5/6/2016

Final Report

CM3203 – One Semester Individual Project, 40 credits

Student name: Aleksandar Kochev

Student number: 1229822

Supervisor: Irena Spasić

Moderator: Wendy K. Ivins

Acknowledgments

I would like to thank Dr. Irena Spasić for her unshakable trust in me and my abilities to complete this project successfully. She was always there when I needed help or advice. In addition to academic support, my weekly meetings with her were always enjoyable and entertaining, which made this final year project experience a lot better.

I would also like to thank Boris Ivanov (a fellow coursemate in BSc Computer Science) who has always given me moral support and cheered me up with his enthusiasm. During our final semester he proved to be a very interesting person to be around and have discussions with about various topics, especially problem solving. I can honestly say that I would not have enjoyed this project as much without his support and sense of humour.

TABLE OF CONTENTS

1. Introduction

We use search everywhere these days. And that is a good thing, because search helps us finish tasks quickly and easily. Whether we are buying something from an online store or visiting a blog, we naturally expect to have a search box somewhere to help us find what we are looking for, without scanning the entire website to do so. We have also come to expect search boxes to be smart. I do not want to have to type in an entire word or phrase, I expect the search box to come up with suggestions, and I do not want results and suggestions to come to me in random order. I want the search to give me the most relevant results first – to guess what I want, if that is possible. For example, if I search for "tablet" in an online shop, but have to scroll through tablet accessories before I get to an actual tablet, I am more likely to go somewhere else after the first page of search results. This is not only because I am in a hurry and spoiled with good search interfaces, but it is also because there is increasingly more content out there to choose from. [1]

In most real world scenarios a good keyword search is often not enough, you need filters and aggregations, so you can narrow down the results to what the user is interested in. Finally, there is the matter of performance, because nobody wants to wait. There are websites out there, where you search for something and get the results after a few minutes. *Minutes!* For a simple search…

In this report I will be discussing the challenges I was confronted with while implementing an information retrieval system and my approach to solving them. The system itself is aimed at helping epidemiologic studies of knee conditions using preprocessed and structured data extracted directly from MRI reports. Because of its purpose and the technologies used in making it I took the liberty of naming my creation GenuSearch (from Latin: genu - "knee").

2. Motivation and Background

2.1. Magnetic resonance imaging and its significance in the diagnosis of knee pathology

Magnetic resonance imaging (MRI) is a technique used to visualise internal body structure by recording radio waves emitted by the tissues in the presence of a strong magnetic field. MRI does a better job at differentiating between soft tissues than X-ray imaging, which uses high frequency electromagnetic waves that pass through soft parts of the human body to create a radiograph, an image resulting from the different absorption rates of different tissues. MRI can also produce three dimensional (3D) images. When it comes to diagnosing knee pathology, MRI has the advantage of visualising all structures within the knee joint, i.e. both soft tissue and bone. When used in conjunction with medical history and physical examination, MRI becomes a valuable tool for increasing diagnostic accuracy and planning surgical treatments. For example, meniscal tears are a relatively common knee injury, having a prevalence of 22.4 % among all soft tissues injuries seen in a trauma department. The accuracy of diagnosing meniscal tears using individual physical tests is reported to be 74 %, but increases to 96 % when MRI is used. [2]

Lately, the importance of MRI in diagnosis and treatment planning for cases of symptomatic early knee osteoarthritis has been emphasized on. If an X–ray image of the knee is found to be normal, but a clinical examination produces specific findings, then a MRI scan can be performed to establish a more accurate diagnosis. It can also be used to identify an appropriate surgical or nonsurgical treatment target and decrease the need for costly and invasive diagnostic arthroscopy. [2]

In clinical practice, radiology images (e.g. the ones produced by MRI or X–ray) are usually accompanied by imaging reports (or radiology reports), which serve the purpose of conveying a specialist interpretation of the images and relate it to the patient's signs and symptoms in order to suggest a diagnosis. This information is then used by clinicians to support decision making on an appropriate treatment. [2]

In terms of research, MRI evidence is often used to support epidemiologic studies of knee pathology. In particular, MRI findings are indispensable features of longitudinal studies of knee osteoarthritis, where lesions detected by MRI were found to precede the onset of clinical symptoms. However, many of the published research findings are assumed to be false due to sampling bias and low statistical power. Small sample size is often the cause underlying these two concerns although the relationship is not simple or proportional. Unfortunately, sample size is typically subject to funding and personnel constraints. Given the complexity and cost of manual interpretation of MRI evidence, it is, therefore, not surprising that the size of such epidemiologic studies has been limited to hundreds or even dozens of cases. If the interpretation of evidence described in MRI reports could be automated, then it would overcome the size limitation in retrospective cohort studies posed by the need to manually sort through the evidence. [2]

2.2. TRAK (Taxonomy for RehAbilitation of Knee conditions)

TRAK is an ontology that formally models knowledge relevant for the rehabilitation of knee conditions [3]. This information includes classification of knee conditions, detailed knowledge about knee anatomy and an array of healthcare activities that can be used to diagnose and treat knee conditions. Therefore, TRAK provides a framework that can be used to collect coded data in order to support epidemiologic studies much in the way Read Codes, a coded thesaurus of clinical terms [4], are used to record observational data in the Clinical Practice Research Datalink (CPRD) – formerly known as the General Practice Research Database (GPRD) [5]. TRAK follows design principles recommended by the Open Biomedical Ontologies (OBO) Foundry and is implemented in OBO, a format widely used by this community. [2]

Initially, TRAK was developed with a specific task in mind – to formally define standard care for the rehabilitation of knee conditions. At the same time, however, it was designed to be extensible in order to support other tasks in the domain. For example, the knowledge about knee anatomy, which is cross–referenced to a total of 205 concepts in the Foundational Model of Anatomy (FMA) [6], is directly applicable to interpretation of reports describing knee MRI scans. However, in order to fully support semantic interpretation of this type of clinical narratives, the TRAK ontology had to be expanded with other types of MRI–specific concepts. [2]

2.3. Natural language processing, information extraction and KneeTex

Natural language processing (NLP) is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages. Some of the main challenges in NLP involve: natural language understanding, enabling computers to derive meaning from human or natural language input, while others involve natural language generation. [7]

Information extraction (IE) is the task of automatically selecting specific facts about pre–specified types of entities and relationships from free–text documents. In other words, the overall goal of IE is to convert free text into a structured form by filling a template (a data structure that has predefined slots) with the relevant information extracted. [2][13]

Because of the fact that clinical narratives, such as those found in radiology reports, convey valuable diagnostic information that is predictive of the prognosis and biological behaviour of a disease process, different studies [8-12] have been conducted to test the feasibility of natural language processing for information extraction in this field. The results of these studies have proven that for simpler information extraction tasks, human–like performance of automated systems can be expected. However, none of the studies had focused specifically on MRI reports in relation to knee pathology, possibly due to the complexity of knee anatomy and a wide range of conditions that may be associated with the different anatomical entities. [2]

Typical processing steps taken in NLP systems related to radiology reports usually include text segmentation into words, sentences, paragraphs and/or sections, part–of–speech tagging, parsing, named entity recognition (NER), normalisation and negation annotation. Recognition of named entities, i.e. phrases that are used to differentiate between entities of the same semantic type (e.g. Osgood-Schlatter disease is a name used to refer to a specific disease), followed by normalising the representation of their meaning (e.g. Osgood-Schlatter disease is also known as apophysitis of the tibial tubercle or OSD), is the crucial step towards semantic interpretation of clinical narratives. In order to disambiguate named entities and assert relationships between them (e.g. relate disease/disorder, sign/symptom or procedure to an anatomical site), domain–specific knowledge needs to be available in a machine–readable form. [2] The existence of the TRAK ontology, as a formally structured knowledge resource about the rehabilitation of knee conditions, allowed the implementation of an NLP system able to interpret knee–related clinical findings from MRI reports. As a result – KneeTex was developed.

KneeTex is an open–source, stand–alone application developed to address the task of information extraction from narrative reports that describe an MRI scan of the knee. KneeTex is an ontology–driven, rule–based system. It takes an MRI report as an input and outputs the corresponding clinical findings in the form of JavaScript Object Notation (JSON) objects. KneeTex not only extracts, but also maps the extracted information to the TRAK ontology. The resulting formally structured and mapped data allows complex searches to be conducted efficiently over the original MRI reports, thereby effectively supporting epidemiologic studies of knee conditions. [2]

2.4. The aims of this project

From the very beginning, the aims of this project have revolved around the completion of six big tasks:

1. Creating a database that would store:
   (a) the original MRI reports
   (b) the structured KneeTex output data, created from processing the original MRI reports
   (c) the TRAK ontology
2. Creating a simple, user-friendly search interface over the database, consisting of:
   (a) Query formulation (a Search Page)
   (b) Presentation of the search results (a Search Results Page)
3. Matching the queries generated from the UI (User Interface) to the data stored in the database and retrieving the right results.

4. Implementing a suitable ranking algorithm that would score the relevance of search results, so they can be displayed in proper order.
5. Evaluating the overall correctness of information retrieval.
6. Testing the final product’s user interface and drawing conclusions from user feedback.

The successful completion of these tasks ensures that the finalised version of this project can effectively aid future epidemiologic studies in the sphere of knee pathology. The target users of this system will be researchers in the physiotherapy domain who wish to conduct analysis of knee conditions.

