Feed-me!: Automatic Identification and Crawling of Event Feeds from the Web and Social Media

Punpikorn Rattanawirojkul

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

A dissertation presented in part fulfillment of the requirements of the Degree of Master of Science at the University of Glasgow

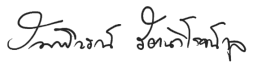
8th September 2018

Abstract

Information are now available in the electronic format which makes records, like news articles, are now accessible online. News agencies are not the only sources of news, social networks also play an important role in the world of the Internet. There are many specific business areas that requires a lot of data. For example, the stock trading. On the other hand, news agencies themselves need data for their publishing and disseminating. However, gathering the information as much as possible of particular events from different sources together could be difficult and ineffective. Sources do not provide all data for free, offering only a limited quota. Information from different sources are not in the same format and the approach to access is also varying. On the other hand, the content of the event is changing overtime. For this reason, all aspects of that event may not be captured. Feed-me! is the event information aggregation web application that aims to aggregate information from different sources together as much as possible. To aggregate data from different sources, it lets users specify how to collect the data from a particular source by providing the mapping information from raw responses to a desire format of record, an authentication approach, including the period of crawling that allows the system to frequently get and update more records. It also features with query expansion, expanding from related data of a given query to another new query. Users could manually perform an expansion with the help of information provided by the system or let the system automatically perform an expansion periodically according to the given configuration. In addition, two graphs are available to let users gain insight from related records of a specific query. Bar graph shows the number of records from each source in timeline fashion while timeline graph explains the relationships between queries-records and queries-queries (expansion) in node-edge graph. Users can interact with the system via web pages that allow them to query for a specific event, see the result and visualisation of a query, track the existing queries, manage crawlers and manage query expansion. The project was evaluated using the software testing and the user evaluation. Software testing tests the correctness of core operations while the user evaluation evaluates overall functional requirements. For the user evaluation, evaluators will be asked to complete a set of tasks while the author records completion time and their suggestions. According to the evaluation result, query expansion is feasible and would be worked in practice. However, user interfaces improvement, especially indications and guides, is needed to help users understand the system without training sessions. Overall, the evaluation is successful by having more than 80% of passed evaluators and more than 80% of passed criteria in the questionnaire rating section.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic form.

Name: Punpikorn Rattanawirojkul Signature: 

Acknowledgements

I am particularly graceful for the assistance, support and solution to the problems given by Prof. Iadh Ounis and Dr. Richard McCreadie, my dissertation supervisors from the school of computing science, University of Glasgow. I wish to acknowledge the help provided by the evaluators of this project for their precious time, opinions and suggestions toward the project. Many thanks to Twitter and News API for such wonderful main data sources of this project. I wish to thank my friends and my parents for the support and encouragement throughout my study. Finally, I am honored to obtain an amazing opportunity for a study and a scholarship from the University of Glasgow.

Contents

Chapter 1 Introduction

1.1 Background and challenges

Nowadays, information is being published over the Internet in a digital form. The advantage of publishing information through Internet is a short distribution time. For example, we could see online news about the natural disaster published right after the incident happened. A notable example that follows this approach is news agencies. Previously, news was distributed through newspapers and television channels. However, agencies now currently have their own websites to persistently keep and distribute online news regarding to the increasing of information consumption through online medias. There are a lot of news sources, not only online news agencies, but also social networks where information is mostly published by ordinary people. Publishing information via social networks is faster compared with news agencies because of fewer steps made before the publishing, resulted in higher chance of being false information. News agencies have to look for events or information that could be a news first and verify the correctness then go through production steps further on. Information would spread from news sources through different channels when a single event is taking place. Developers or people who have an interest on data could get an information related to a specific event using web APIs (Application Programming Interface) provided by online news sources. News APIs usually come in 2 types. REST APIs provide access to past *records*, such as news article or tweet, which is collected in the provider's databases while streaming APIs provide access to a particular portion of real time records. Twitter is an example of a provider that has streaming APIs. However, they do not offer all of this data for free. News providers typically offer restricted rate limits, limited number of records in a response and few options to filter the records. Users have to pay if they need higher rates or better services and support. There are specific business areas that need lots of data, such as stock trading. Market data assets the decision of traders, especially day traders where real-time market data is impactful [1]. On the other hand, news providers themselves are organizations who need high amount of data. They need to track information emerging from different sources and publish as their content or disseminate to other needed clients. Bloomberg L.P is a company that provides financial news, media, software and data [2]. There are 5000 of stories from 120 countries published daily from the company's private news agency and about 325,000 clients subscript to the company professional service called Bloomberg Terminal [3]. Unimaginable amount of data has been ingested within the firm. Therefore, the solution to reduce the expenses on APIs, of a single provider, and to get lots of data as much as possible is to aggregate different information from different sources together which could be difficult due to variants of record formats and methods of access data.

There are many challenges to aggregate records together. Since they naturally came from different sources, approaches to get records from APIs could be different too. Clients need to authenticate themselves before using the API services. There are different authentication types available such as API Key and OAuth which also require different key information too. Responses from the web APIs are usually in JSON (JavaScript Object Notation) format. However, each source has its own object structure. Mapping from different structures of responses to the same structure is necessary to collect those records together in the same database collection.

Information from a particular event is continually published from the beginning of the event. So, records gathered at the beginning may not represent whole event. It is

important to frequently *crawl*, to get and update new records, more content over time in order to get all related data and better understand the nature of an event.

Content of an event changes from time to time. At the beginning, event like a natural disaster, articles portray destruction of housing and surrounding environment. After that, focus might move to the rescues. It is also possible to have sub-events such as actions of other countries or celebrities related to the event. As you can see, the event comprises of many interesting aspects. It may not be possible to cover every element of the event if *query*, terms that represent the event, is static. For instance, "cave thailand" is a query representing an event of Thai soccer team rescue. News related to other aspects like the help from Elon Musk and cheers from famous soccer players may not be included from the tracking of query "cave thailand" since the query gives overall information.

Humans understand information better with the help of visualisation. There could be insight within the relationships between records and queries. For example, it is interesting to see why a record is related to several queries. That record may give you the idea to generate a query to get more information related to that event.

Overall, identified problems are restrictions from free APIs, differences of APIs formats and accesses, the needs of frequent crawling, changes of content over time and lack of visualisation that improve human understanding. The project represents approaches to tackle these problems in the following section.

1.2 Solution

Feed-Me! is an event information aggregation web application powered by configurable *crawlers* (a unit that perform crawling from a specific source) with query expansion feature. A system main objective is to gather as much information as possible from given queries. It tackles the differences between news sources by providing an interface to define how to collect and map responses and authenticate API requests with key information to access API of a specific source. This enables the application to aggregate data from different sources together. The aforementioned configurations are called *source* information, where user can also enter the crawling period that let the system automatically schedule a *crawling,* getting and updating more records from source API, on tracked queries periodically. Moreover, the application provides several operations to view and manipulate collected data. User can browse through collected records and tracked queries, see visualisation of a query, manage crawlers, and manage query expansion. *Query expansion* is another main feature. "Expand" is to establish a link from one query to another query assuming that 2 queries are related in a particular aspect. This mechanism helps the system adapting to the changes of an event's content and get more related data.

1.3 Structure

In the following section, we provide comparisons between similar systems and give brief overview of tools and libraries used in the development. Functional and non-functional requirements including how they are gathered are listed, prioritized by MoSCoW method, and indicated with implementation status in Section 3. Section 4 shows overall software architecture diagrams with the explanation of components and interactions also the UI storyboard. Section 5 explains system components in detail of how it was built, challenges and solution to overcome those. Detail of the software testing and user evaluation take place in Section 6. Finally, conclusions and future works are discussed in Section 7.

Chapter 2 Related Works and Tools

This section discussed about the related works, and tools that were used for the project. Section 2.1 describes the related works while Section 2.2 explained the related Tools.

2.1 Related Works

2.1.1 Query Expansion

In the area of Information Retrieval (IR), the general definition for a query expansion is the query reformation techniques that improves the performance of IR operations, while the expansion is specifically expanding a query using user's input (including terms in the query and other data) to get more related data in the search engines context [4].

The authors of the book stated that, in the query expansion, users could give an additional input by suggesting terms for a query, especially used in Web [5]. For example, the web suggests users to add terms like "Trump" or "Duck" when querying for "Donald". Additionally, the expansion could be *interactive or non-interactive*, the previous example is interactive since it lets users choose the term. *Automaticity* is also considered; the previous example is manual since users are responsible to choose the word. They remarked that the main problem for the query expansion is an approach to generate these suggested terms or an expanded query for users and suggested that there are commonly 2 approaches which are *global analysis* or *local analysis*. Global analysis corporates the use of a specific thesaurus, it expands the query by analysing each term and automatically adds the related term from the thesaurus. Local analysis expands a query by analysing related documents of the query instead.

For the local analysis, getting the related documents of a query requires the asset of document relevancy judgement. Users could give a relevancy directly to a result document, called *explicit relevance feedback*. However, collecting such relevancy is difficult because users need to mark all documents which is not practical. Therefore, automating the relevancy judgement of documents might be a more practical solution, called *pseudo relevance feedback* [6]. For each querying, the top "k" ranked documents in a set of result documents are assumed to be relevant. Then, the system analyses these documents and selects top ranked terms from the documents using the weighting techniques such as tf-idf (term frequency-inverse document frequency). These top ranked terms will be used as suggestion terms in the query expansion.

The query expansion approach of the dissertation expands a query, after the analysis, by *creating another query*. It does *not change* or *improve the first query*, however generating a new query to collect more relevant data. The query expansion of the dissertation performs a local analysis using the pseudo relevance feedback because of its practicality. More detail of the query expansion approach including the interactivity and automaticity will be explained in Chapter 4, in Section 4.3.

2.1.2 Sover!

Sover! is a system that monitors events real-time in Twitter using a technique called adaptive crawling and represents them in a feature-rich dashboard [7]. The adaptive crawler fetches data from the Twitter streaming API via 2 streams. First is the *filter stream*, which is the system main input, that filters tweets according to filter parameters. Another is the *sample stream* that simply provides 1 percent of the global real-time tweet stream. The system analyses tweets from the filter stream to get and

track new *query keywords*, keywords that are used as search queries, and eliminates *noise keywords,* keywords that coming from other events, with the help of the sample stream. Dashboard provides content from different sources. It performs URL unshortening on tweets, groups them by sources and shows them in the dashboard. The dashboard also shows images and videos according to the source of a tweet.

The key differences between Sover! and Feed-Me! are listed in table 1.

Criteria	**Sover!**	**Feed-me!**
Goal	Real-time event monitoring with adaptive crawling	Event monitoring and information aggregation from different sources with query expansion
Data source	Mainly from the Twitter and enriched by the source of tweets	Any sources that provide API with supported authentications (OAuth and API Key)
Crawler	A crawler is specific to a seed keywords and parameter setting and keyword is updated and added from the adaptation feature.	A crawler is specific to a source, crawling new records for every tracked query.
Adaptation feature	Adaptive via adaptive crawling by analysing tweets from the filter stream to get new query keywords and eliminate noise keywords with the help of the sample stream	Adaptive via query expansion by analysing related records of tracked queries using any text analysis techniques (provide interface to let users implement their own technique) to get new expanded queries
Assistance from human	Manually adding keywords to a crawler	Manually perform the query expansion with the aid of statistical information generated by the system
Dashboard	Rich with content from different sources including videos and images	Query result page shows the related records, in text, of a specific query allowed user to filter and delete records
Visualisation	Histogram of popular hashtags and users in dashboard	2 visualisations: Bar graph shows the number of related records from each source of a specific query and timeline graph shows the relationship between queries-records and queries-queries (expansion)

Table 1: Key differences of Sover! And Feed-me!

2.2 Related Tools

2.2.1 Web Application Frameworks

Play: Play is a modern web application framework for Scala and Java [8]. Play adopts Model-View-Controller (MVC) architectural pattern. It has 3 components. Model (M) is responsible for the data management which accepts parsed inputs from the controller. View (V) is a user interface which is a representation of model. Controller (C) is a middleman that control data flows between view and model [9]. Implementation using Play is a Must-have requirement from the beginning.

2.2.2 JavaScript Framework

The JavaScript frameworks offer powerful tools that enhances the frontend web development. They provide methods to efficiently manipulate DOM (Document Object Model), elements that build up a web document. According to Matt Burgess, the purpose of these frameworks is *"to map the application state to the DOM*" [10]. The following is the list of related tools that have been considered for the development.

jQuery: According to its official website, it is a simple, fast and light-weight JavaScript framework providing utility tools for DOM manipulation, animations, making asynchronous request and etc. [11].

Vue.js: Vue is a progressive framework for web view development which is easy to integrate with other libraries [12].

The author decided to use Vue as a main implementation of views for several reasons. First, it was a good chance to learn this new and popular JavaScript framework. It demonstrates a new style of web application development since the author still had an experience on the old era of the development using just jQuery. Another reason is that the author has bad experience on jQuery, having used it during the team project last semester. Developing using pure jQuery is not appropriate with complex web application. The codes are very long and hard to refactor, unlike Vue which highly adopts the concept of code reusability. However, jQuery is still required to be imported for some frameworks such as Bootstrap.

