition and Other Applications Requiring Character Image Extraction", *IBM Journal of Research and Development*, vol. 27, no. 4, pp. 400-411, 1983.

[6] R. Smith, "An overview of the Tesseract OCR engine". In *icdar,* pp. 629-633, IEEE, 2007

[7] "The Xapian Project : Features", *Xapian.org*, 2016. [Online]. Available: https://xapian.org/features.

[8] "Theoretical Background", *Xapian.org*, 2016. [Online]. Available: https://xapian.org/docs/intro_ir.html.

[9] I. LLC, "ImageMagick: Convert, Edit, Or Compose Bitmap Images", *Imagemagick.org*, 2016. [Online]. Available: http://www.imagemagick.org/script/index.php.

[10] "Chapter 5. Basic O/R Mapping", *docs.jboss.org*, 2016. [Online]. Available: https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/mapping.html.

[11] T. M. Gottfried, “JavaServer Pages in a Nutshell.(Technology Information)”, *Enterprise Systems Journal*, vol. 16, no. 6, 2001.

[12] R. V. Thakare, S. Kakade, B. Sapre and B. B. Meshram, "Spring MVC Framework for Web 2.0." *International Journal of Engineering Innovations and Research,* vol. 1, no. 3, 2012.