3. Specification and Design

3.1. Deciding on which technologies to use

One of the first discussions I had with my supervisor was about determining the most suitable medium for this project. We both agreed that creating a stand-alone web application would be the best choice, because of the ubiquity of web browsers and their cross-platform compatibility. Another convenience of using a web browser as a client is the ability to update and maintain a system without distributing and installing software on potentially hundreds or thousands of client computers. [14]

Due to the vast popularity of web applications in this day and age there is also a large amount of web development technologies to choose from. Some good examples would be: Node.js, PHP, Python, Ruby on Rails, ASP and Java. With those, however, you can only build one part of a complete system. Web applications, as well as web development technologies, are usually broken into logical chunks called "tiers", where every tier performs a single role. Though many variations are possible, the most common structure is the three-tiered application, where the three tiers are called presentation (front-end), application (back-end) and storage (database), in this order. [14]

After closely examining the aims of this project, it became clear to me that I had to create a three-tiered web application. Because of this conclusion and some previous experience in the field of web development, I decided to use the following technologies:

- **HTML5, CSS3 and Bootstrap** *(presentation tier)*

  Hyper**T**ext **M**arkup **L**anguage (HTML) is the standard markup language used to create web pages. Alongside CSS and JavaScript, it is one of the three essential technological blocks used in mobile and web development. Web browsers can read HTML files and render them into user-friendly, meaningful web pages. HTML describes the structure of a website semantically. [15]

  Cascading **S**tyle **S**heets (CSS), is a styling language used for describing the presentation of a document written in a markup language. Its main purpose is to enable the separation of document content from document presentation, including aspects such as the layout, colours, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple HTML pages to share the same formatting and reduce complexity and repetition in the overall structure of a web page. [16]

  Bootstrap is a free and open-source front-end library for creating websites and web applications. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components. It was created to ease the development of dynamic websites and support responsive web design, which means that the layout of those pages can adjust dynamically, taking into account the characteristics of the device being used to access them (desktop, tablet, mobile phone). [17]

Together, these three technologies allowed me to easily create new web pages and tailor their appearance to the application's needs.

- **JavaScript with jQuery** *(presentation and application tier)*

  JavaScript is the programming language of HTML and the Web. It is supported on every modern web browser. While HTML and CSS are used to create static web pages, JavaScript is used to make them "come to life". Through its use, elements on a website can react to what a user is doing and display, hide or animate content based on his/her actions.

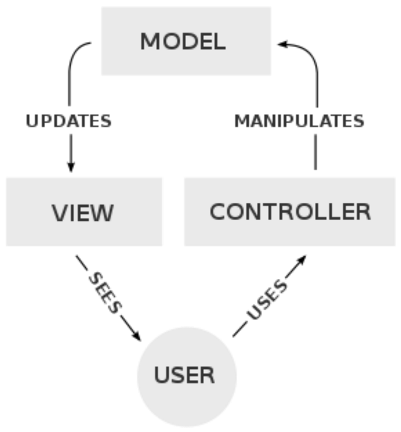
  jQuery is a cross-platform JavaScript library designed to simplify the manipulation of HTML and CSS on a web page. It also provides a shorter way of writing JavaScript functions and executing AJAX requests.

  I used these technologies to manipulate the appearance of GenuSearch based on the user's actions on the website. It also allowed me to make the application send queries to the back-end, execute them in the background and display their results without reloading or changing something on the page.

- **Java and the Play Framework** *(application tier)*

  Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented and specifically designed to have as few implementation dependencies as possible. Java code can run on all platforms that support it without the need for recompilation. It is also one of the most popular programming languages in use, especially for client-server web applications, with a large interactive community on the Internet. [18]

  Java is the language I have the most experience with, which made it a suitable candidate to use in implementing the more complex logical blocks of this system. Also, since this was forming to be a 3-tier web application, I had to find a way to split the project into logical parts (front-end, back-end and data), so I decided to use the Java Play framework to do this job for me. Play is based on a lightweight, stateless, web-friendly MVC (Model-View-Controller) architecture and features predictable and minimal resource consumption (CPU, memory, threads) for highly-scalable applications.



➔ Models represent the data. They are stored in and retrieved from the database (Elasticsearch)

➔ Views are what the user sees (HTML, CSS, Bootstrap)

➔ Controllers are event handlers for user actions (Java and JavaScript + jQuery)

Figure 1. Model-View-Controller Design Pattern

- **Elasticsearch** *(storage tier)*

Elasticsearch is a full-scale, modern search engine that also does a good job storing NoSQL data. It is open-source, distributed and built on top of Apache Lucene, an open-source search engine library, which allows developers to implement search functionality in their own Java applications. Elasticsearch takes this functionality and extends it to make storing, indexing, and searching faster, easier, and, as the name suggests, elastic. [1]

Since KneeTex outputs data in JSON format it was reasonable for me to stick to it, instead of converting the documents to something else. Having had experience with two JSON data storage technologies: MongoDB and Elasticsearch, I decided to go with Elasticsearch, because Mongo is a just a general purpose NoSQL database with fewer query and aggregation options compared to Elasticsearch.

Some of the main benefits I derived from using Elasticsearch in this project are:

1. Automatic data indexing
An index is a data structure, which you create along with your data and is meant to allow faster searches. You can add indexes to fields in most databases (like MongoDB), and there are several ways to do it. Lucene does it automatically with inverted indexing, which means it creates a data structure where it keeps a list of where each word belongs. [1] For example, if you need to search for TRAK Terms by their names, using inverted indexing might look like the table in Figure 2.

2. Ranking algorithm
With Elasticsearch, you have a few algorithms for calculating the relevancy score, which is used, by default, to sort the results. The relevancy score is a number assigned to each document that matches the search criteria and indicates how relevant the given document is to the criteria. By default, the algorithm used to

calculate a document's relevancy score is TF-IDF, which comes from term frequency–inverse document frequency. [1]

- Term frequency — the more times the words you're looking for appear in a document, the higher the score.
- Inverse document frequency — the weight of each word is higher if the word is uncommon across other documents.

For example, if you're looking for "horizontal tear" through the MRI reports, the word "tear" would counts much less for the score than "horizontal". But the more times both words appear in a document, the higher that document's score.

Raw data		Indexed data	
ID	Name	Names	IDs
1	tear	tear	1, 2, 4
2	surface tear	surface	2
3	vertical	vertical	3, 4
4	vertical tear		

Figure 2. Inverted Index

3.2. Constructing data models

It is true that the quality of an information retrieval system is only as good as the quality of the data it uses, however a lot depends on how the data is structured as well. Having good data models makes a programmer's job easier, because it reduces the difficulty of displaying, querying, updating and deleting information from the system.

GenuSearch has a total of fifteen models, six of which can be classified as the "main" models, while the rest are sub-models, constructing only certain parts the main model they belong to. *(All of the system's models can be viewed within the app/models package of the application)*

KneeTexOutput
Term
TermExpansion
Suggestion
ReportDocument
SavedSearch

Figure 3. Main model names

KneeTexOutput

KneeTexOutput is the model made after the design of the KneeTex JSON output template. The reason for making this model is so that I can access KneeTex data inside of my application easily. It uses three sub-models: Source, OntologyMappedDataSegment and AnatomyObservation.

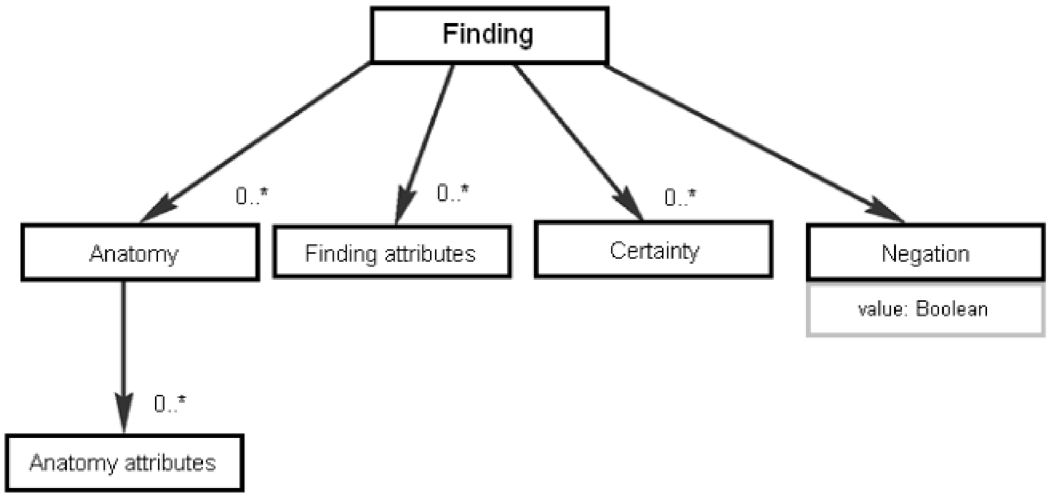


Figure 4. UML representation of a KneeTexOutput model

Term and Typedef

Term and Typedef objects are extracted directly from a normal *trak.obo* file. [Typedef] objects reporesent actions such as: affects, occurs, diagnoses, etc., there are only 16 of them and they do not have an actual use in GenuSearch, but are still extracted and put in a separate collection in case they need to be used in the future.

[Term] objects represent TRAK ontology terms. They number 1619 in total and each of them has a unique name and id. The sub-models used in the creation of the Term model are: CrossReference, Synonym and OntologyLink.