2.2.3 Frontend Frameworks

Frontend frameworks provide tools to enhance look and feel of the web application. The author decided to use **Bootstrap** because of prior experiences and its simplicity. Bootstrap is a HTML, JavaScript and CSS framework that provides lots of nice reusable components such as navigational bar, jumbotron and etc. [13]. It also offers various styles of existing components such as buttons and give a way to customize them easily. Since the framework is very popular, there is a customized version for Vue called **Bootstrap Vue**. Bootstrap Vue integrates Bootstrap components with Vue. It increases the ease of calling and controlling Bootstrap components within Vue codes. The author used both frameworks on this project but depending on the situation considered by the implementation simplicity.

2.2.4 Visualisation frameworks

The development of a complex component such as visualisation would be difficult without helps from frameworks. According to the design of visualisation in Chapter 4, the system will have two graphs which are bar graph and timeline graph. Bar graph shows number of related records from each source of a query and is developed using **Amchart**. Amchart offers many types of rich-features chart, including bar graph. Additional features are added to the graph to enhance data browsing such as a zoomable and scrollable chart [14]. Timeline graph is a node-edge graph that reveals the relationships between records-queries and queries-queries, showing the expansions from a query to others. It is not an ordinary graph which is needed to develop by the deep-to-fundamental framework like **D3.js**. D3.js allows developers to create visualisation from the core concept, by binding data with DOM [15]. It is flexible because the developers are not bound into any pre-implemented visualisation features, instead they need to develop their own visualisations by manipulating DOM. In timeline graph, nodes and links were built by D3.js.

2.2.5 News APIs

There are various of APIs for news. However, not all of them are free to use and even free one offers only a limited rate limit. **Twitter APIs** is very popular and offers free standard APIs for developers. There are many APIs available, but the system only uses the search tweets API. Rate limit for the API that authenticates with Twitter user account is limited to 180 requests per 15-minutes window which is enough for the system [16]. However, news from news agencies are necessary too. **News API** offers free API for getting information of a news from over 30000 news sources [17]. Information includes the headline, short description, authors, published date and etc. The rate limit for non-commercial project is 1000 requests per day which is plenty to satisfy the requests made for this project [18]. Twitter and News API are the main data sources for the system crawler. Especially, News API is used for getting articles from different sources such as BBC and CNN.

2.2.6 Database

The author and supervisors agreed to use NoSQL database because we could have a lot of records and queries in the future. With NoSQL, the system could be easily scalable. However, Play does not provide native APIs for NoSQL, unlike SQL. After some studies, the author found that **MongoDB** is the only NoSQL database listed in the Play official driver list and decided to use MongoDB through the driver called **ReactiveMongo** [10]. MongoDB keeps and represents data as a document, in JSON-like format [20]. ReactiveMongo is the well documented and has an active community, compared with other MongoDB drivers. It also supports Scala. The main features are asynchronicity and non-blocking operations [21]. The system uses Reactivemongo widely in every part that need to communicate with the database.

Chapter 3 **Requirements**

The chapter starts with how we gather the requirements in Section 3.1. Then, the definition of user is defined in Section 3.2. Section 3.3 shows the list of discovered requirements identified as functional or non-functional, categorized by MoSCoW method followed with the implementation status.

3.1 Requirements Gathering

Requirements were gathered from the weekly meeting in the form of ideas and discussions given to the deliverables of that week by supervisors. The deliverables could be ideas from the author or pieces of implemented requirement that have been developed during the week. The author normally gave the presentation on deliverable at the beginning of the meeting. The ideas and discussions on deliverables given by supervisors are mostly refinements on that deliverable, which could be extracted into requirements. After the meeting, the author transformed these ideas into requirements, identified as functional or non-functional and categorized by the MoSCoW method. MoSCoW categories helped ordering the features by labeled the priority on them, beginning with Must-have (M), Should-have (S), Could-have (C) and Won't-have-this-time (W) feature. To be more specific, M requirements are stated at the beginning in the introduction slides of 1st week while other categories were discovered in the weekly meeting of following weeks. The author then decided to choose a task to design and implement considering these criteria, for instance picking M requirements first.

3.2 User Definition

The supervisors of this project are *real users* for the system, they actually use this system in the future. However, the general definition of user is still needed to give the idea of target group. Users in our system are people who have an interest in the event data collection. However, users are recommended to have a certain level of programming skill because several main features require knowledge of web technologies and the understanding of how features work. For an instance, users need to specify technical information, such as JSON structure and API URL, when adding a crawler. One of the important attributes is a mapping information from a raw response to mapped records, which represented in JSON structure. In addition, the detail of an API like the authentication, an API URL and the access attribute are needed too. Another example is a query expansion feature that allows users to compose their own query expansion technique. But, they should understand how each component of the query expansion works and what will be the result of those. Overall, users without programming background could find that it is difficult to understand and use the system. Additionally, even evaluators where everyone is a programmer, found that the system was hard to use if there is no detailed guides and tutorials.

3.3 Requirements List

Functional requirements will be written in a format *"<who> can <what>"* while there is no writing format for non-functional requirement. MoSCoW method were applied in order to rank requirements in term of priority. Discovered requirements will be grouped into section by features, identified as functional or non-functional, categorized into MoSCoW categories, and indicated with the implementation status in table 2. Description of ranking criteria is in table 3 and summary of requirements grouped in MoSCoW categories are in table 4.

No.	Functional/ Non-functional	Description	MoSCoW Category	Imple-mented
A. Query and records				
1	Functional	User can query for records from sources	M	Yes
2	Functional	User can see information of tracked queries	M	Yes
3	Functional	User can delete a query	M	Yes
4	Functional	User can check if a query is tracked in the system	S	Yes
5	Functional	User can turn specific crawler(s) of a query on/off	S	Yes
6	Functional	User can see statistical information of a query	S	Yes
7	Functional	User can group queries together in term of expansion	C	No
8	Functional	User can export selective data of a query	W	No
9	Functional	User can filter out duplicate records	C	No
10	Functional	User can query for multiple queries using semantic keywords like and/or	W	No
11	Non-functional	System can link similar records together	C	No
12	Non-functional	System can keep related images from a record in the database	W	No
13	Functional	User can delete a record	S	Yes
B. Crawler				
1	Functional	User can add a crawler	M	Yes
2	Functional	User can delete a crawler	M	Yes
3	Functional	User can edit a crawler	M	Yes
4	Functional	User can see information of a crawler	M	Yes
5	Functional	User can force a crawler to perform crawling before a schedule	S	Yes
6	Non-functional	System can schedule a task for a crawler periodically	M	Yes
7	Non-functional	System can handle different API authentications	S	Yes
8	Non-functional	System can handle limited number of API rate limit efficiently	C	No
C. Visualisation				
1	Functional	User can see visualisations of a query statistic information	M	Yes
2	Functional	User can see relationships between records and queries	M	Yes

3	Functional	User can see relationships between queries and expanded queries	M	Yes
D. Query expansion				
1	Functional	User can perform query expansion on a query	M	Yes
2	Non-functional	System can perform query expansion on a query	M	Yes
3	Functional	User can control how system will perform query expansion	M	Yes
E. Other				
1	Non-functional	System must be implemented using Play framework	M	Yes
2	Non-functional	System must aggregate records from different sources together	M	Yes
3	Non-functional	System should keep raw responses in the database	S	Yes

Table 2: List of requirements grouped by features, identified as function or non-functional, categorised by MoSCoW categories and indicated with the implementation status

Category	**Criterion**
(M) Must have	The requirement corresponds with main goals and is obviously stated at initial meeting. The system will be incomplete if this is missing.
(S) Should have	The requirement is derived from suggestion during weekly meeting and significantly improves user experience of must-have requirements if implemented.
(C) Could have	The requirement is derived from suggestion during weekly meeting. It could improve user experience of must-have requirements if implemented but is estimated to spend effort excessively compared with should-have requirements.
(W) Won't have this time	The requirement is derived from suggestion during weekly meeting and is considered as optional because it is out of the project scope. Additionally, it could be relatively hard to implement.

Table 3: Criteria for categories of MoSCoW

Category	**List of requirements**	**Total requirements**	**Implemented**
(M) Must have	A1, A2, A3, B1, B2, B3, B4, B6, C1, C2, C3, D1, D2, D3, E1, E2	16	16
(S) Should have	A4, A5, A6, A13, B5, B7, E3	7	7
(C) Could have	A7, A9, A11, B8	4	0
(W) Won't have this time	A8, A10, A12	3	0

Table 4: Summary of requirements grouped in MoSCoW categories

Chapter 4 **Design**

Designs of the system will be discussed within this chapter. Section 4.1 shows the high-level system diagram derived from related tools in Section 2.2. System design in the aspect of MVC architectural pattern is explained in Section 4.2. Database schema will be shown in Section 4.2.1 under model while storyboard UI will be shown in Section 4.2.2 under view. Finally, the system diagram in term of MVC is displayed in Section 4.2.3 under controller.

4.1 Overall System Design

The diagram in Figure 1 explains high-level relationships between each related tool, from Section 2.2, in term of components that will be implemented, including the interactions of a user.

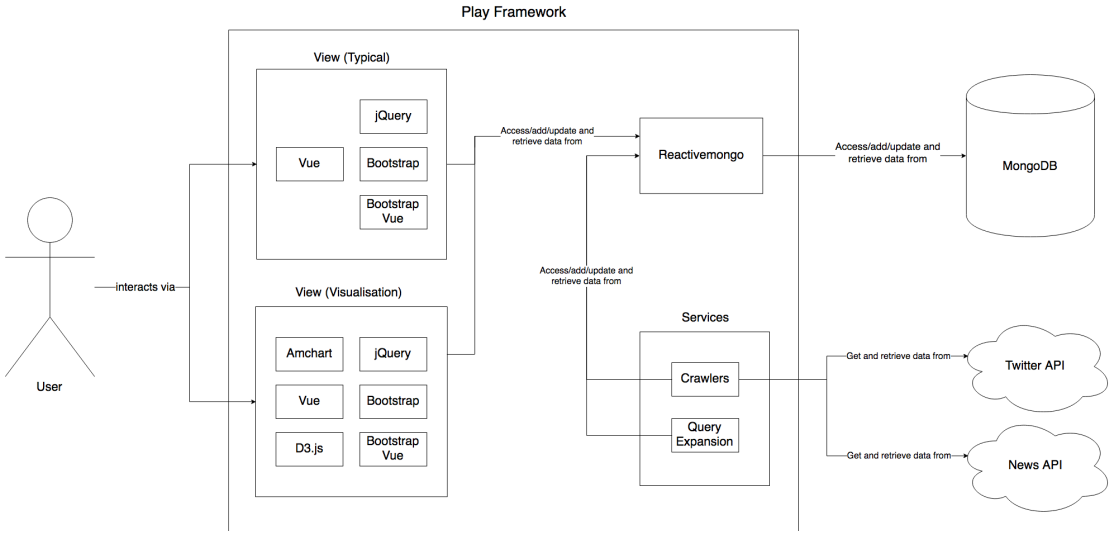


Figure 1: System diagram showing related tools, interacted with a user

Considering the requirement E1, the project must be implemented using Play framework. Play has a model-view-controller (MVC) architectural pattern as stated in Section 2.2.1. Section 4.2 describes the system design in 3 aspects according to MVC components.

4.2 Model-View-Controller (MVC)

4.2.1 Model

Model design is focused on how it will be kept and represented in the database. There is also a scenario when a model is not actually kept in database, source and query expansion unit are wrapper for system services. Figure 2 explains the detail of all attributes and interactions of models via the database schema. Concepts that could be extracted into models, highlighted attributes and ideas behind the model are described as follows.

SearchQuery: Query could be redundant in many concepts, specifically *query parameter* in the APIs. So, the term `SearchQuery` would be more explicit. Other than the text and typical model attributes, it includes previous queries (`prevQueries`) and expanded queries (`expQueries`) to conform the query expansion requirements (group D). The advantage is that it could be represented in graph, nodes and edges, later on in the visualisation part. The query is also designed to be independent of crawlers

through `EnabledSources` attribute. A query may be specific to particular crawlers, such as a hashtag query which is specific to the Twitter.

Record: Record is a model of the information unit in the system, such as news articles or tweets. There is no single static format of records because they come from different sources. However, they still have some attributes in common such as the text, created time, URL and etc. On the other hand, there are also attributes that are specific to the source. For example, hashtags, tweet ID and user ID are specific to Twitter while records from news agencies have authors and news source name. Additionally, the system adds other necessary attributes such as hash and database ID.