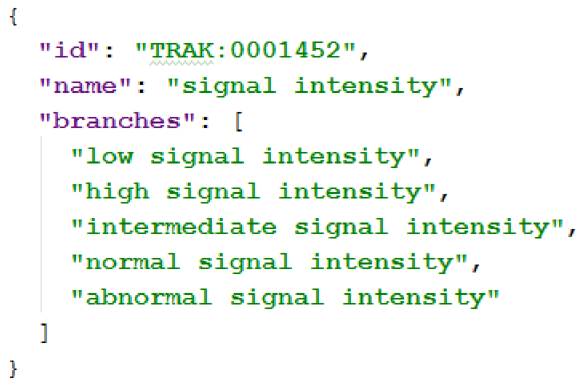
```
[Term]
id: TRAK:0000939
name: limp
def: "A limp is a type of asymmetric abnormality of the gait." [Wikipedia:Limp]
is_a: TRAK:0000893 ! gait abnormality
```

```
{
  "id": "TRAK:0000939",
  "name": "limp",
  "definition": "A limp is a type of asymmetric abnormality of the gait. ",
  "synonyms": [],
  "crossReferences": [],
  "is": [
    {
      "id": "TRAK:0000893",
      "name": "gait abnormality"
    }
  ],
  "relationships": []
}
```

Figure 5. Mapping a Term model from OBO to JSON format

TermExpansion

TermExpansion is a custom object used specifically for constructing queries. GenuSearch uses the TRAK ontology to expand the user's possibilities when searching. For example, if a user searches for a finding -> "*signal intensity*", the system will look up a TermExpansion object and see the "*signal intensity*" sub-tree (child branches) and look for any instance of signal intensity or its children in the KneeTex structured data as a finding.



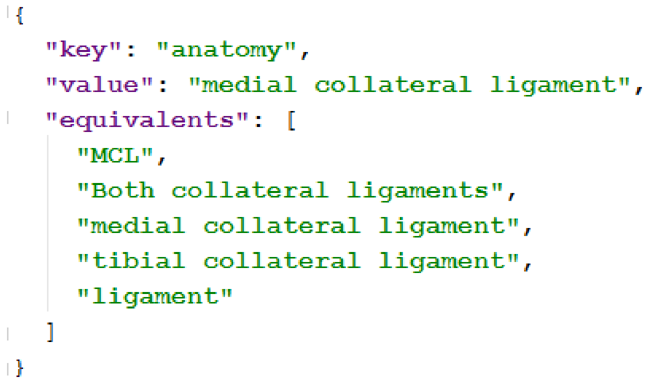
```
{
  "id": "TRAK:0001452",
  "name": "signal intensity",
  "branches": [
    "low signal intensity",
    "high signal intensity",
    "intermediate signal intensity",
    "normal signal intensity",
    "abnormal signal intensity"
  ]
}
```

Figure 6. TermExpansion model in JSON

It would be computationally infeasible to generate this type of data on each search, so TermExpansion objects are created only once, right after the TRAK ontology has been mapped and indexed into Elasticsearch.

Suggestion

Suggestions are quite essential for almost all types of search interfaces. GenuSearch provides four types of suggestions: *finding*, *finding attributes*, *anatomy* and *anatomy attributes*. They are all generated from the data contained within in the KneeTexOutput objects as well as the synonym values of ontology terms. However, only a single "unique" value is picked to be displayed as a suggestion (usually the one from the *name* field of a TRAK ontology term), while the rest are used to provide certain flexibility and convenience for user input.



```
{
  "key": "anatomy",
  "value": "medial collateral ligament",
  "equivalents": [
    "MCL",
    "Both collateral ligaments",
    "medial collateral ligament",
    "tibial collateral ligament",
    "ligament"
  ]
}
```

Figure 7. Suggestion model in JSON format

For example if a user inputs "*mcl*" into an anatomy input box -> "*medial collateral ligament*" would come up as a suggestion, because "*MCL*" exists as an *equivalent* value of that term in the suggestion object.

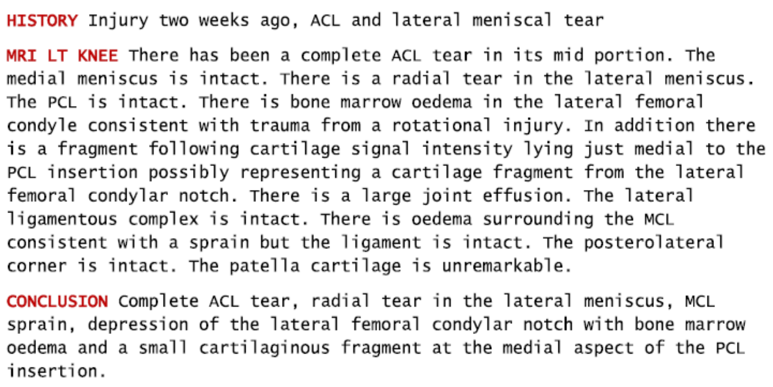
This makes suggestions on the system both smart and less time consuming.

ReportDocument

The original, unprocessed by KneeTex, version of the MRI report documents is also stored on the system. Although the structure of each report is different, they all generally tended to organise information under the following headings: [2]

- MRI OF THE LEFT/RIGHT KNEE
- INDICATION
- HISTORY
- FINDINGS
- CONCLUSION.

Within the reports, which were distributed in a plain text format, these sections were indicated with upper case.



HISTORY Injury two weeks ago, ACL and lateral meniscal tear

MRI LT KNEE There has been a complete ACL tear in its mid portion. The medial meniscus is intact. There is a radial tear in the lateral meniscus. The PCL is intact. There is bone marrow oedema in the lateral femoral condyle consistent with trauma from a rotational injury. In addition there is a fragment following cartilage signal intensity lying just medial to the PCL insertion possibly representing a cartilage fragment from the lateral femoral condylar notch. There is a large joint effusion. The lateral ligamentous complex is intact. There is oedema surrounding the MCL consistent with a sprain but the ligament is intact. The posterolateral corner is intact. The patella cartilage is unremarkable.

CONCLUSION Complete ACL tear, radial tear in the lateral meniscus, MCL sprain, depression of the lateral femoral condylar notch with bone marrow oedema and a small cartilaginous fragment at the medial aspect of the PCL insertion.

Figure 8. Knee MRI Report structure and contents [2]

Apart from its content, each report comes with a unique document id. The GenuSearch model of a knee MRI report adds a few extra fields, however, because of the term highlighting and report tagging options on the system.

```
{
  "id": "1205.txt.xml",
  "content": "MRI RIGHT KNEE There is no significant joint effusion .
  "defaultHighlighting": "MRI RIGHT KNEE There is no <code>significant</code>
  "highlightedSearchTerms": null,
  "combinedHighlighting": null,
  "tag": null
}
```

Figure 9. ReportDocument model in JSON format

Content highlighting is done by adding specific HTML tags between terms, by default:

<code> term </code>

Terms themselves are identified by using the KneeTex findings in each sentence of a report. If you inspect *Figure 9* more closely you might see that there are different types of highlighting:

- ➔ Default highlighting - all terms in the content of an MRI report are highlighted in the same way. This highlighting is done when the ReportDocument objects are gotten from SQLite and indexed into Elasticsearch.
- ➔ Search Terms Highlighting – only terms that are searched for are highlighted and their highlighting differs depending whether the term is a *finding*, *anatomy* or

attribute. This highlighting is done during runtime, hence the field is *set* only when the results are about to be sent to the front-end.

➔ Combined Highlighting (*not implemented*) – this should highlight the terms searched in one colour and all other terms in another.

Tags are set during runtime as well, because each report can be marked as *relevant*, *irrelevant* or *view later* for each different saved search.

SavedSearch

SavedSearch is a model that allows the user to store the queries he/she executes on the system. Saved searches can be given names and populated with tags about each search result that comes up after query execution. If a SavedSearch is deleted, all the report tags associated with it are removed as well.

```
{
  "name": "high signal intensity search",
  "queries": [
    {
      "finding": {
        "value": "high signal intensity",
        "type": "extended"
      },
      "anatomy": null,
      "findingAttributes": null,
      "anatomyAttributes": null,
      "exclude": null,
      "negated": false
    }
  ],
  "method": null,
  "relevantReports": ["1430.txt.xml", "991.txt.xml", "29.txt.xml"],
  "irrelevantReports": ["527.txt.xml", "500.txt.xml", "428.txt.xml"],
  "toViewLater": ["372.txt.xml", "1183.txt.xml"]
}
```

Figure 10. SavedSearch model in JSON format

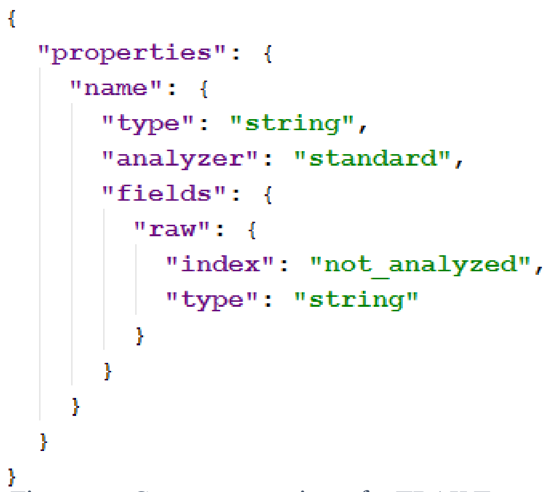
3.3. Data analysis

Analysis is the process Elasticsearch performs on the body of a document before the document is sent off to be indexed. Elasticsearch goes through a number of steps for every analysed field before the document is added to the index:

- Character filtering: Transform the characters using a character filter.
- Breaking text into tokens: Break apart the text into a set of one or more tokens.