Source: A source holds information of a particular source for a crawler. It is special because it is not saved into database, it is saved in a configuration file instead. Users need to specify 2 types of collection names. `collectionName` is the name of collection where mapped records will be kept while `rawCollectionName` is the name of collection that collects raw records, satisfy the non-functional requirement E3. According to the requirement B7, the system supports 2 authentication types which are APIKey and OAuth. Information for such authentications, such as API key for the first type, is kept as JSON in a source model. The advantage is that it supports any authentication types, assuming that all information is correct and corresponding to the type. `APIFormat` specifies where the system needs to insert a query and authentication information in the API URL, depending on the authentication type. Response from API is in JSON and the system might need to access specific attribute name to get actual records, which is the responsibility for `recordAccess`. The most important attribute for a source is a `recordMapping`. It provides a way to map attributes from raw responses to mapped records which will be saved into the database collection (`collectionName`). There are few mandatory fields which are `text`, `title`, `createdTimestamp` and source `url`. Title could be "none", in case of tweets, and user could give a string pattern for source url if it is not available directly in a raw response. Example of a mapping is shown in Figure 6 in Section 5.2.

QueryExpansion (Unit): It is a wrapper for query expansion components. Query expansion is designed to be flexible in order to satisfy the requirement D3. Basically, the expansion has 2 main components which are generator and selector and will be described in detail in Chapter 5. The unit is a reference of the generator and selector that users chose and it will be used by the query expansion service. The model is not saved into the database.

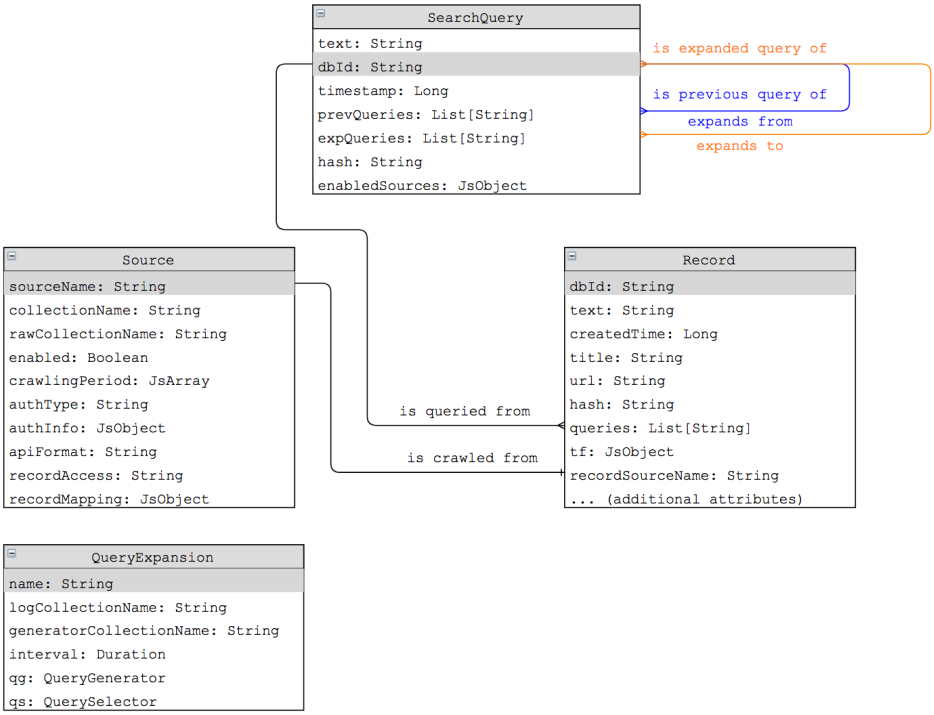


Figure 2: Database schema for models

4.2.2 View

View is simply a user interface representing the model, or the interface that receives data from user. A main goal of views is to let users interact with the system smoothly. The author adopted several core concepts when designing views for the system.

1. User interfaces should be easy to use, simple and to the point
2. Minimalism is a must, no excess decoration
3. Information should be displayed in the appropriate user interface components, such as cards and modals

The first views design was introduced on the 4th week, shown in an Appendix A. It comprises of index, result, track query, and manage crawler page. However, there were important changes listed as follows.

- ***Card layout:*** There are different cards in the design and each of them packs information of the model and binds operations with buttons in the card. The card layouts in initial design were changed due to the difficulties of an actual implementation. To be specific, most components on right side were moved to the bottom instead.
- ***Query setting:*** During the implementation of the requirement A5, the author found that users might not remember which queries were tracked in the system and especially their current query settings of which crawler is enabled or disabled.
- ***Manage QE page:*** Query Expansion (QE in short) is designed after the 5th week. At that time, we still didn't know how the feature is working and the appearance of UI was still unknown. It was later designed on the 8th week.

Overall, the ideas behind current designs for each view are listed as follows.

- ***Index page:*** It is the first view of the system. Basically, users can query and check for the existence of a query in the database if they cannot remember it. By checking the query, a previous query setting will be automatically applied

to the current query setting if the query is existed. In other views, a check button is located at top-right button group besides search bar ("C" button in the storyboard UI)

- ***Result page:*** The list of related records associated with the submitted query is listed here, displaying each in a card layout. This page allows users to browse the records, see more information, delete irrelevant records, filter and order the records. Submitting a query via top-right search bar will redirect the user to this page.
- ***Track query page:*** The view shows information of each tracked query in a card. User can delete queries and change query setting of a query via switch-like buttons. Users perform manual expansion in this page through the expand query modal. Information from each QE unit that showed in the modal also helps users to make decision on manual expansion. A generate now button is there to let users force the system to generate information for them immediately, in case of the system was just started and the automatic expansion has not been performed yet. Clicking at the visualisation button navigates users to the visualisation page of that query.
- ***Visualisation page:*** This page shows visualisations of a specific query. Users can switch between visualisations by clicking the tabs under the query text.
- ***Manage crawler page:*** Crawlers are displayed in this page as a form of cards. Users can edit the information of a crawler (source information), add new crawler via the add/edit crawler modal and turning a crawler on/off via a switch-like button. By clicking the crawl button on a crawler card, the system forces a crawler to start before its schedule time and crawling result, number of new records and total records, will be shown in the modal.
- ***Manage QE page:*** QE units are listed in cards. A card shows information of the QE unit including the expand and logs buttons. Users can force the system to perform an automatic expansion immediately before the schedule. After the expansion is completed, the expand summary will be shown in a modal. Clicking at logs button allows user to see the previous automatic expansions logs.

Figure 3 shows the latest storyboard UI of the system, comprises of pages design, interactions flow and main components design. Bold texts are the *main pages* (or *main views*), which are index, result, manage crawler, manage QE and track query pages. Bold arrow lines represent main page navigations where each page can navigate to any main pages from the navigation buttons on the left or top panel. The exception is for a visualisation page, which is not a main page, but navigation buttons are available. Dotted lines indicate a magnification of the component in that page, such as the layout of cards. Normal arrow lines are interactions after triggering a button, the annotations are explained in the following list:

A1: Submitting a query by clicking a search button redirects to the result page
A2: Check if query is existed in the database by clicking check button
A3: See more record information by clicking a more info button
A4: Delete a record, removing a card from the record cards list
B1: Show the add/edit crawler modal after clicking an add crawler button
B2: Show the add/edit crawler modal after clicking an edit crawler button
B3: Force a crawler to start before a schedule by clicking a crawl button then a crawl result modal will be shown
B4: Delete a crawler, removing a card from the crawler cards list
C1: Force a QE unit to start before a schedule by clicking expand button then an expand result modal will be shown
C2: Expand logs modal will be shown after clicking a logs button

D1: Clicking an expand button lets user perform a manual query expansion via the expand query modal
D2: Clicking a visualisation button in a query card navigates user to the visualisation page of that query
D3: Delete a query, removing a card from the query cards list

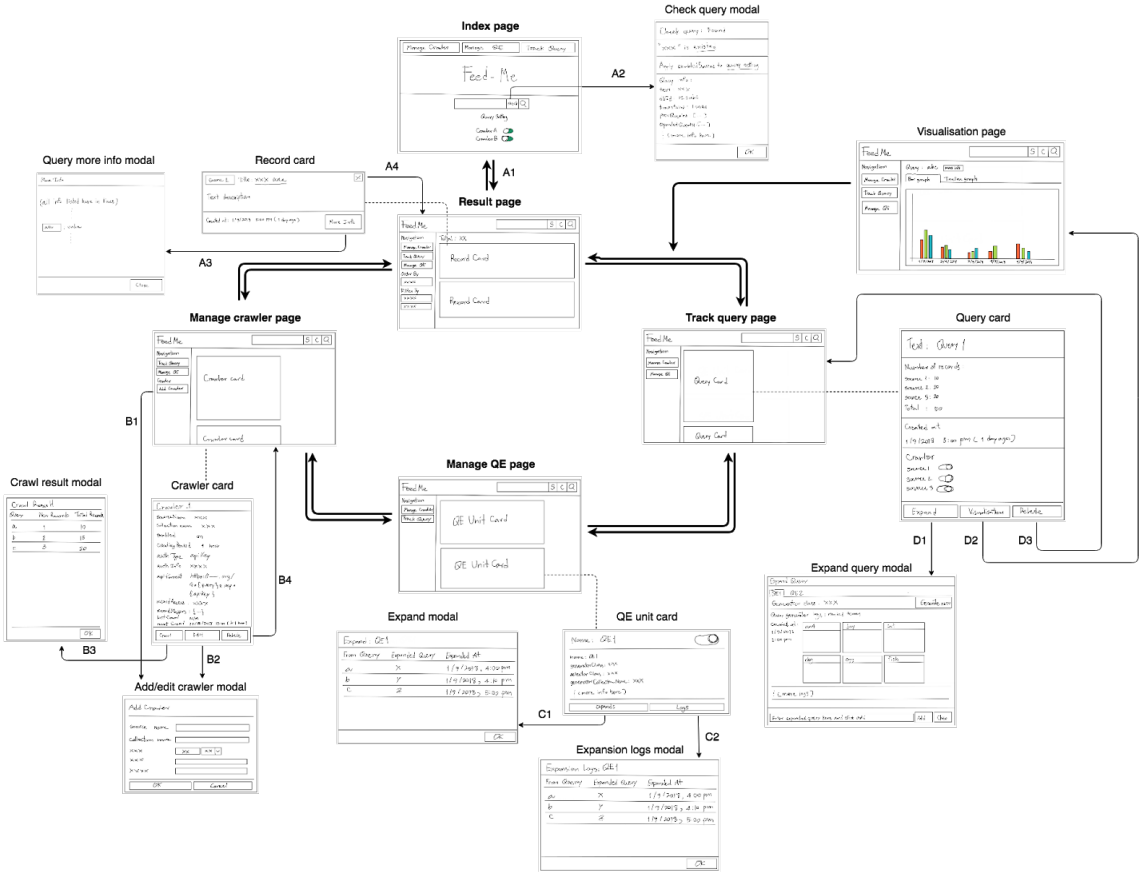


Figure 3: Storyboard UI

Another focus of the system is the visualisation. The aim of visualisation is to provide the insight into a query that cannot be achieved by simply looking at related records of a query. There are 2 graphs, which are bar graph and timeline graph.

- ***Bar graph:*** A query could be related with thousands of records from different point of time. The idea behind bar graph is to summarize the number of records from each source in timeline fashion.
- ***Timeline graph:*** Relationship between queries and records is interesting. We need to represent those relationships somehow in term of the query expansion. In other words, showing how queries are expanded. Also, there could be some particular records that have been crawled many times from different queries, these records link with many queries. This kind of scenario is important for records as it could be considered like gaining more reputation from queries. The goal of this graph is to show relationships between queries-records and queries-queries (query expansion). Initial design of the graph is shown in Figure 4.

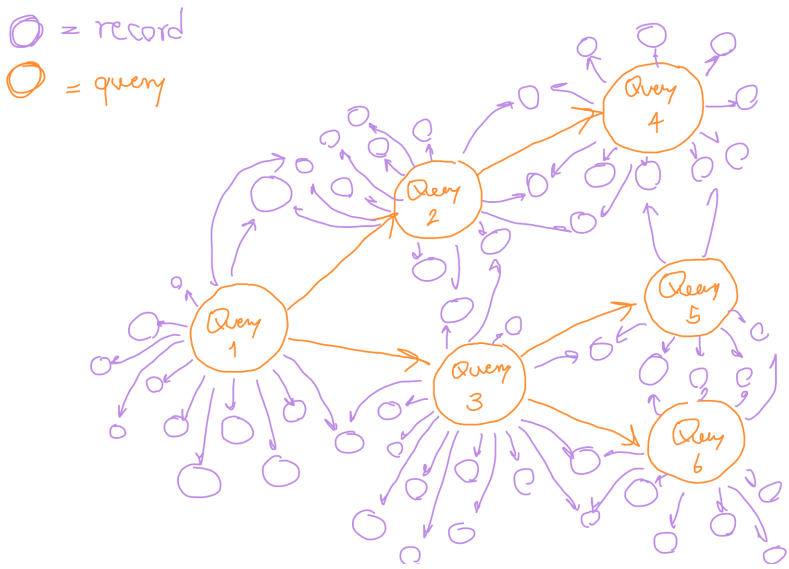


Figure 4: Initial timeline graph design

4.2.3 Controller

A controller manages the data flow from views to models and also interacts with other components in order to perform operations on the input data. The system basically has a single controller for a main view. It mainly handles interactions, or requests, from users in that particular view. For example, the index page controller has to handle a query request by parsing the input query text to a search query (model) and save it to the database. Visualisation page also has a controller since it needs to process and transform lots of data for graphs before returning back to users, even it is not a main view. There is a special controller called `DbController`. During the implementation, the author found that code chunks that are responsible for the database connection were very long and redundant. So, those chunks were refactored to a controller in order to maintain the cleanliness. The summary diagram of system interactions including models, views, controllers and database is in a Figure 5. The relationships between each controller and model(s) are explained as follows. Again, please note that DB controller handles all connections with the database.

Result controller

- SearchQuery: composes a query from the text input and save it to the database
- Record: retrieves records from the database and displays in the result page

Manage crawler controller

- Source: displays information of the crawler, edits, adds and deletes a crawler by manipulating source model

Manage QE controller

- QE Unit: shows QE units information, including turning on/off

Track query controller

- SearchQuery: lists all queries from the database
- QE Unit: shows the information of each QE Unit in the manual expand modal

Visualisation controller

- SearchQuery: gets the information of a specific query from the database
- Records: finds related records of a query and process those records into graphs

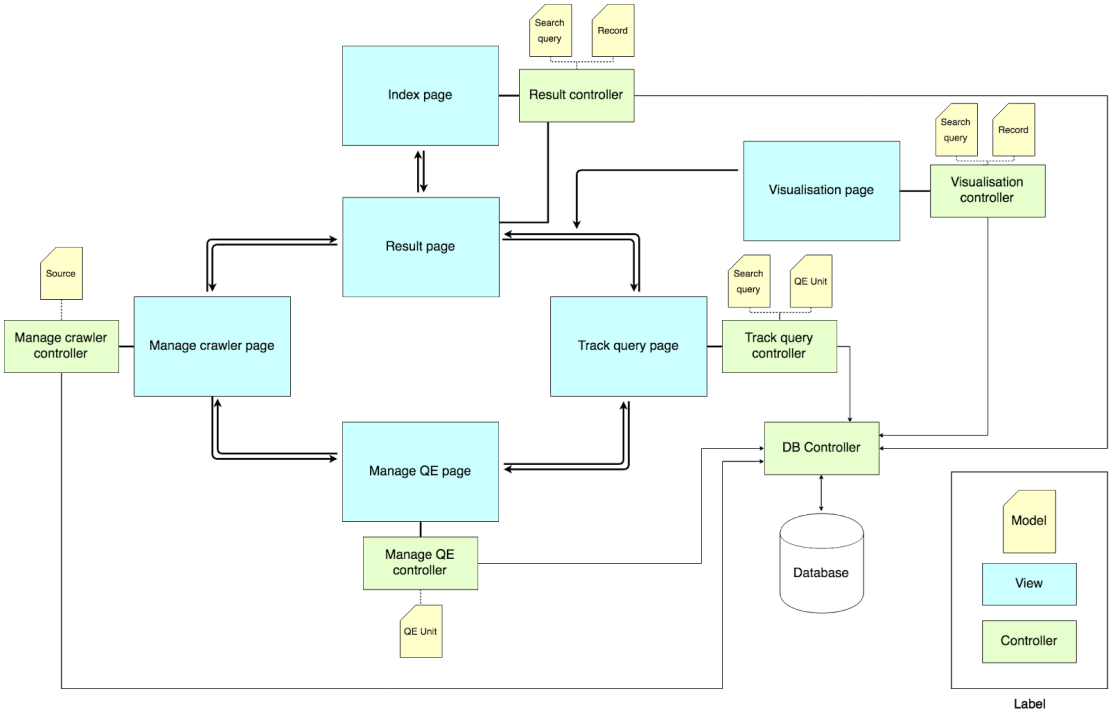


Figure 5: Summary diagram of system interactions

4.3 Query Expansion

We believe that it is possible to get more records by generating more queries from the existing data. Basically, the input for an expansion is the data of related records, from a given query and outputs an expanded query. The requirement D1 and D2 give explicit idea on designing query expansion approaches. There will be 2 approaches for query expansion which are *automatic expansion* and *manual expansion*. Automatic expansion satisfies the requirement D2, the system automatically performs a query expansion on the tracked queries, this is *non-interactive*. In each round of an automatic expansion, the system looks for related records of a query, analyses and processes them into statistical data, and composes a new expanded query. Manual expansion is *interactive*. It satisfies the requirement D1, allowing users to perform expansion on queries by their own judgements. The author also believes that human could identify potential insight from data better that the machine and would be good if there is an aid from the system. A help from the system is in a form of information. Actually, the system has that information in hands. According to the design of an automatic expansion above, the processed information from related records could be helpful to users, showing that information could assets the decision on manual query expansion.

The idea is a good new query comes from a good data analysis, using various techniques from the text analysis. Therefore, the author designed the software architecture of the query expansion that supports any text analysis on the collection of records. We divided query expansion into 2 components, *query generator* and *query selector* (generator and selector in short). Generator generates ranked terms and helpful information to be used in the selector. Selector selects the best term and composes an expanded query from the given ranked terms. More detail including the class diagram of the query expansion will be discussed in Chapter 5, in Section 5.5.2.

Chapter 5 **Implementation**

The detail of implementation starts from the discussion of programming language choices in Section 5.1. As mentioned in previous chapter, the MVC architectural pattern were used to describe the design. Therefore, the implementation will also be described in three aspects of the MVC components which are Model, View and Controller in the Section 5.2, 5.3 and 5.4 respectively. Finally, Section 5.5 explains how the system services works and implemented.

5.1 Programming Languages

We decided to develop the project with Scala for several reasons. The author already has certain experience with Java and it would be a great opportunity to learn how Play works in Scala. In addition, most developers in a community recommended using Scala if the project is not necessary to build upon the infrastructure that only supports Java [22]. The current version of Play itself is also written in Scala [8] which means Scala is native for the framework and is a main reason for me to go with Scala.

5.2 Model

Most models, except record, are easy to implement using Scala case class. A case class is similar to a normal class, but it is good for modeling an immutable data [23]. In order to save the object to the database or read it from the retrieved JSON document to a specific type, the object type needs to be written in a Scala case class. Writing a case class is very easy, it is simply a class with only attributes however methods are not allowed in a class. There are interesting aspects of specific models listed as follows.

- **Source:** Source is a model that act as a medium for source related operations such as adding and editing a crawler. Several components are related with the source model. A `sources.json` is a configuration file located in a server, while source model is a format of data kept in JSON configuration file. The file basically keeps all sources information. The idea for a file is that users could monitor and manipulate the files directly if they do not want to interact with the frontend. There is also a class called `Sources` which is responsible for holding live sources reading from the configuration file. It also handles incoming operations on `sources.json` which are reading, writing and updating the crawler.
- **Record:** There is no case class for records because the record format is not static. A `RecordMapper` class is responsible for creating records. It maps raw responses to mapped records according to a mapping setting of a particular source in the configuration file. Figure 6 shows the example of a mapping. As stated in Chapter 3, there are several mandatory attributes which are `text`, `title`, `createdTimestamp` and `url`. All 4 attributes need to be stated in the mapping setting. Special cases are for title and URL. Title's value could be "None", in case of no title is available like tweets. URL could be specified as a pattern, specified with { … } where the value in a bracket is a raw response attribute name. The example of the URL pattern is also shown in Figure 6. Additional attributes that are specific to a source are also supported such as tweet ID, hashtags and etc.

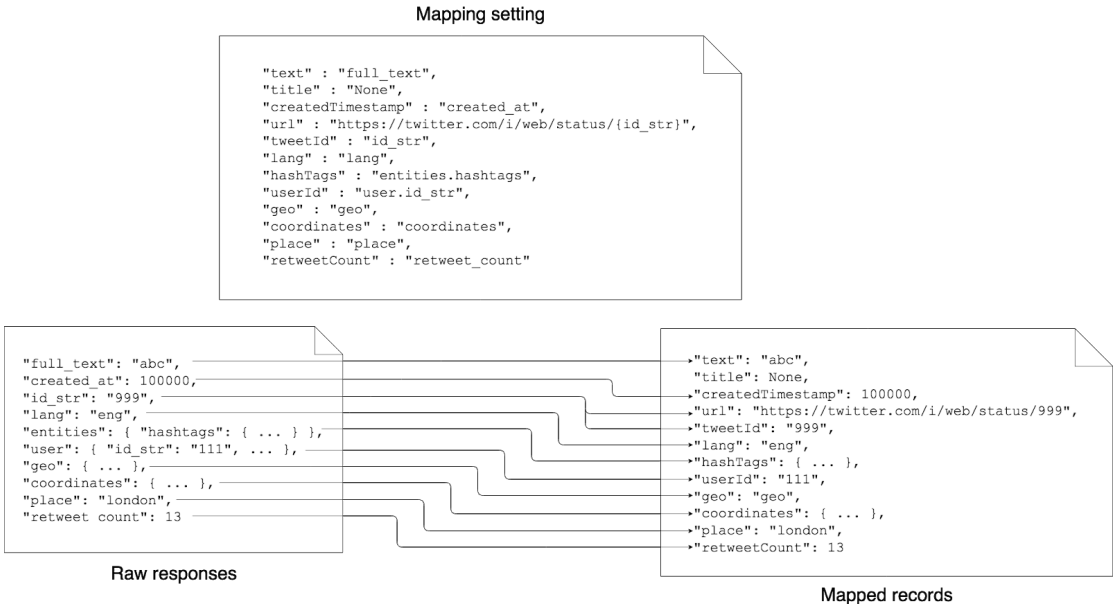


Figure 6: Example of a record mapping according to the mapping setting above

5.3 View

5.3.1 Main views

Tools: Native Play twirl template, Vue.js, Bootstrap and Bootstrap Vue

Overall: A main view is consisted of a template file, JavaScript files and CSS files. The template file is a Twirl template and it mainly has fundamental html components and imports of files and packages. JavaScript files are the main actors. There will be a single Vue application file and the rest would be Vue components reusing throughout the view. The view mechanism is controlled by the Vue application. Figure 7 explains the structure of a main view in term of files.

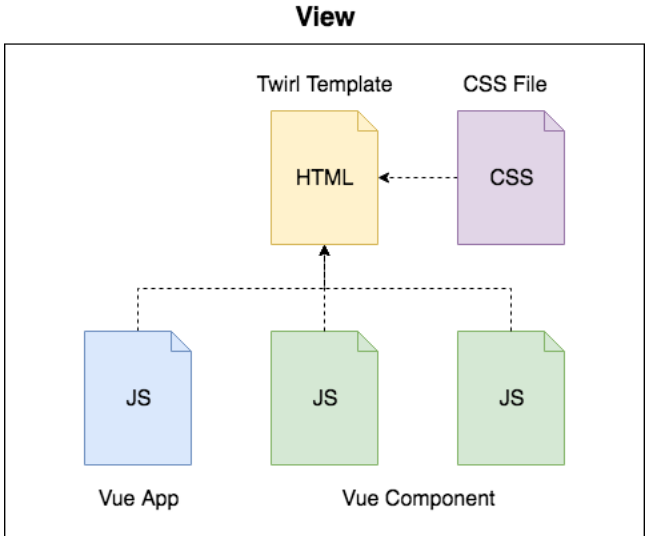


Figure 7: Structure of a main view explained in term of files

Figure 8 shows an example of implemented main views, a result page. All screenshots of implemented main views are in Appendix B and it is highly recommended for the readers to look at the appendix to understand the system clearer.