- Token filtering: Transform each token using a token filter.
- Token indexing: Store those tokens into the index

Though analysis is useful most of the time, sometimes you need a field to stay as it is, without being broken down into tokens or transformed into lowercase. It can become very tricky to look for *exact* values in the inverted index. For example if you search for “complex tear” the query you execute can break this into “complex” and “tear”, which will bring back results that contain not only “complex tear”, but also only “complex” and only “tear”. For this reason I have made custom Elasticsearch mappings for all major models in order to remove any chance of bugs no matter the user input.



```
{
  "properties": {
    "name": {
      "type": "string",
      "analyzer": "standard",
      "fields": {
        "raw": {
          "index": "not_analyzed",
          "type": "string"
        }
      }
    }
  }
}
```

Figure 11. Custom mapping of a TRAK Term object

In this example Elasticsearch creates a custom *raw*, whose value is the same as that of the *name* field, however the *raw* field value does not get analysed during indexing. This happens automatically, for each Term.

(All mappings can be viewed within the “resources” package of the application)

Elasticsearch also allows the creation of custom filters and analysers. I found it appropriate to configure two such filters on the Suggestion model, in order to make user input interpretation better and type-ahead suggestions smarter. The two filters are known as the shingles filter (no not the disease!) and the Edge NGrams filter.

EDGE NGRAMS

Edge NGrams is one of the more unique ways of tokenizing text in Elasticsearch. The filter splits a token into multiple sub-tokens for each part of the word, starting from the front edge. For example, if we have the word “*osteoarthritis”* this analyser will breaking it down into:

“o”, “os”, “ost”, “oste”, “osteo”, “osteoa”, “osteoar,” “osteoart”, …

SHINGLES

The shingles token filter similar to an NGrams one, but it operates at the token level, instead of the character level. For example, if we have the phrase “high signal intensity” the shingles analyser will break it down to:

“*high*”, “*high signal*”, “*high signal intensity*”, “*signal*”, “*signal intensity*”, “*intensity*”

3.4. Pages and design

GenuSearch is comprised of five different pages:

- Search page
- Search results page
- Saved searches page
- Tagged reports page and
- Data configurations page (which usually should be accessible by the average user).

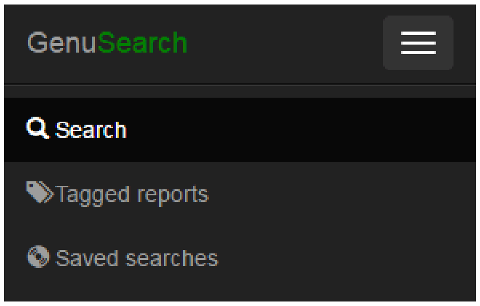


Figure 12. GenuSearch navigation bar

Search page (Main page)

Designing the application's search page appropriately was the most important task in this project. In order to do this, I had to determine how to expose inputs that would be effective links to the fields of a KneeTex object and at the same time not confuse the user with too much technical complexity. My efforts produced this result:

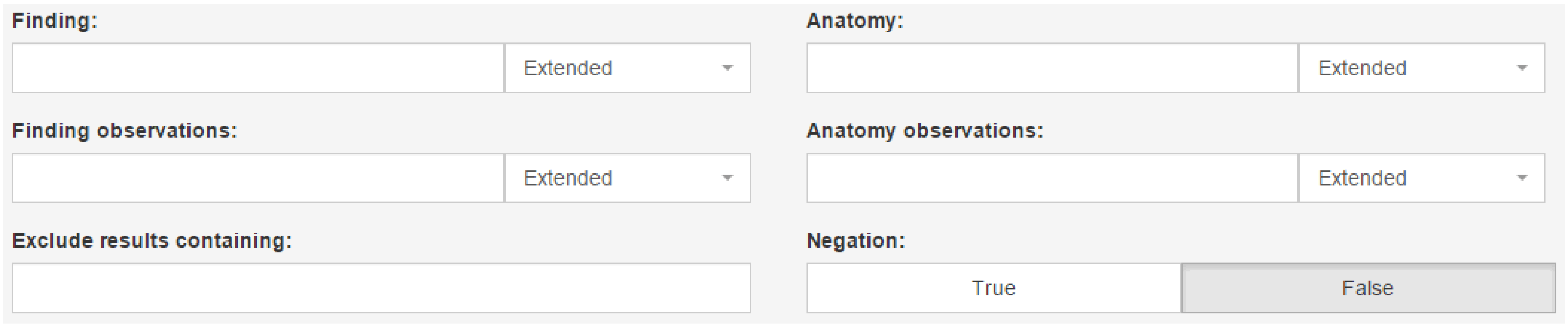
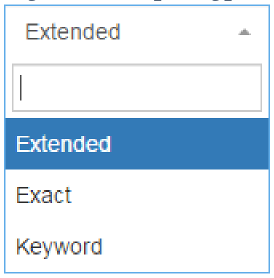


Figure 13. Search page query options

The *Finding* and *Anatomy* search boxes allow only a single input, while the other ones allow up to four inputs. Negation specifies whether the finding is positive or negative.

Figure 14. Input type



These are three options that indicate what type of input a user is meaning to use:

- *Extended* - means that the input will be matched against the TRAK ontology and all of its child branches will be used in the search as well

- *Exact* – means that the user wants to search for an exact term, without checking the TRAK ontology for child branches

- *Keyword* – instead of forcing the users to choose from suggestions, this option allows them to input free-text

The *exclude* box allows the users to select terms, which they do not wish to see in their search results. All types of suggestions can be displayed for this box: anatomy, finding, attributes as well as keyword.

Terms are automatically used in their "extended" form, meaning that all their child branches will be excluded from the search results as well.

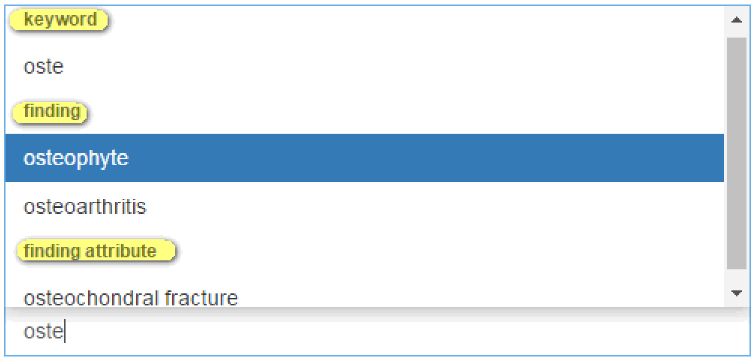
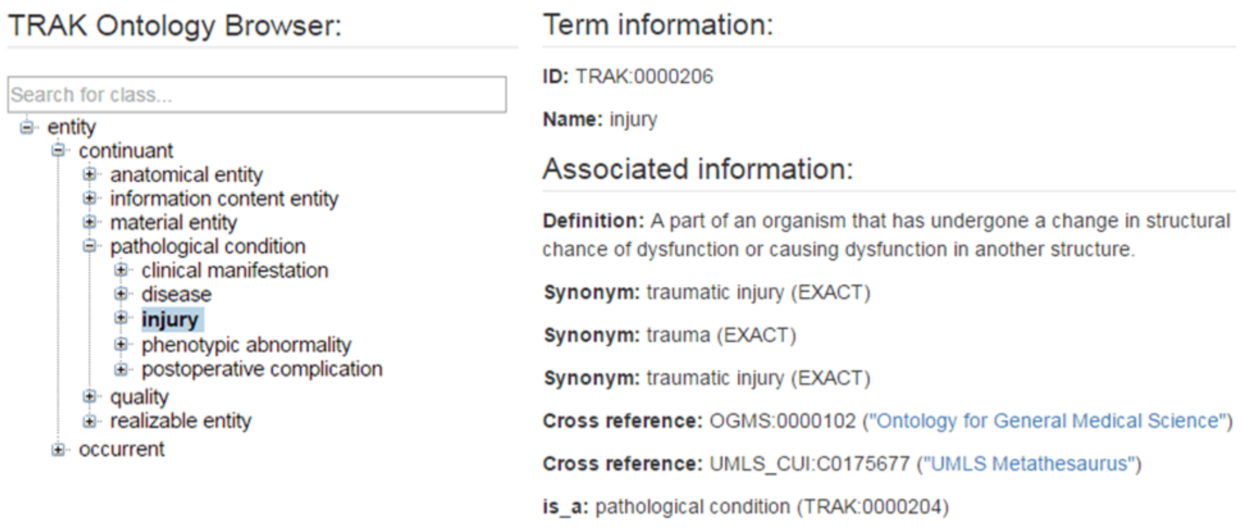


Figure 15. Exclude options

Search results page

The search results page is designed to provide the user with means of easily exploring and analysing knee MRI reports data. This page would contain the report documents relevant to the executed query, a TRAK ontology browser, together with Term explanation and highlighting options. Other functionalities on this page allow the download of search results in XML format, saving the executed search and tagging report documents.


TRAK Ontology Browser:

Search for class...

- entity
  - continuant
    - anatomical entity
    - information content entity
    - material entity
    - pathological condition
      - clinical manifestation
      - disease
      - injury
      - phenotypic abnormality
      - postoperative complication
    - quality
    - realizable entity
  - occurrent

Term information:

ID: TRAK:0000206

Name: injury

Associated information:

Definition: A part of an organism that has undergone a change in structural chance of dysfunction or causing dysfunction in another structure.

Synonym: traumatic injury (EXACT)

Synonym: trauma (EXACT)

Synonym: traumatic injury (EXACT)

Cross reference: OGMS:0000102 ("Ontology for General Medical Science")

Cross reference: UMLS_CUI:C0175677 ("UMLS Metathesaurus")

is_a: pathological condition (TRAK:0000204)

Figure 16. TRAK Ontology Browser

Both the browser and the explainer react to what terms the user clicks on or searches for.

Highlighting options, on the other hand, allow a user to choose how search results content is displayed.

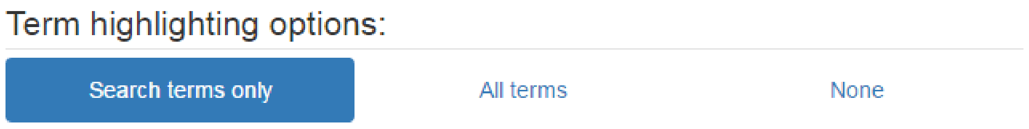
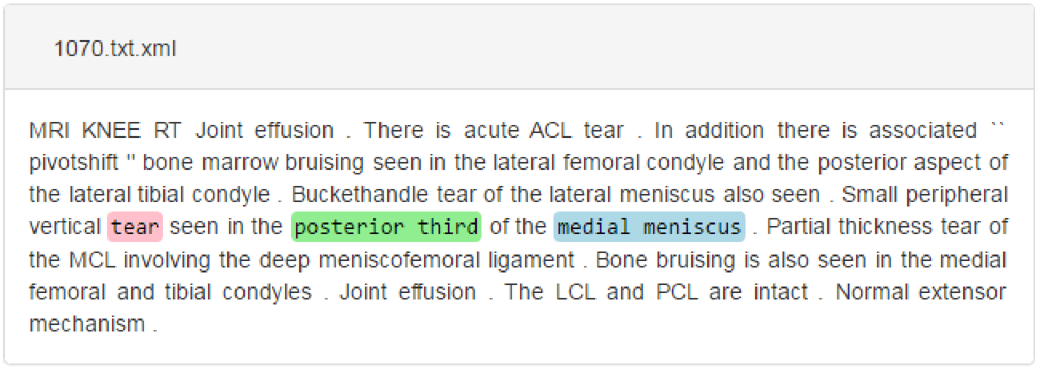


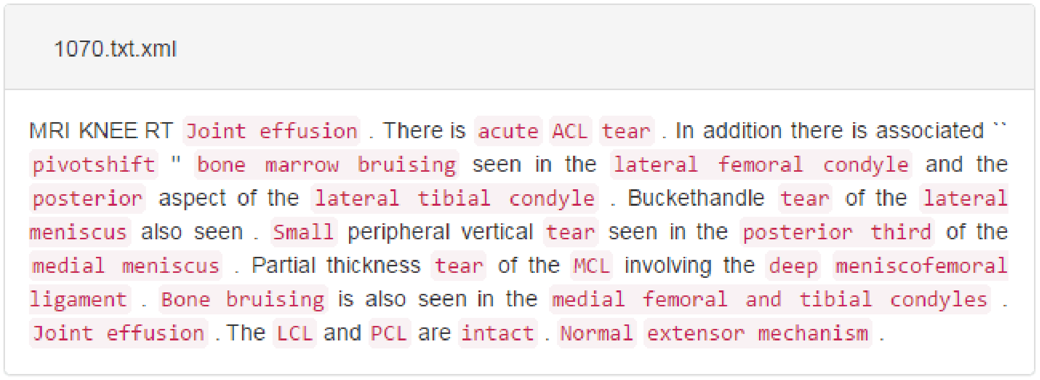
Figure 17. Highlight options


1070.txt.xml

MRI KNEE RT Joint effusion . There is acute ACL tear . In addition there is associated `` pivotshift '' bone marrow bruising seen in the lateral femoral condyle and the posterior aspect of the lateral tibial condyle . Buckethandle tear of the lateral meniscus also seen . Small peripheral vertical tear seen in the posterior third of the medial meniscus . Partial thickness tear of the MCL involving the deep meniscofemoral ligament . Bone bruising is also seen in the medial femoral and tibial condyles . Joint effusion . The LCL and PCL are intact . Normal extensor mechanism .

Figure 18. Highlighted search terms

A search result is composed of the report document id and its content. If a user includes a finding, anatomy or observations, they will be highlighted in red, blue and green respectively. If the search criteria a user specifies is only negation or exclusion, all terms will be highlighted and the “Search Terms Only” highlighting option will be greyed out and prohibited from usage (since it would be the same as no highlighting at all).


1070.txt.xml

MRI KNEE RT Joint effusion . There is acute ACL tear . In addition there is associated `` pivotshift '' bone marrow bruising seen in the lateral femoral condyle and the posterior aspect of the lateral tibial condyle . Buckethandle tear of the lateral meniscus also seen . Small peripheral vertical tear seen in the posterior third of the medial meniscus . Partial thickness tear of the MCL involving the deep meniscofemoral ligament . Bone bruising is also seen in the medial femoral and tibial condyles . Joint effusion . The LCL and PCL are intact . Normal extensor mechanism .

Figure 19. All terms highlighted

Another two key functionalities on the search results page are the Download/Export search results option and save search option.

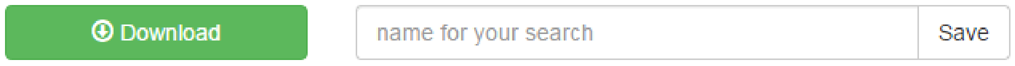
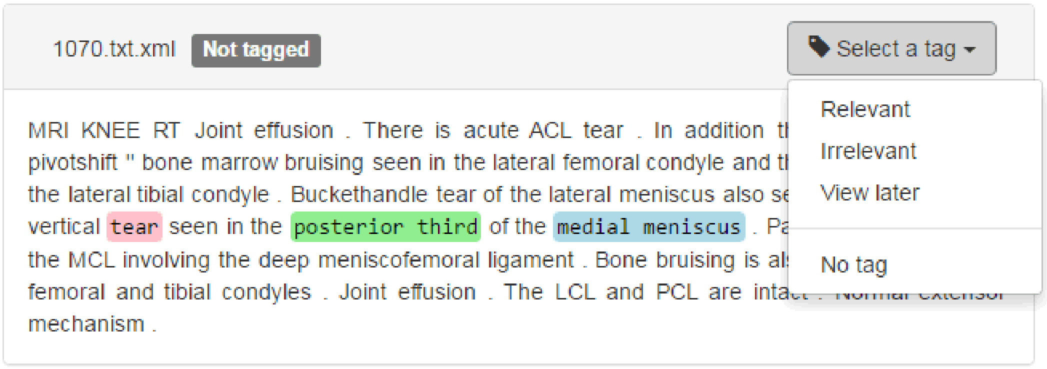


Figure 20. Download and save search buttons

The users are allowed to specify a name for their saved search, in order to help them differentiate between them. This it is not obligatory however, so if they just clicks *Save* without giving an input, the current date and time will be used for creating a unique name.

After saving, tagging options will appear on each report allowing the user to mark each of them as either *relevant*, *irrelevant* or *view later*. This has a more specific use later on in the *Tagged reports* page.



1070.txt.xml Not tagged

Select a tag

Relevant

Irrelevant

View later

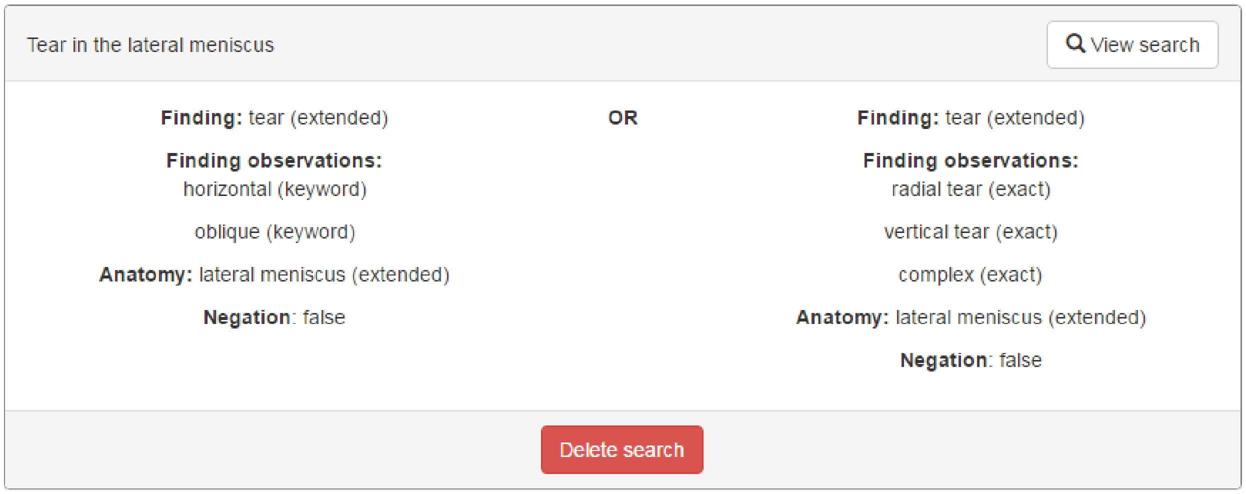
No tag

MRI KNEE RT Joint effusion . There is acute ACL tear . In addition t
pivotshift " bone marrow bruising seen in the lateral femoral condyle and t
the lateral tibial condyle . Buckethandle tear of the lateral meniscus also se
vertical tear seen in the posterior third of the medial meniscus . Pa
the MCL involving the deep meniscofemoral ligament . Bone bruising is al
femoral and tibial condyles . Joint effusion . The LCL and PCL are intact . Normal extensor
mechanism .

Figure 21. Tagging options

Saved searches page

The saved searches page is just a listing of all the saved searches and all their details. This page allows to user to execute previously saved queries or permanently delete them.