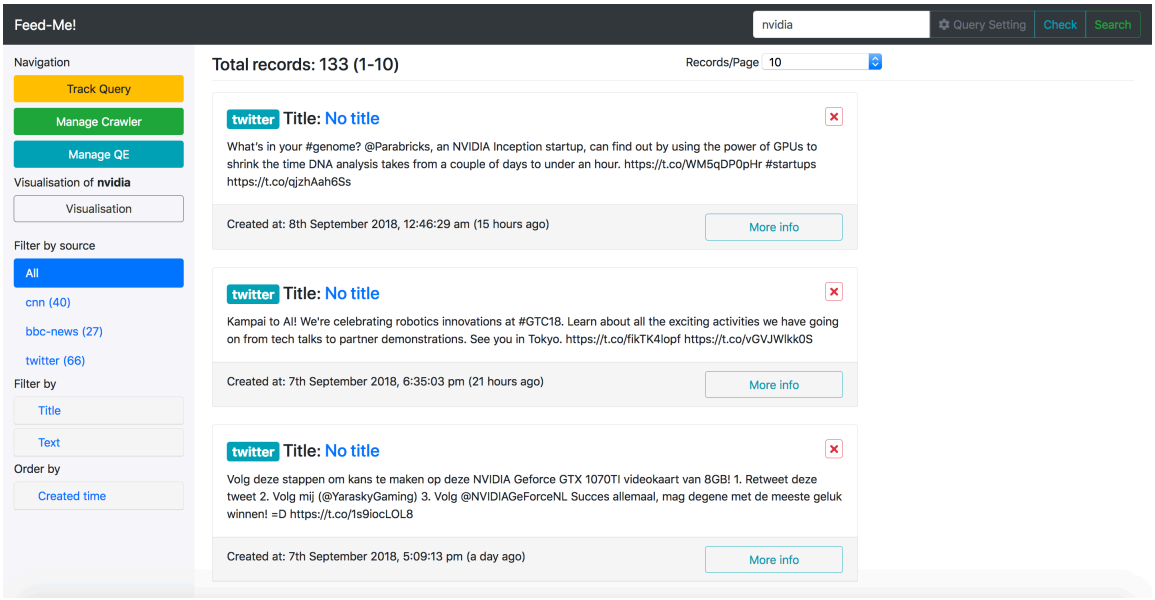


Figure 8: Screenshot of result page

Challenges and solutions: Modern web applications focus on interactions between users which consequently improves user experiences. In order to achieve this, JavaScript, one of the fundamental web elements, is necessary. Native Twirl template offers limited features for simple web development through Scala. Yet, it reduces the developer learning time due to its simplicity [24] however, in the author's opinion, it misses a point of JavaScript. Many JavaScript libraries are very helpful. They provide efficient and effective solutions, but coding in JavaScript are required in order to use them. Coding in Twirl does not directly support the writing of JavaScript codes, particularly the integration of different JavaScript libraries. This is the main reason why I chose Vue.js and implemented most of frontend code using Vue.js instead. Using Vue.js gave several advantages. The most important thing is a seamless integration of JavaScript libraries, as mentioned above. Another thing is Vue.js has a huge online community which is bigger than Twirl, easier to find solutions of bugs and problems. On the other hand, there are problems persisted from the decision. Data flow from Play controllers to view had to be altered. Normally, data could be bound directly from the controller to Twirl template. However, moving the data from the template to JavaScript is difficult. The solution is to let JavaScript gets data via API explained in Figure 10. This choice of implementation also made the codes longer because more files for Vue.js are needed.

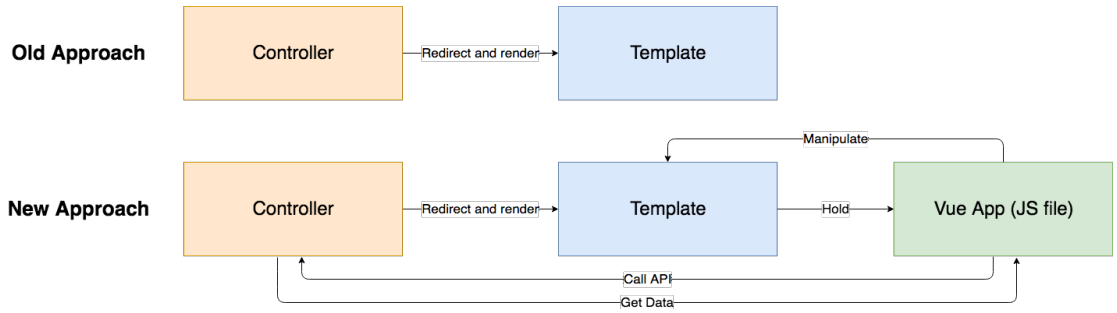


Figure 10: Comparison between old and new approaches

5.3.2 Visualisation

5.3.2.1 Bar graph

Tool: Amchart

Overall: It was easily implemented as a result of Amchart, which provides a rich bar graph feature. The system needs to prepare data via API according to their documentation and everything will be worked smoothly. Figure 11 shows the implemented bar graph in visualisation page.

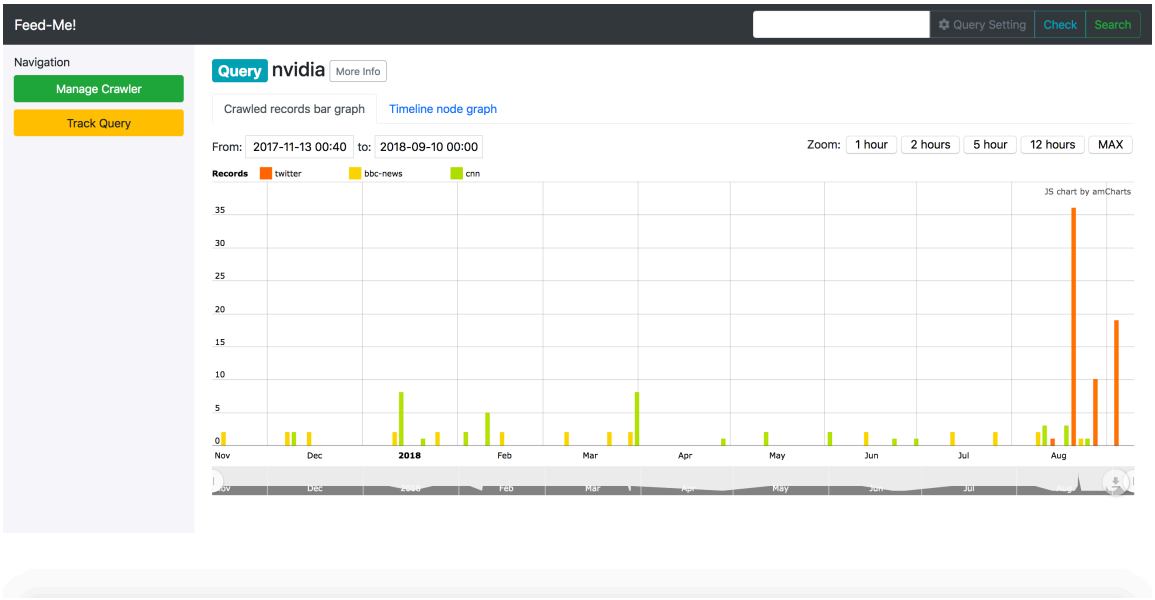


Figure 11: Bar graph

5.3.2.2 Timeline graph

Tool: D3.js

Overall: As stated in Chapter 2, D3.js offers a very fundamental manipulation of DOM. The author found that it was difficult at the beginning to create the node-edge graph. Fortunately, it was successfully implemented as a result of following force directed graph tutorial published by Tom Roth [25]. The guide was extremely helpful for a beginner. It shows a simple code and also the approaches to advance the graph. Overall, steps to create the graph are

- Create DOMs for nodes and links (edges)
- Customise their properties such as colour, size, CSS styles, etc.
- Create a DOM for top-right info box which will be used in click event of nodes
- Create a DOM for graph legends explanation box on the left
- Add click event to nodes (showing information in info box after clicking a node)
- Create a simulation for this graph
- Apply the graph data (nodes and links) and force properties to the graph; like the charge force and link force
- Apply drag and zoom handlers which allows the graph to be draggable and zoomable

The system also needs to provide API that processes and prepares data for the graph similar to the case of bar graph. Figure 12 shows the screenshot of timeline graph. (Please note that the legends were implemented after the user evaluation)

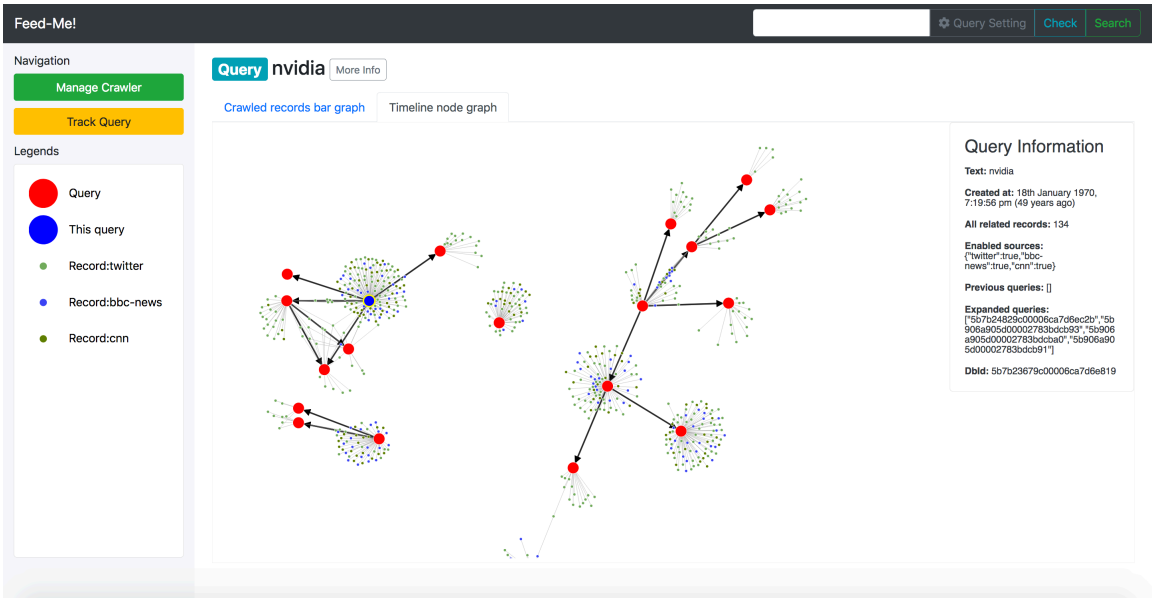


Figure 12: Timeline graph

Challenges and solutions: In real scenario, there would be a lot of records in the database which makes it impossible to render every record in the graph. Since there are forces in the simulation, web browser has to update positions of DOMs frequently and this could be resulted in a lag that deteriorates user experiences. The solution is to limit number of returned records from API and select the records with high number of links to queries first because they have higher reputation and could be insightful for users.

5.4 Controller

Controllers are Scala classes that extends Play's `AbstractController` class. Their responsibility is to handle requests and return actions back to the client [26]. Actions are simply the HTTP responses. Play provides handy classes for frequently used responses such as OK (200), BadRequest (400), NotFound (404) ant etc. [27]. As a result of the dependency injection concept in Play, controllers and other classes are highly reusable. The concept helps modularize classes and lets the developer simply inject classes when they are needed without the use of global static states that rendered the system less flexible [28]. Most controllers have similar operations, such as showing information, add, delete and edit information, and were implemented in the same direction. The following section lists several interesting controllers.

5.4.1 DbController

Tool: Reactivemongo

Challenges and solutions: Since the author had no experience with Scala, it was difficult at the beginning to understand database access codes which heavily rely on asynchronous operations. In the author opinion, problems came from the syntax and the difficulty of debugging asynchronous operations. The only solution to the problems is calmly spending time on learning fundamental Scala asynchronous concepts. The author was misled by the idea of just following examples provided by the website without any fundamental knowledge at first. Learning the asynchronous operation syntax, how it works, and understanding basic examples are much more crucial.

5.4.2 ResultController

Overall: The interesting point here is how querying works. Figure 13 shows complete flow of a querying.

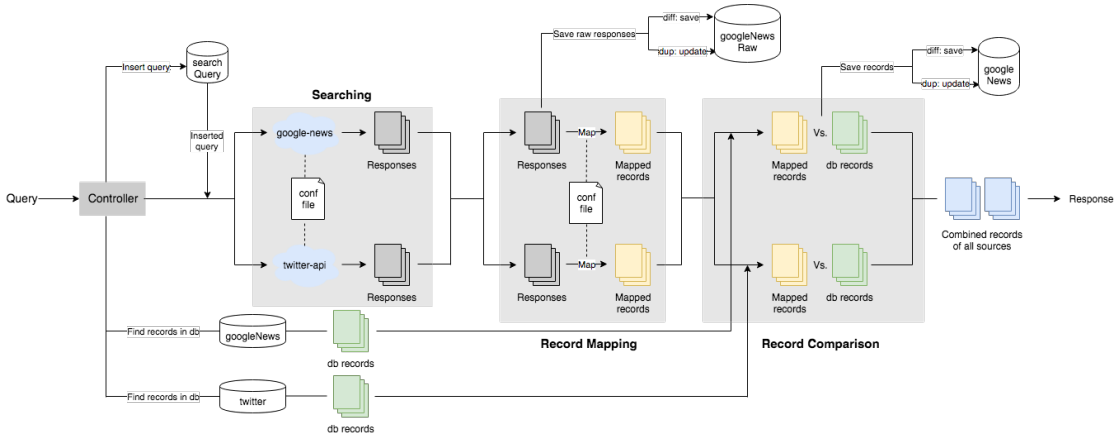


Figure 13: Flow of querying starting from the input query

Please note that each horizontal line means an asynchronous operation. It starts from a submission of query to a controller. The query will be inserted into the database. If it is already existed, the existed query will be retrieved and if not, the new query will be inserted and returned to the controller. At the same time, the controller gets all records from the existing database collections which will be used during the record comparison step. After a query is retrieved from the database, the controller asynchronously performs searching from API according to the number of enabled sources listed in the configuration files. Then, raw responses will be mapped into mapped records according to the mapping setting of a particular source in the record mapping step. In addition, raw responses are saved into corresponding raw database collection. In this case, raw responses from google-news are saved into "googleNewsRaw" collection while twitter would be "twitterRaw". Subsequently, the controller compares mapped records with the retrieved database records. The comparison aims to identify new unique records and save them to the database. On the other hand, duplicated records will not be saved but updating the attribute called "query" of the existing records by adding the new query database ID to this field. This attribute acts like a foreign keys keeper, telling which query the record is related. Lastly, the controller combines new and existing records (not all records but only records related to the query) together and return them back to a client.