Tear in the lateral meniscus

View search

Finding: tear (extended)
Finding observations:
horizontal (keyword)
oblique (keyword)
Anatomy: lateral meniscus (extended)
Negation: false

OR

Finding: tear (extended)
Finding observations:
radial tear (exact)
vertical tear (exact)
complex (exact)
Anatomy: lateral meniscus (extended)
Negation: false

Delete search

Figure 22. Saved search example

Tagged reports page

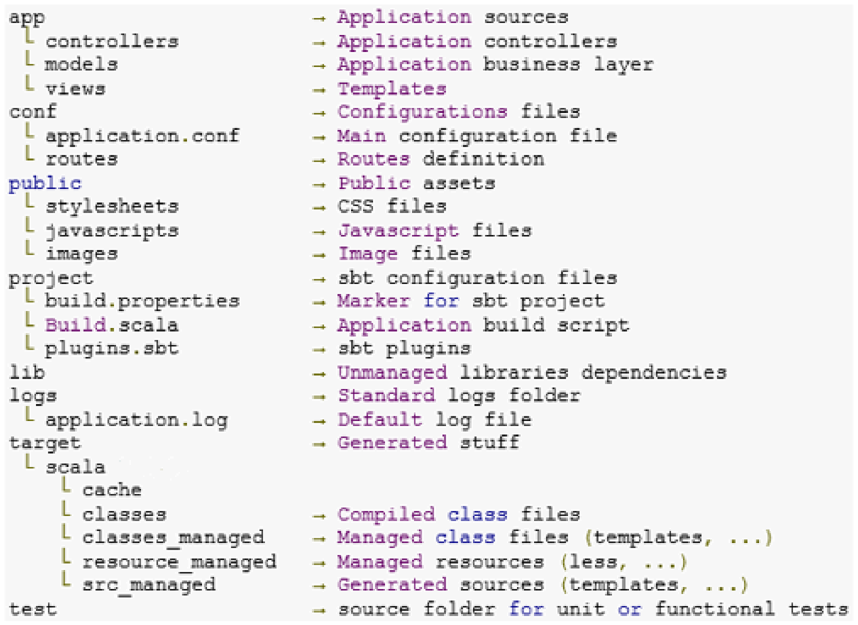
This page is very similar to the *Search results* one, however the key differences are that it is only available for saved searches and allows the use of tag filters. These filters allow the system to produce more precise results and exports. For example, a user can filter through all the reports and leave only the *relevant* ones on screen, inspect them and export only them.

Figure 23. Filter options

All other important functionalities, such as the TRAK Ontology Browser and the term highlighting options, are still available on this page.

4. Implementation

4.1. Project structure



```
app                         → Application sources
 └ controllers              → Application controllers
 └ models                   → Application business layer
 └ views                    → Templates
conf                        → Configurations files
 └ application.conf         → Main configuration file
 └ routes                   → Routes definition
public                      → Public assets
 └ stylesheets              → CSS files
 └ javascripts              → Javascript files
 └ images                   → Image files
project                     → sbt configuration files
 └ build.properties         → Marker for sbt project
 └ Build.scala              → Application build script
 └ plugins.sbt              → sbt plugins
lib                         → Unmanaged libraries dependencies
logs                        → Standard logs folder
 └ application.log          → Default log file
target                      → Generated stuff
 └ scala
    └ cache
    └ classes               → Compiled class files
    └ classes_managed       → Managed class files (templates, ...)
    └ resource_managed      → Managed resources (less, ...)
    └ src_managed           → Generated sources (templates, ...)
test                        → source folder for unit or functional tests
```

Figure 24. Project structure

The application layout, which the Play framework provided, is standardized to keep things as simple as possible. Because of the MVC (Model-View-Controller) architectural design pattern it was easy to separate Java, HTML, CSS and JavaScript files, and at the same time write loosely coupled and highly cohesive code.

4.2. HTTP request handling

Two of the biggest benefits of using the Java Play framework are the HTTP server and router. In order for a web application to run it needs to have a stand-alone server to intercept incoming requests from browsers and a router to translate each incoming request to an action call. Controller classes are the ones used as a home for action methods, so they do all the heavy lifting in the application.

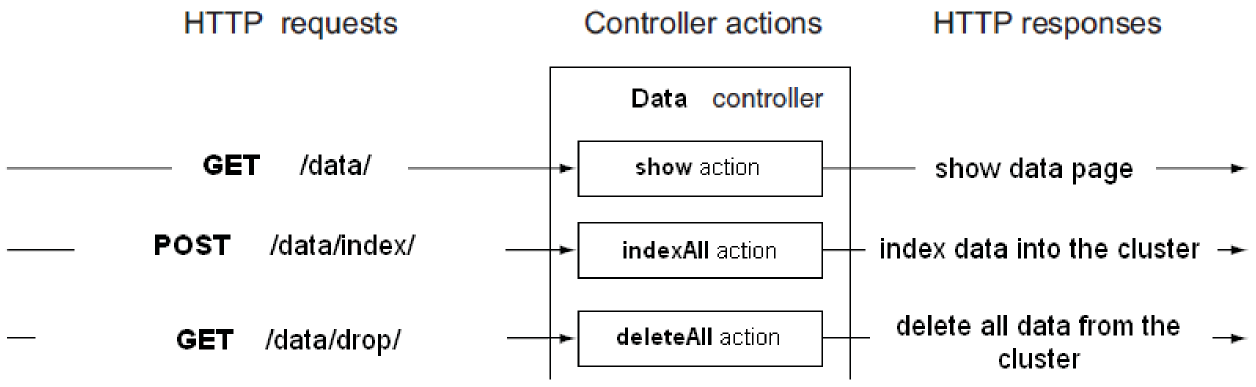


Figure 25. HTTP request handling example (Data Controller)

4.3. Data

In Elasticsearch the unit used for indexing and searching is called a *document*, very much like a row in a relational database. Documents are grouped into mapping types, or just *types*, which contain documents in a similar way to how tables contain rows. Finally, one or multiple *types* live in an *index*, the biggest container, similar to a database in the SQL world. Elasticsearch divides each index into *shards* that make up a *cluster*.

GenuSearch has its own Elasticsearch *cluster* that is named after it. The *index* that contains all the different *types* is called "app" and is configured to have five shards in order to maximize performance. Each document in the GenuSearch cluster corresponds to a model in the application and all identical models are placed into a single *type*.

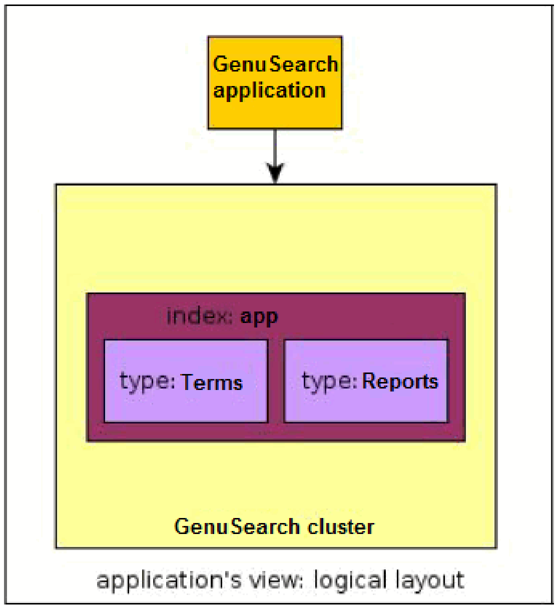


Figure 26. Logical layout

Model	Type	Count
KneeTexOutput	KneeTex	1253
ReportDocument	Reports	100
Term	Terms	1619
Typedef	Typedef	16
Suggestion	Suggestions	273
TermExpansion	TermHierarchy	395

Figure 27. Data statistics

The first chunk of data that gets indexed into the cluster is the TRAK ontology, which is parsed from a text file in OBO format and is iterated over line-by-line until each *term* is mapped and extracted.

The second chunk is the test set of KneeTex data and knee MRI reports that I was given, which consists of 1253 KneeTex findings and 100 MRI reports. Their original storage is a SQLite database, which GenuSearch accesses once in order to retrieve and index the data into its own data storage engine.

Suggestions get indexed third. They are gotten from the KneeTexOutput objects on the system, which means that each of them has at least one search result relevant to them. I could have used the TRAK ontology to create suggestions, but it is difficult to derive all the findings, anatomical entities and different observations from it. Having "empty" suggestions (TRAK terms that do not exists in any of the KneeTexOutput objects on the system) would have been inefficient as well, because it would increase the possibility of zero result search responses.

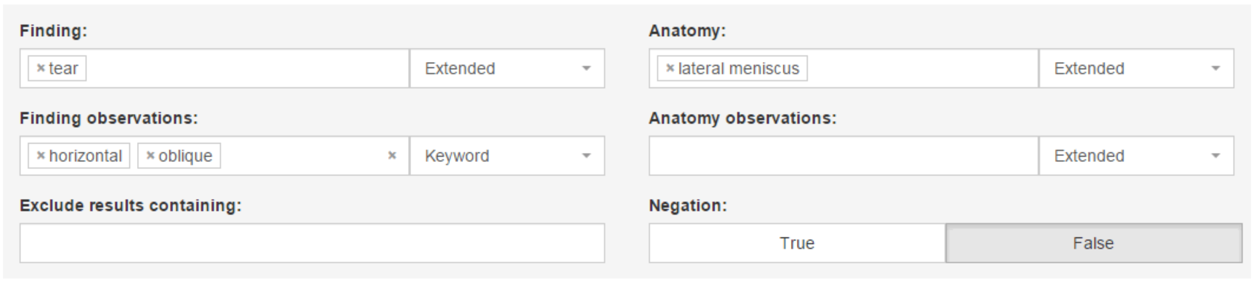
The suggestions extracted from the test set of 1253 KneeTexOutput objects ca be summed up into:

- Finding: 85 unique suggestions
- Finding attributes: 73 unique suggestions
- Anatomy: 61 unique suggestions
- Anatomy attributes: 53 unique suggestions

The model that gets indexed last in the TermExpansion one. It uses the TRAK "is_a" relationship between terms to find all the names of a term's child branches in the ontology. This model is later used for construction queries dependant on the TRAK ontology.

4.4. Searching

Step 1.) After the *Search* button on the main screen is pressed, all the input values and options get parsed with jQuery, and constructed into an easy to interpret object. The object is then transformed into JSON and sent to the *Search Controller* via POST request. If the number of queries is more than one, separate objects are built for each of them. These objects get assigned different keys in the HTTP request, so that the back-end can processes each query individually. A third key is also added, which indicates the method that should be used for combining the search results (disjunction or conjunction).



Finding: × tear — Extended

Anatomy: × lateral meniscus — Extended

Finding observations: × horizontal × oblique — Keyword

Anatomy observations: — Extended

Exclude results containing:

Negation: True [✓ False]

```
{
  "finding": {"value": "tear", "type": "extended"},
  "anatomy": {"value": "lateral meniscus", "type": "extended"},
  "findingAttributes": [
    {"value": "horizontal", "type": "keyword"},
    {"value": "oblique ", "type": "keyword"}
  ],
  "anatomyAttributes": null,
  "exclude": null,
  "negated": false
}
```

Figure 28. Search Query transformation into JSON

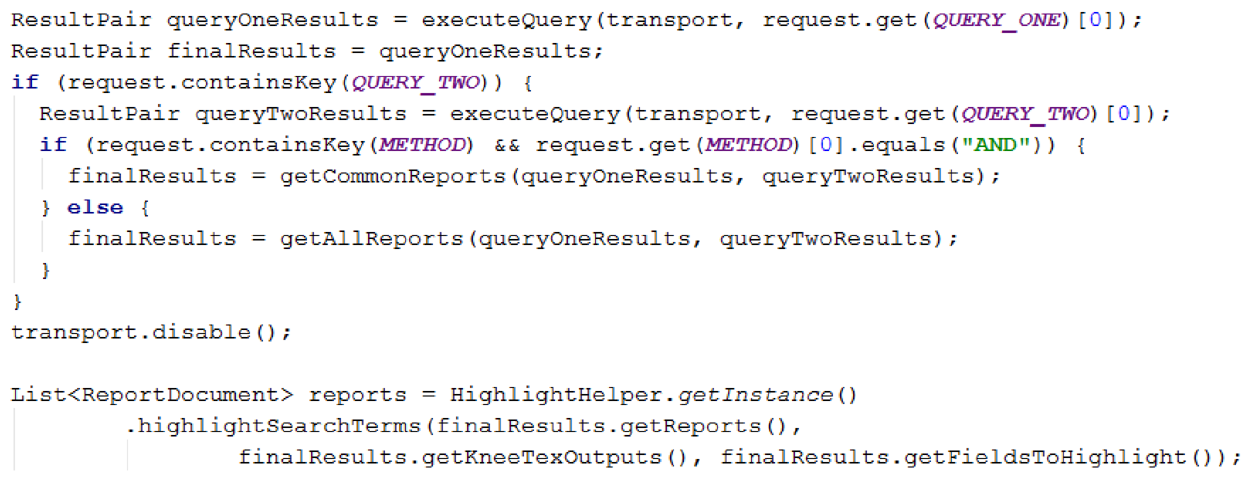
Step 2.) Once the *Search Controller* receives the POST HTTP request from the front-end, it parses each query and send it to the *QueryGenerator* class in the *app/search* package. This class combines multiple small queries, constructed based on the "value" and "type" criteria in each input, into a one big query. This query is then used to retrieve relevant KneeTex objects from Elasticsearch. Once search results are found, the application gets the original MRI document ids from them. This is done because a KneeTex object represents only a single processed sentence from an MRI report (hence the different 100 reports -> 1253 KneeTex objects), but we want to display the entire report.

Step 3.) Handling exclusions

The *Exclude results* input is treated as its own individual query and is processed separately from the other inputs of the overall search query. The application sends the exclude criteria to the QueryGenerator than uses the created Elasticsearch query to look for any KneeTex objects that match it. From those objects the system gets the MRI report document source ids and compares them to those of the positive query. If any matches are found, they are removed from the positive query search results.

Step 4.) Handing search results and Highlighting search terms

KneeTexOutput objects and ReportDocument objects go hand-in-hand while searching, because we are querying KneeTex data, but displaying the entire MRI reports. In *figure 29* you can see a code snippet, where both lists of results (KneeTex and MRI reports) are kept in one single object called ResultPair.



```
ResultPair queryOneResults = executeQuery(transport, request.get(QUERY_ONE)[0]);
ResultPair finalResults = queryOneResults;
if (request.containsKey(QUERY_TWO)) {
  ResultPair queryTwoResults = executeQuery(transport, request.get(QUERY_TWO)[0]);
  if (request.containsKey(METHOD) && request.get(METHOD)[0].equals("AND")) {
    finalResults = getCommonReports(queryOneResults, queryTwoResults);
  } else {
    finalResults = getAllReports(queryOneResults, queryTwoResults);
  }
}
transport.disable();

List<ReportDocument> reports = HighlightHelper.getInstance()
        .highlightSearchTerms(finalResults.getReports(),
                finalResults.getKneeTexOutputs(), finalResults.getFieldsToHighlight());
```

Figure 29. Search controller code for processing a search

Once each individual query has been processed and executed, the system has to decide how to combine the results of the two queries. There are two combination methods – the first is *AND* (conjunction), the second – OR (disjunction). Both methods are made available to the user through the UI. If conjunction is used only MRI report documents that are present in both lists of search results are left. If disjunction is used the lists of search results are merged and displayed to the user. Once combined, results are fed to the highlight engine and sent back to the front-end.

If only a single query was used for searching the application just leaves the MRI reports to be highlighted and sends them back to the frond-end.

5. Results and evaluation

5.1. Testing retrieval

Considering the size of the dataset on the system - 100 MRI Reports, I and my supervisor agreed that the Precision at K documents metric was a suitable choice for testing information retrieval. This metric is also known as a "Precision at 10" metric, which corresponds to the number of relevant results from the first batch of 10. This test is easy to score manually, since only the top K results need to be examined to determine if they are relevant or not.

I ran 10 different queries and examined the first 10 results for each of them. The queries that I used were created based on sample studies of knee conditions.

Study 1: Prevalence of abnormalities in knees detected by MRI in adults without knee osteoarthritis.

(1) Query:

Finding: **irregularity** *(extended)*

Exclude results containing: **osteoarthritis** *(finding)*

Results score:

Total results: **20**

Relevant results from the first batch of 10: **10**

When checking if the exclude worked on the results I used the browser to look for the word osteoarthritis or OA (which is a synonym) and it was not present. I also ran the query without excluding osteoarthritis and the results total increased to 22, which means that this worked successfully.

Study 2: Given the limitations of the medical literature, it is not possible to determine the exact risk of MCL injury by sport. Football and basketball players appear to have a relatively high prevalence, as do participants in contact sports such as wrestling, hockey, and rugby. Whether the risk of MCL injury varies significantly by gender remains unclear.

(2) Query:

Finding: **injury** *(extended)*

Finding observations: **football** *(exact)*

Results score:

Total results: **1**

Relevant results from the <u>first batch of 10:</u> **1**

Unfortunately the test dataset does not contain a lot of reports related to knee injuries that occur in sports. This test still proved to be a success though, even though it had only one result.

Study 3: Acute knee injuries in adults.

(3) Query:

Finding: **injury** *(extended)*

Finding observations: **acute** *(exact)*

Results score:

Total results: **3**

Relevant results from the <u>first batch of 10:</u> **3**

Study 4: EPIDEMIOLOGY — According to one systematic review, studies of the epidemiology of knee injuries are fraught with problems. Nevertheless, ligament injuries account for up to 40 percent of all knee injuries, and of these, medial collateral ligament (MCL) injuries appear to be the most common. MCL tears accounted for 7.9 percent of all injuries in an observational study of 19,530 knee injuries in 17,397 athletes over a 10 year period.

(4) Query:

Finding: **injury** *(extended)*

Anatomy: **ligament** *(keyword)*

Results score:

Total results: **50**

Relevant results from the <u>first batch of 10:</u> **10**

(5) Query:

Finding: **injury** *(extended)*

Anatomy: **medial collateral ligament** *(extended)*

Results score:

Total results: **20**

Relevant results from the first batch of 10: **10**

(6) Query:

Finding: **tear** *(extended)*

Anatomy: **medial collateral ligament** *(extended)*

Results score:

Total results: **7**

Relevant results from the first batch of 10: **7**

Study 5: Horizontal or oblique meniscal tears were found medially in 32 and laterally in 11 symptomatic knees, and medially in 29 and laterally in eight asymptomatic knees. Radial, vertical, complex, or displaced tears were found medially in 18 and laterally in five symptomatic knees, and medially in five and laterally in none of the asymptomatic knees. Collateral ligament abnormalities were found in 53 symptomatic knees and in six asymptomatic knees. Per capsular soft-tissue abnormalities were found in 64 symptomatic and in 12 asymptomatic knees. Edema-like bone marrow abnormalities were found in 36 symptomatic and in three asymptomatic knees.