Challenges and solutions: As you can see, the querying is a long sequence of asynchronous actions. Chaining these actions together was tough. As mentioned in Section 5.4.1, the author started looking for example of codes instead of studying the fundamental concepts first. It led to lot of time wasting on googling for examples and the errors solving. However, after decided to study the Scala asynchronous concepts, the author found an official tutorial about the Scala Future, one of the Scala asynchronous actions [29]. Future acts like a placeholder. The codes within a Future will operate asynchronously and it could also have a callback after the completion. The interesting syntax is called a for-comprehension. It lets the developers chaining Futures together in a sequential manner. Figure 14 shows the code snippet of the querying using a for-comprehension with Futures. Within a for, the right-hand side is a Future operation while the left is a result returned from the Future. `Future.sequence` is just the execution of a Future array, showing in Figure 13 that operation such as searching on each source is executed in parallel. `dbc.seqInsertCrawlerStats` saves number of crawled records into the database

as a log. The values specified in `yield` are returned result to a caller of this querying. In this case, it returns list of records from each source with the query and statistic of this querying.

```
val aggFutResponse = for {
    // insert if it isn't exist [or] return the existing query
    insertedQuery <- dbc.insertSearchQuery(searchQuery)
    // find only records from query's enabled sources
    dbRecords <- Future.sequence(dbc.seqFindRecordsFromCollection(
        sources.getAll
    ))
    // call the api(s) and get responses back
    apiResponses <- Future.sequence(apiHandler.seqSearch(insertedQuery))
    // map responses according to sources.json
    mappedRecords <- Future.sequence(recordMapper.seqMapRecord(insertedQuery, apiResponses))
    // compare the mapped records with db records then
    recordsTriples <- Future.sequence(recordComparison.seqCompare(insertedQuery, mappedRecords, dbRecords))
    // insert only new records
    dbResponses <- Future.sequence(dbc.seqInsertJsonDocs(recordsTriples.map(t => (t._1, t._2))))
    // save crawling statistics to db
    crawlerStats <- Future.sequence(dbc.seqInsertCrawlerStats(insertedQuery, recordsTriples))
} yield (recordsTriples.map(t => (t._1, t._3)), insertedQuery, crawlerStats)
```

Figure 14: Code snippet of the querying in `ResultController`

5.5 Services

There are 2 services scheduled on the background after the system started. The description are as follows.

5.5.1 Crawler

Overall: It basically updates queries by performing a querying on each query to get new records. As a result of a query method (Figure 14) located in the `ResultController`, this was easily implemented by injecting a `ResultController` into the crawler service and call a query method when time is triggered.

5.5.2 Query Expansion

Challenges and solutions: *Strategy software pattern* is used to satisfy the idea of flexibility. The pattern allows the system to choose an execution of an algorithm, from the same family, at runtime [30]. Query expansion is divided into 2 components, query generator and query selector. Both of them were implemented with abstract classes, which means users need to implement their own generator and selector by overriding the abstract methods. Detail of the generator and selector input, output and methods are summarised in Table 5.

Method name	**Abs-tract**	**Input**	**Output**	**Description**
Query Generators				
Process	Yes	List of related records	Map of information extracted from records	Generate information and output as a map given the list of associated records
Rank	Yes	Map of information from process	Sequence of sorted ranked terms	Generate sorted rank terms with the information from process method
onSch-edule	No	None	Map of queries and their sorted ranked terms	The method is called by a scheduler, performing process and rank then save the sorted ranked terms into database. This information helps users make decision on manual expansion.

Selector				
Select	Yes	Query and sorted ranked terms from its related records	Expanded query	Select the best term and compose into an expanded query from the given sorted ranked terms

Table 5: Generator and selector component summary

The query expansion unit wraps both components together. When time is triggered, a query expansion scheduler will automatically call the generator and selector within a query expansion unit. In addition, the author has provided pre-implemented generator and selector classes. There are two generators which are `MostFrequentQueryGenerator` and `TfIdfQueryGenerator`. The first one counts term frequency (tf) of all records and ranks terms by tf in descending order. While the later additionally computes inverse document frequency (idf) and multiplying tf and idf to get tf-idf value, a normalised tf value. `SimpleQuerySelector` is a pre-implemented selector class. It selects the best rank terms and compose an extended query based on the number of terms of an input query. Table 6 explains the pre-implemented classes in more detail, including the equation used for idf calculation. Figure 15 shows a class diagram of the query expansion and pre-implemented classes.

Generator		
Class	**Method**	**Description**
MostFrequentQueryGenerator	Process	Combine term frequency (from the record's tf attribute) of all records together and output a map with term frequency data
	Rank	Rank the terms from the highest to lowest frequency
TfidfQueryGenerator	Process	It combines term frequency of all records together, compute inverse document frequency (idf) and tf-idf by multiplying tf with idf. The formula for idf calculation for each term is $\log \frac{n-dk+0.5}{dk+0.5}$ where n is number of all records and dk is the document frequency of the term.
	Rank	Rank the terms from the highest to lowest tf-idf
Selector		
SimpleQuerySelector	Select	Select the first term (best term) from the ranked terms and compose an expanded query considering following criteria • Query has exactly a term: Expanded query will be the best term of ranked terms (ignoring a query) • Query has more than a term: Expanded query will be the best term concatenated with the *most frequent query term* according to the ranked terms

Table 6: Details of pre-implemented classes

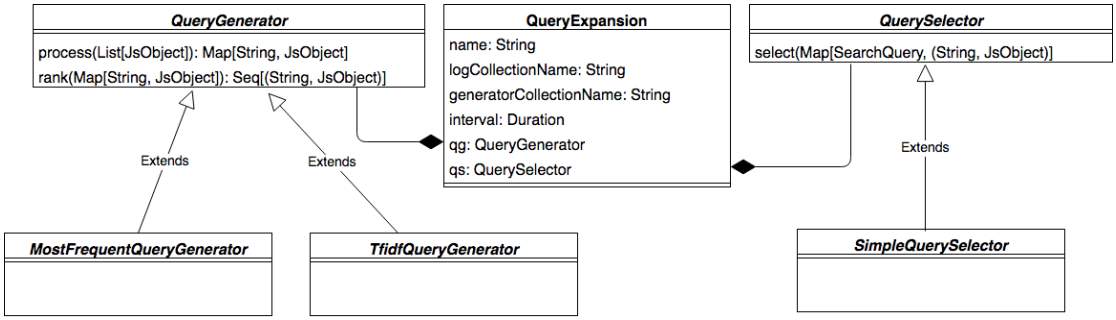


Figure 15: Class diagram of query expansion and pre-implemented class

5.5.3 Scheduler for Crawler and Automatic Expansion

Overall: The crawler and automatic query expansion services are registered with Play module. A module starts right after Play application started, including both registered services. The implementation approach of both services is similar. The system schedules a crawling and expansion task using Akka scheduler. It is similar to scheduling a runnable task in Java but based on Akka, which is a Play official supported concurrent platform [31].

Challenges and solutions: The challenge is a performance issue. The system needs to control number of threads for a task properly to prevent memory leak, which resulted in an internal crash. Burst number of threads could happen because there could be many asynchronous operations within a single task. The solution is a thread blocking. Scheduled tasks that performed within a thread and background operations do not have to be fast. Therefore, blocking a task to perform within a single thread is a good choice to prevent the crash.

Chapter 6 Testing and Evaluation

The project is evaluated in 2 ways, through correctness test on crucial operations using the software testing and evaluated overall functional requirements using user evaluation. Section 6.1 explains the detail of the software testing and the user evaluation is described in Section 6.2.

6.1 Software Testing

The system was tested using ScalaTest, a popular testing framework for Scala, and it is also officially recommended and documented by Play [32]. ScalaTest provides simple and clear test cases implementation including the interesting feature called matcher that simplifies the assertion into more readable structure [33].

The author identified 2 crucial main features for the software testing, a querying and a query expansion. Querying is also applied to the "crawling" of crawlers. As explained in Figure 13, there are 3 main steps for querying starting from searching, mapping and comparison. Searching is trivial because all it needs to do is to use settings provided by a source and then perform the searching. Therefore, the author chose to test on mapping and comparison steps since there are many logics behind them. Query expansion is another important feature, especially the insert expanded query step. When new expanded query is inserted, there are lots of small steps happened such as updating all previous queries of the expanded one.

After the selection of testing features and steps, test cases for each step were identified. For example, there could be a case like this when inserting an expanded query.

- Assume that there are 2 queries in the database: Expanding the query #1 by query #3 must update
    - expanded queries attribute of query #1 with database ID of query #3
    - previous queries attribute of query #3 with database ID of query #1

This is just an example test case of the insert expanded query step. In addition, there could be any number of a*ssertions*, which are Boolean expressions that tell whether a program works correctly [34], within a test case. More detail of all identified test cases is explained in the Appendix C. A test case would be passed if all of its assertions are passed, otherwise failed.

Table 7 shows the summary and results of software testing in term of test cases and assertions which categorized by feature and step. The number of test cases indicates the identified test cases of that step while number of assertions is the total assertions within all test cases followed by number of assertions in each test case in the parenthesis. Pass and fail columns indicate the status of the test cases. As you can see, the system had successfully passed all test cases. However, it would be practically good to have test cases for other features as well.

Feature	Step	No. of test cases	No. of assertions	Pass	Fail
Querying	Record mapping	3	5 (2, 2, 1)	3	0
	Record comparison	3	11 (3, 4, 4)	3	0
Query expansion	Insert expanded query	3	22 (7, 8, 7)	3	0

Table 7: Summary of test cases grouped by feature and step

6.2 User Evaluation

User evaluation is the evaluation that performs on users, called evaluators, that aims to test main functional requirements of the system. It is an important part of the project since it allows the developers to observe interactions between users and the system. Developers could also ask for feedbacks from the evaluators, qualitative results including likes, dislikes and suggestions.

Overall, the author first listed all implemented M and S requirements and then generated the evaluation tasks from them, grouped by main views. After that, database and materials used during the evaluation were prepared. Next, the author looked for suitable evaluators, and asked them for the meeting date, time and location. The first evaluator was a test pilot for this evaluation. The author used the observations from the first evaluation and suggestions given by the evaluator to improve and change materials that could enhance the experience of other evaluators.

Section 6.2.1 explains the overview of evaluators and their background information. Preparations needed before the evaluation are discussed in Section 6.2.2. All materials that were used during the evaluation are described in Section 6.2.3. After that, Section 6.2.4 describes the steps for the user evaluation. The success measurements are determined in the Section 6.2.5. The result and discussion are shown in Section 6.2.6. Lastly, discovered problems were explained in Section 6.2.7.

6.2.1 Evaluators

There are 6 evaluators. All of them are programmers, they have a certain level of programming experience. Five of them studies here at University of Glasgow while three studies the Data Science, one studies the Information Technology and another studies the Computer Science. Another evaluator studies the Information Technology at University of Reading. Their ages range from 23 to 35. An evaluator from University of Reading do not have any working experience while the rest have worked in IT sectors for at least 2 years.

6.2.2 Preparation

6.2.2.1 Task-based Test Cases

Test cases focus on the evaluation of must-have and should-have requirements and are grouped by main views. They are task-based which means a test case is a single task and evaluators need to complete the given task in order to pass a test case. Completion of a test case is specified by its success measurement. For example, the success of a query deletion task is that the query is removed from the view. The task could be anything that could measure a success. There could be a variable in a task. For example, task A1 asked evaluators to pick a query, named query “A”. They are free to choose any interesting event and represented as a query used in the task. The detail of all test cases is very long, please having a look in Appendix D for all test cases.

6.2.2.2 Data Preparation

The database was mocked in order to provide the same environment for all evaluators. Models were mocked to conform the tasks from Section 6.2.2.1. Models that were mocked are queries, records, sources (crawlers) and query expansion units. List and description of prepared models are as follows.

- Queries: 7 in total
    - Nvidia: represented an announcement of new 2000 graphic card series and used for evaluating querying requirements
    - 2080: expanded from Nvidia
    - Morandi bridge: represented a collapse of highway bridge in Italy and used for evaluating visualisation requirements
    - Morandi victims: expanded from Morandi bridge
    - Collapse: expanded from Morandi bridge
    - Building: expanded from collapse
    - Cave Thailand: represented old query for deletion requirement
- Crawlers: 3 in total which are from Twitter, BBC and CNN sources
- Records: 346 records from 3 sources
- Query expansion unit: 2 in total
    - QE1: using TfidfQueryGenerator + SimpleQuerySelector
    - QE2: using MostFrequentQueryGenerator + SimpleQuerySelector

6.2.3 Materials

There are numbers of materials for the evaluation listed as follows.