(7) Query:

Finding: **tear** *(extended)*

Finding observations: **horizontal** | **oblique** *(keywords)*

Anatomy: **lateral meniscus** *(extended)*

Results score:

Total results: **8**

Relevant results from the first batch of 10: **8**

(8) Query:

Finding: **tear** *(extended)*

Finding observations: **horizontal** | **oblique** *(keywords)*

Anatomy: **medial meniscus** *(extended)*

Results score:

Total results: **8**

Relevant results from the <u>first batch of 10:</u> **8**

(9) Query:

Finding: **tear** *(extended)*

Finding observations: **radial** | **vertical** | **complex** | **displaced** *(keywords)*

Anatomy: **lateral meniscus** *(extended)*

Results score:

Total results: **16**

Relevant results from the <u>first batch of 10:</u> **10**

(10) Query:

Finding: **tear** *(extended)*

Finding observations: **radial** | **vertical** | **complex** | **displaced** *(keywords)*

Anatomy: **medial meniscus** *(extended)*

Results score:

Total results: **20**

Relevant results from the <u>first batch of 10:</u> **10**

Overall the system's retrieval appears to be yielding appropriate results. Out of the 5 sample studies and 10 relevant queries all of the checked results were 100% correct. Since all the queries used for testing were singular, I executed a multi-query search combing sample *Query 8* and sample *Query 10*. When combined with conjunction (AND) there were a total of 4 results, each of which was correct. When combined with disjunction (OR) there were 24 search results and out of the first 10, all 10 were correct.

5.2. Usability testing

The usability testing of the application was conducted by five people: 2 computer science students, 1 civil engineering student and 2 mechanical engineering students. A study conducted by Jakob Nielson shows that 80% of the usability findings are yield by not more then 5 participants. Therefore, according to Jakob Nielson the optimal number of participants for most of the usability studies is between 4 - 5 people. Summarising the feedbacks from a smaller group of testers is easier as well.

Each test was done individually and did not require a background in pathology. The single purpose of these test was to highlight subjective user preferences about the application including its general usability, user perception and appropriate wording of content. The part about general usability was based on the System Usability Scale (SUS), a questionnaire for assessing the perceived usability of interactive systems. It consists of 10 questions based on a 5-point Likert scale (1=strongly disagree, 5=strongly agree). In comparison to other commonly used questionnaires, it was shown to be the simplest and most reliable in determining website usability. Not surprisingly, it is the most used questionnaire for measuring perception of usability. The overall SUS score is calculated on a scale from 0 to 100. The widespread usage of the SUS questionnaire allows the usability of a system to be benchmarked against others.

Based on the average SUS score, any score above 68 is considered above average. In my case, the SUS score calculated from user's responses was 65, which belongs to a percentile range of 60-67%. In other words, the application has higher perceived usability than 60% of systems. This rank can be interpreted as "Grade C" on a scale from A to F

6. Future work

6.1. Taking into account tester feedback

Although the overall comments about the system after usability testing were great, it revealed a lot of issues with the application, most of which consist of unclear functionality and design. Some of them were:

- The "Extended" option on the main search screen - people found it hard to understand what it is, others did not even notice that it was there, because it did not have a label above it. This would need to be resolved in the future.
- "Negation" – many people did not understand what exactly negation was meant for, how exactly it is used when searching. A tooltip of some kind would need to be created in order to explain to the users what negation means and how it is used.
- The "save" button was hard to recognise – some of the testers though it was a "search box" at first. This needs to be taken into account and restyled in the future.
- People tried to "view" a search after they deleted it, which led to a white screen with an error message saying "this search does not exist". It would be good to fix this in the future and disable the "view" button together with the "delete" button when a saved search is deleted.

6.2. More data

It would be very interesting to expand the data set, see how the system behaves and try to come up with improvements based on the new data.

7. Conclusions

This project required the accomplishment of six tasks.

The first one was to creating a database that would store:

- the original MRI reports
- the structured KneeTex output data, created from processing the original MRI reports
- the TRAK ontology

This task was completed, with the use of Elasticsearch as the default database for the project. Apart from the TRAK terms, MRI reports and KneeTex outputs, the system also stores saved searches, suggestions and other objects.

The second and third tasks consisted of making a search interface that would retrieve information from the database.

GenuSearch has successfully provides a *Search* Page and a *Search Results* Page, as well as pages for *Saved Searches* and *Tagged Reports* – all linked to Elasticsearch through the controllers of the application.

The forth tasks was to implement a suitable ranking algorithm that would score the relevancy of search results, so that they can be displayed in proper order. This task was automatically handled by Elasticsearch, which ranks search results by default using Lucene.

The fifth task was to test information retrieval. This was done by using the Precision at K metric and gave 100% correctness of results.

The sixth task was to test the system's user interface and draw conclusions from user feedback. Although I did not manage to fully describe the testing stage, the system was formally tested and did not show many functionality flaws. The only major problems that came up were with terminology and wording, which brought up uncertainty to what the functional purpose of certain elements is in the eyes of users.

Overall, GenuSearch is a very robust and simple web-application that can be used to store and query KneeTex data in order to help future epidemiologic studies of knee conditions.

8. Reflection of learning

8.1. Project and time management skills

The biggest mistake I made during this project was to underestimate the scale of it. I started working seriously only halfway through the semester, which meant I fell severely behind on my initial project plan. Although I was able to catch up at the very end, I will try not to do work like this again, and instead accomplish task little by little.

A good habit I developed, that will definitely be useful in the future, is to create a task list for every single day. Although I managed to be on point with my task lists most of the time, there were cases when unexpected problems occurred during implementation and I had to stop and look for solutions, which in its own turn was very time consuming. The made me adapt this type of obstacles and predict their future occurrence.

Overall, I learned to consider and evaluate multiple approaches and tools in order to select the best ones for the needs of the project. I learned how improve my work based on feedback. I learned how to manage my time correctly and to be able to finish my work alongside other commitments in life at the same time. I learned how to document my work in an understandable and clear manner. I learned that good communication, and accepting responsibility is essential for the project success.

8.2. Communication with supervisor

I had regular weekly meetings with my supervisor during which the work done was presented and recommendations were given for future.

8.3. Software engineering skills

My software engineering skills improved drastically during this project, because of the many technologies I used to accomplish its creation. There was a lot of learning involved and a lot of practice as well, so my understanding of and appreciation for web development has grown quite a lot. The main lesson I learned from the software development stage of this project is to "learn by doing". In the beginning, I found myself reading a lot about technologies like Bootstrap and jQuery, but when I started practicing I could not do anything and that led to a lot of disappointments. Instead, I started doing theory and practice in parallel, which helped me start improving faster.

8.4. Problem solving skills

Due to the sheer size of this project new problems were a constant. Sitting down and solving problems every day required a lot of courage and self-belief. Having gone through this project, I have become better at analysing, breaking down and solving large problems.

References

1. Matthew Lee Hinman and Radu Gheorghe. 2014. Elasticsearch in Action. Manning.
2. Irena Spasić, Bo Zhao, Christopher B. Jones and Kate Button. 2015. KneeTex: an ontology–driven system for information extraction from MRI reports [Online]. Available at: http://www.jbiomedsem.com/content/pdf/s13326-015-0033-1.pdf [Accessed: 03-05-2016]
3. Button K; van Deursen RW; Soldatova L; Spasić I. 2013. TRAK ontology: Defining standard care for the rehabilitation of knee conditions. J Biomed Inform.
4. Health & Social Care Information Centre. 2015. Read Codes [Online]. Available at : http://systems.hscic.gov.uk/data/uktc/readcodes [Accessed: 03-05-2016]
5. Herrett E, Thomas SL, Schoonen WM, Smeeth L, Hall AJ. 2010. Validation and validity of diagnoses in the General Practice Research Database: A systematic review. Br J Clin Pharmacol.
6. Rosse C and Mejino JJ. 2003. A reference ontology for biomedical informatics: the Foundational Model of Anatomy. J Biomed Inform.
7. Wikipedia. April 2016. Natural language processing [Online]. Available at: https://en.wikipedia.org/wiki/Natural_language_processing [Accessed: 05-05-2016]
8. Friedman C, Alderson P, Austin J, Cimino J, Johnson S. 1994. A general natural-language text processor for clinical radiology. J Am Med Inform Assoc.
9. Hripcsak G, Austin JH, Alderson PO, Friedman C. 2002. Use of natural language processing to translate clinical information from a database of 889,921 chest radiographic reports. Radiology.
10. Mamlin BW, Heinze DT, McDonald CJ. 2003. Automated extraction and normalization of findings from cancer-related free-text radiology reports. In: Proceedings of the AMIA Annual Symposium.
11. Dang PA, Kalra MK, Blake MA, Schultz TJ, Halpern EF, Dreyer KJ. 2008. Extraction of recommendation features in radiology with natural language processing: exploratory study. Am J Roentgenol.

12. Burnside ES, Davis J, Chhatwal J, Alagoz O, Lindstrom MJ, Geller BM, et al. 2009. Probabilistic computer model developed from clinical data in national mammography database format to classify mammographic findings. Radiology.

13. Cowie J, Lehnert W. 1996. Information extraction. Commun ACM.

14. Wikipedia. May 2016. Web Application [Online]. Available at: https://en.wikipedia.org/wiki/Web_application [Accessed: 05-05-2016]

15. Wikipedia. April 2016. HTML [Online]. Available at: https://en.wikipedia.org/wiki/HTML [Accessed: 05-05-2016]

16. Wikipedia. May 2016. Cascading Style Sheets [Online]. Available at: https://en.wikipedia.org/wiki/Cascading_Style_Sheets [Accessed: 05-05-2016]

17. Wikipedia. April 2016. Bootstrap (front-end framework) [Online]. Available at: https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework) [Accessed: 05-05-2016]

18. Wikipedia. April 2016. Java (programming language) [Online]. Available at: https://en.wikipedia.org/wiki/Java_(programming_language) [Accessed: 05-05-2016]