6.2.3.1 Project overview slides

It aims to provide background information of the project. The slides give ideas on introduction, problems, system main features, important terms and the main pages overview.

6.2.3.2 Task sheets

Task sheets give background information, especially intuition to help evaluators understand what they need to do in order to complete tasks. It also improves the continuity of the evaluation by linking corresponding tasks together in the same sheet. A sheet could link 1-2 tasks together, the example is shown in Figure 16 below. The author showed task sheets in a form of presentation that displayed on iPad besides a laptop.

Background: You wonder how query **"nvidia"** is being tracked in the system and would like to see the number of related records. Since event "nvidia" is popular on Twitter, you find that there are few records from "bbc" and "cnn" sources. So, you are going to **turn off "bbc" and "cnn" crawler on query "nvidia"**.	
1	Find the **number of related records** of query "nvidia"
2	**Turn "bbc" and "cnn" crawlers off** on query "nvidia"

Figure 16: Task sheet example

6.2.3.3 Questionnaire

It aims to quantitatively evaluate the interesting aspects of the system and qualitatively asked for evaluators opinions. It is in a Google Form, which is easy to aggregate results. The questionnaire is divided into 2 sections.

Section 1: Rating

The score ranges from 1 to 4 where one is poor, two is passable/acceptable, three is good and four is excellent. These are the list of rating criteria followed by the interesting aspects of the system. Please have a look at Appendix E to find more description of each aspect.

1. *System overall:* Ease of use, feasibility, directions/guides within the system, and responsiveness
2. *Visualisation [Bar graph]:* Insight gaining from the graph and ease of understand
3. *Visualisation [Timeline graph]:* Insight gaining from the graph and ease of understand
4. *Expansion:* Feasibility and helpfulness of generated information for manual expansion
5. *Evaluation:* Directions and enjoyment

Section 2: Qualitative section

This section asks evaluators about the *likes*, *dislikes* and *suggestions* they have toward the system.

6.2.4 Evaluation Steps

1. The author gives a presentation on project overview
2. Starts the evaluation by
    a. The author shows a task sheet that comprises of the task background information and list of tasks that an evaluator needs to complete
    b. The author asks the evaluator to read and understand the task thoroughly and he/she could ask questions about the task if it was not clear
    c. Let the evaluator starts working on a task until completed, while the author records time spent for the task
    d. Starts evaluating next task by repeating step a to c until all tasks are completed
    e. The author only gives a suggestion only when the evaluator is misled
    f. Task is incomplete if the evaluator gave up

6.2.5 Success Measurements

As mentioned in Section 6.2.2.1, the task would be marked as completed if the evaluator successfully completes the task. Task will be failed if the evaluator cannot find the way to meet the success measurement of that task and gave up. An evaluator would successfully pass the evaluation if he/she has at least 80% pass rate, calculating from all tasks.

A criterion in the first section of questionnaire would be considered as success if the average score of that criterion from all evaluators is higher than 2.5 (out of four).

The evaluation is successful if there are at least *80% of passed evaluators* and *80% of passed criteria in rating section*.

6.2.6 Result Summary

6.2.6.1 Average Time spent

#	Evaluators												Average time spent (s)
	E1		E2		E3		E4		E5		E6		
	R	TS	R	TS	R	TS	R	TS	R	TS	R	TS	
A. Index and Result page													
1	P	35	P	8	P	18	P	10	P	30	P	12	19
2	***F***	***119***	P	47	P	52	P	41	P	46	P	40	***58***
3	P	22	P	13	P	14	P	23	P	14	P	14	17
4	P	31	P	15	P	4	P	30	P	21	P	5	18
5	***F***	***74***	P	56	***F***	***74***	P	54	P	33	P	22	***52***
B. Track query													
1	P	34	P	20	P	13	P	20	P	6	P	11	17
2	P	13	P	9	P	6	P	3	P	3	P	20	9
3	P	28	P	12	P	42	P	10	P	20	P	23	23
4	P	27	P	3	P	18	P	21	P	22	P	32	21
5	P	30	P	30	P	7	P	3	P	18	P	6	16
6	P	10	P	6	P	5	P	9	P	7	P	6	7
C. Visualisation													
1	P	19	P	75	P	51	P	106	P	35	P	60	***58***
2	P	120	P	79	P	139	P	80	P	95	P	47	***93***
3	P	5	P	2	P	4	P	3	P	5	P	1	3
D. Manage crawler													
1	P	26	P	53	P	48	P	79	P	51	P	54	52
2	P	12	P	6	P	33	P	16	P	25	P	4	16
3	P	15	P	6	P	6	P	3	P	3	P	12	8
4	P	5	P	3	P	2	P	4	P	4	P	7	4
5	P	5	P	2	P	4	P	3	P	2	P	5	4
E. Manage QE													
1	P	4	P	4	P	6	P	5	P	5	P	7	5
2	P	5	P	10	P	2	P	2	P	2	P	1	4
3	P	3	P	4	P	5	P	2	P	4	P	2	3
EPR	**91%**	-	**100%**	-	**95%**	-	**100%**	-	**100%**	-	**100%**	-	-

= Number
E = Evaluator
R = Result
TS = Time spent (s)
P = Pass
F = Fail
Bold = Important or interesting (discussed below)

EPR = Evaluator Pass Rate

Table 5: Recorded time spent for each task

Table 5 shows the time spent of tasks, average time spent, pass or fail statuses and the pass rate of each evaluator. As you can see, *all evaluators have more than 80% pass rate indicating that they all passed the evaluation*. This is the discussion for the result above.

A2: Look for overview information of event A by querying "A"

- The first evaluator chose "Ed Sheeran" as "A" which was not an event that could be a news directly, since there was no big movement like new album release at that time. So, it is hard to find overview information of that event.

A5: Disable a “twitter” crawler and search for the event “A” again

- Every evaluator said that the icon is hard to find and understand. Gear is not meaningful and one of them feel reluctant, it looks dangerous, to click it.

C1: [Bar graph] Find the number of records from each source on 18/08/2018

- Most evaluators had problem with counting number of records within the bar graph, except the first one. The mistake was in tasks and data preparation.

C2: [Timeline graph] Identify the records that have multiple links and interpret their relationship between queries

- The aim of this task is to let evaluator explains relationships of records that links with many queries, why they have such relationships. However, most evaluators did not get the idea of timeline graph because the graph itself lacks indications. Most critical part is a lack of legends, telling the meaning of each component in the graph.

6.2.6.2 Questionnaire: Section 1

Subsection	Criteria	E1	E2	E3	E4	E5	E6	Average score
1. System overall	Ease of use	2	4	4	4	3	3	3.3
	Feasibility	***4***	***4***	***4***	***4***	***4***	***4***	***4.0***
	Directions/guides within the system	***4***	***2***	***3***	***2***	***1***	***2***	***2.3***
	Responsiveness	3	4	4	4	4	4	3.8
2. Visualisation: Bar graph	Insight gaining from the graph	3	3	4	3	2	4	3.2
	Ease of understand	3	3	4	3	3	3	3.2
3. Visualisation: Timeline graph	Insight gaining from the graph	3	4	3	3	4	4	3.5
	Ease of understanding	***2***	***3***	***3***	***3***	***3***	***2***	***2.7***
4. Expansion	***Feasibility***	***3***	***4***	***4***	***4***	***4***	***4***	***3.8***
	Manual expansion: helpfulness of generated information	3	4	4	4	2	4	3.5
5. Evaluation	Directions	2	4	4	4	4	4	3.7
	Enjoyment	3	4	4	3	4	3	3.5

Table 6: Rating score summary

Table 6 shows the summary of rating score from the first section of the questionnaire, describing the scores from all evaluators and average score of criteria. As you can see, most criteria have the average score higher than 2.5, except the directions/guide within the system which is 2.3, which means *there are 92% (eleven-twelfths) of passed criterion.* The discussion of the interesting result are as follows.

System overall: All evaluators believes that the system would be practically worked. However, the directions and guides within the system is just acceptable. One of them even rated as poor. It would be impossible for typical users to begin using the system without any tutorials. However, it is interesting that the first evaluator rated the directions as excellent. The author believes that it was because the author gave him too many directions and helps since he was the first evaluator and he might be biased by this point.

Visualisation: Timeline graph is not easy to understand, because of the lack of indications. The graph itself provides interesting insight but they could not identify the meaning of each component. Legends are very important for the graph.

Expansion: They believes that the expansion feature would be practically worked, which is a promising sign.

6.2.6.3 Questionnaire: Section 2

The summary of likes, dislikes and suggestions are listed as follows. However, the suggestions from the questionnaire is too short and not descriptive. So, the author also adds the recorded suggestions given by the evaluators during the evaluation into the list and grouped them by tasks.

Likes

- The system could get reliable news from many sources
- Query expansion can be well explained by node graph, which reveals potential relations between events
- The system would be good for many developers
- Query expansion feature
- Timeline graph
- Idea of expanding query from time to time

Dislikes

- The system is somehow lag at the beginning of the evaluation
- There is only connected nodes and edges. There is no text label or each relation display on the graph as it requires user to click see in detail each by each
- There are some unclear UI, especially icons that are hard to understand
- Some evaluation tasks are complicated and have complex interactions
- The location of query setting
- There are lots of text and sometimes it is hard to find what is needed to do

Suggestions

- A4: Remove irrelevant record(s)
  - It is good to have duplicate filter or retweet filter
- A5: Disable a “twitter” crawler and search for the event “A” again
  - Icon is hard to find, and gear icon is not appropriate
  - Icon is too dark, gear is not appropriate, and some keywords are needed
  - Query setting could be moved to the left (on filter panel) and gear icon looks dangerous
  - Query setting could be moved to the empty right panel
- B1: Find the number of related records of query “nvidia”
  - It would be great to show number of related records in check query too
  - Total records could be moved to the first line instead
- B2: Turn “bbc” and “cnn” crawler off on query “nvidia”
  - Query card could be collapsible to save more spaces and see more queries
- C1: [Bar graph] Find the number of records from each source on 18/8/2018
  - The bar graph is not easy to use
- C2: [Timeline graph] Identify the records that have multiple links and interpret their relationships between queries
  - Node type in info box is too big, legends are needed for nodes, and also indication of colors (source)
  - Query nodes can be represented with pictures/symbol/text
  - Legends for node types in timeline graph are needed, also for arrows (expansion)
- D1: Add new crawler to the system
  - Newly added crawler could be added at top of the list, instead of bottom

- D2: Figure out when is the next crawling time of a specific crawler
    - Highlighting next crawl and last crawl attributes could make it clearer

6.2.7 Problems

This section highlights interesting problems occurred during the evaluation.

- The evaluation on a first evaluator was the most troublesome. It was like a test pilot. The materials, especially the task sheets, were ambiguous. The direction of a task was not descriptive enough. In addition, the overview slides did not provide enough information for the first evaluator to help him understand the system. A good example would be the selection of a query "A", mentioned in Section 6.2.6.1. He picked "Ed Sheeran", which is not actually represented any specific event at that time. Therefore, before evaluating the rest, the author refined all materials again by adding more information to the overview slides such as important terms and definition of a query, highlighted the task with colors, changed some ambiguous words and phrases and rehearse more for the overview slides.
- There might be too few records for the evaluation. The system is designed to collect a lot of data so the visualisations could reveal potential insight from them. An evaluation of the bar graph was affected by this problem. Bar graph had a weird behavior during the evaluation. The time axis (x) was not separated by day (1/9/2018 following by 2/9/2018 and so on). But, it was separated by day-noon time-day order (1/9/2018, 12:00 of 1/9/2018, 2/9/2018 and so on). As a result of this, task C1, which required evaluators to count number of all records on a specific date, was harder than expected.

6.2.8 Conclusion

In summary, the evaluation was *successful* by meeting both 2 statements, *having at least 80% of passed evaluators* and *80% of passed criteria*. The evaluation process was smooth as a result of efforts from all evaluators. Identified problems in the above section could also be considerations for other evaluations of the similar projects in the future.

Chapter 7 **Conclusions**

Feed-me! is the event data aggregation web application that collects data from different sources together. It featured with customisable crawlers, query expansion and visualisation, that allows users to draw more insight from the data. The system met all must-have and should-have requirements, which is about 76% of all requirements. Although no requirement from could-have and won't-have were met, the system still works with its full potential to achieve the main goal. System overview could be identified very early as a result of initial view designs and all designed main views are successfully implemented. Starting the system design from views provided a high-level understanding and it could be used as a great medium for a discussion with stakeholders. There are a lot of detail on the implementation. Especially when building a web application, many tools were used in different components. Integration of Vue.js, Bootstrap and Bootstrap Vue significantly helped and boosted the development of the frontend. Visualisations were completely developed with the aid of visualisation frameworks. Even Play does not officially provide APIs for NoSQL database connection, the database driver like ReactiveMongo handles this task perfectly. However, the knowledge of asynchronous programming style is heavily used to achieve the fast and non-blocking access to the database. Learning the fundamental knowledge of Scala asynchronicity is highly recommended before working with the driver to prevent the confusion and further unknown errors. Query expansion was developed with the idea that allows users to develop their own expansion technique by extending the generator and selector abstract classes. The current pre-implemented classes for generator and selector might not have sophisticated and highly effective logics. But, this architecture gives the promising future that could support more advanced techniques. The user evaluation result is promising. The feasibility of query expansion was questionable at the beginning, however all users believed that it could be worked in practice according to the user evaluation. On the other hand, guides and directions within the system should be improved hence allowing users to learn how to use the system without the need of any user training beforehand. In summary, the evaluation was successful by having all evaluators passed the evaluation and 92% of passed criterion.

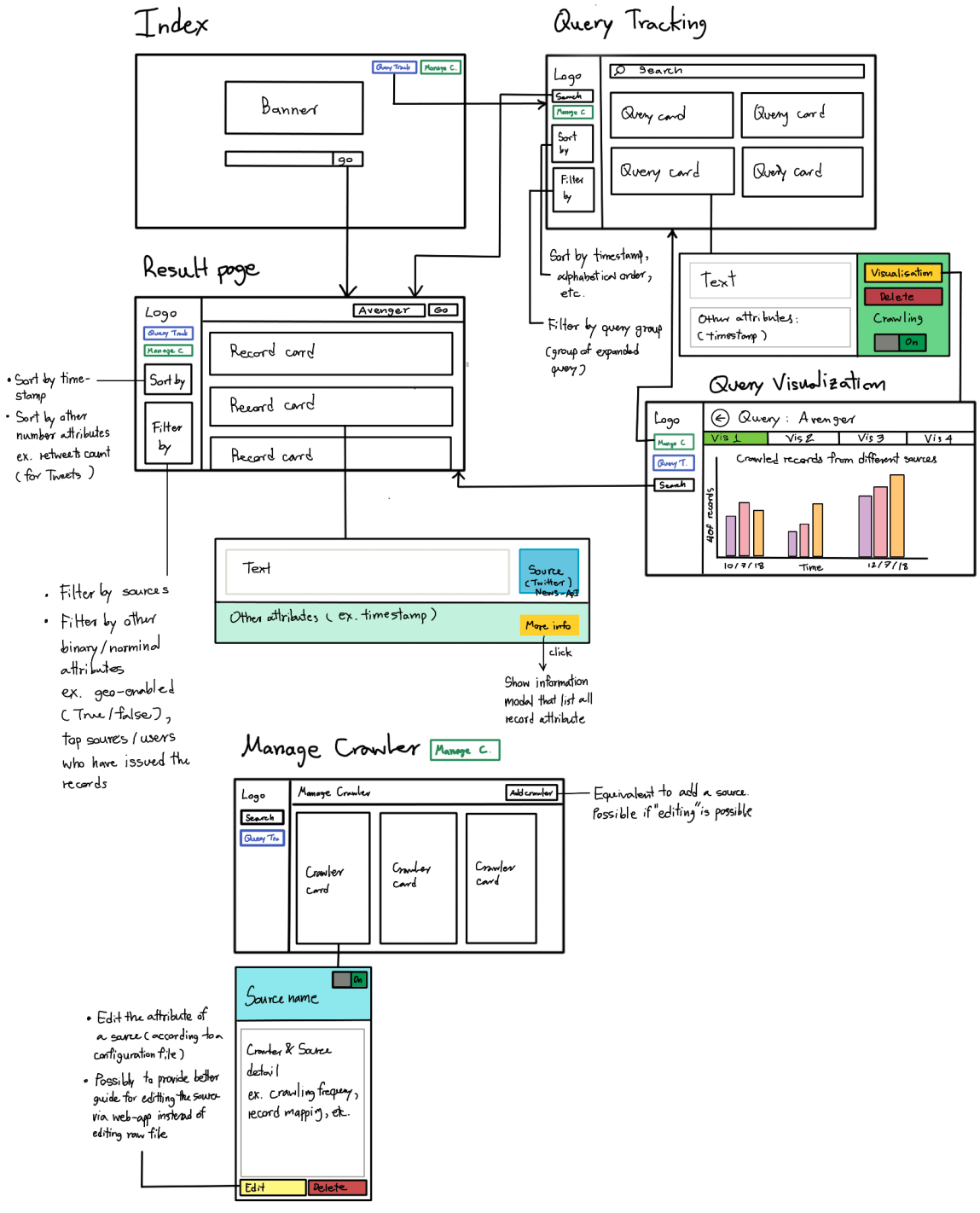
There are many interesting could-have and should-have requirements that could enhance existing features. Grouping queries together in term of query expansion (requirement A7) is a good idea since this could improve the query classification in timeline graph, also in query tracking page. The idea of keeping images (requirement A12) of raw responses should help enriching the content displayed together with the records in result page. In addition, URLs within the records would be beneficial to the system since they could be used to extract more information from the source of URL, or even extracting new queries out of it. Crawler is still too simple in term of the resource utilisation. It should manage limited rate limits more efficiently (requirement B8) by computing crawling period from the available rate limit. An important feature of the query expansion that is still missing is adding query expansion unit from frontend. Users could conveniently compose their query expansion unit from manage QE page if there are many implemented generators and selectors available. In addition, it could be good to have more pre-implemented generators and selectors. The frontend needs minor changes on user interfaces and components position, many suggestions from the evaluation were given. By doing this, it would significantly improve overall user experiences. The missing feature for visualisation was the indications, like legends for timeline graph, however the author had already implemented this in the final version as a last development task.

Timeline graph should handle more records and queries in the future, and at the same time approach to improve the performance of rendering is also necessary. Lastly, the export feature could benefit the uses of gathered data. Users might be able to export selective portion of data in several formats such as JSON.

Chapter 8 **References**

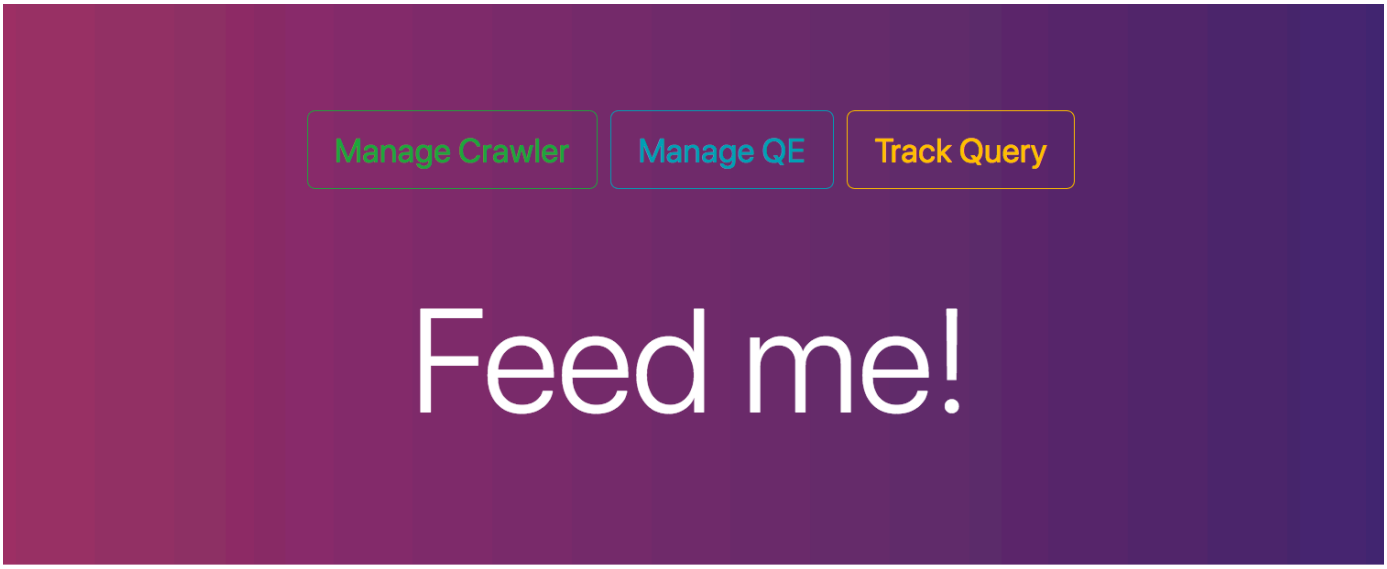
[1] Day trading. (2018). Retrieved from https://en.wikipedia.org/wiki/Day_trading#Market_data
[2] Bloomberg L.P. (2018). Retrieved from https://en.wikipedia.org/wiki/Bloomberg_L.P.
[3] Bloomberg L.P. | About, Products, Contacts. (2018). Retrieved from https://www.bloomberg.com/company/?utm_source=bloomberg-menu&utm_medium=terminal
[4] Query expansion. (2018). Retrieved from https://en.wikipedia.org/wiki/Query_expansion
[5] Manning, C., Raghavan, P., & Schütze, H. (2009). *An Introduction to Information Retrieval (Online edition (c) 2009 Cambridge UP)* (pp. 189-193). Cambridge: Cambridge University Press.
[6] Relevance feedback. (2018). Retrieved from https://en.wikipedia.org/wiki/Relevance_feedback#Blind_feedback
[7] Hadgu, A., Abualhaija, S., & Niederée, C. (2018). Sover! Social Media Observer. *SIGIR '18 The 41St International ACM SIGIR Conference On Research & Development In Information Retrieval*, *41*, 1305-1308. doi: 10.1145/3209978.3210173
[8] Play Framework. (2018). Retrieved from https://en.wikipedia.org/wiki/Play_Framework
[9] Model–view–controller. (2018, August 22). Retrieved September 3, 2018, from https://en.wikipedia.org/wiki/Model–view–controller
[10] Burgess, M. (2018). JavaScript Frameworks — Why to use them at all – Matt Burgess – Medium. Retrieved from https://medium.com/@mattburgess/javascript-frameworks-why-to-use-them-at-all-8cce2804c092
[11] JS Foundation, J. (2018). jQuery. Retrieved from https://jquery.com/
[12] Introduction — Vue.js. (2018). Retrieved from https://vuejs.org/v2/guide/
[13] Bootstrap. (2018). Retrieved from https://getbootstrap.com/
[14] Intra-day Data - amCharts. (2018). Retrieved from https://www.amcharts.com/demos/intraday-data/
[15] Bostock, M. (2018). D3.js - Data-Driven Documents. Retrieved from https://d3js.org/
[16] Rate limits. (2018). Retrieved from https://developer.twitter.com/en/docs/basics/rate-limits.html
[17] News API - A JSON API for live news and blog articles. (2018). Retrieved from https://newsapi.org/
[18] Pricing - News API. (2018). Retrieved from https://newsapi.org/pricing
[19] ModuleDirectory - 2.6.x. (2018). Retrieved from https://www.playframework.com/documentation/2.6.x/ModuleDirectory#datastore
[20] What Is MongoDB?. (2018). Retrieved from https://www.mongodb.com/what-is-mongodb
[21] ReactiveMongo - Home. (2018). Retrieved from http://reactivemongo.org/
[22] Why is Scala recommended over Java in Play Framework 2?. (2015). Retrieved from https://www.quora.com/Why-is-Scala-recommended-over-Java-in-Play-Framework-2
[23] Case Classes | Scala Documentation. (2018). Retrieved from https://docs.scala-lang.org/tour/case-classes.html
[24] ScalaTemplates - 2.6.x. (2018). Retrieved from https://www.playframework.com/documentation/2.6.x/ScalaTemplates
[25] Roth, T. (2018). Force Directed Graphs with d3 version 4. Retrieved from http://www.puzzlr.org/force-graphs-with-d3/
[26] ScalaActions - 2.6.x. (2018). Retrieved from https://www.playframework.com/documentation/2.6.x/ScalaActions
[27] Results. (2018). Retrieved from https://www.playframework.com/documentation/2.6.x/api/scala/index.html#play.api.mvc.Results@values
[28] ScalaDependencyInjection - 2.6.x. (2018). Retrieved from https://www.playframework.com/documentation/2.6.x/ScalaDependencyInjection#Motivation
[29] Futures and Promises | Scala Documentation. (2018). Retrieved from https://docs.scala-lang.org/overviews/core/futures.html
[30] Strategy pattern. (2018). Retrieved from https://en.wikipedia.org/wiki/Strategy_pattern
[31] JavaAkka - 2.6.x. (2018). Retrieved from https://www.playframework.com/documentation/2.6.x/JavaAkka
[32] ScalaFunctionalTestingWithScalaTest - 2.6.x. (2018). Retrieved from https://www.playframework.com/documentation/2.6.x/ScalaFunctionalTestingWithScalaTest
[33] ScalaTest. (2018). Retrieved from http://www.scalatest.org/user_guide/using_matchers
[34] Assertion Testing. (2018). Retrieved from https://www.tutorialspoint.com/software_testing_dictionary/assertion_testing.html

Appendix A Initial UI Design



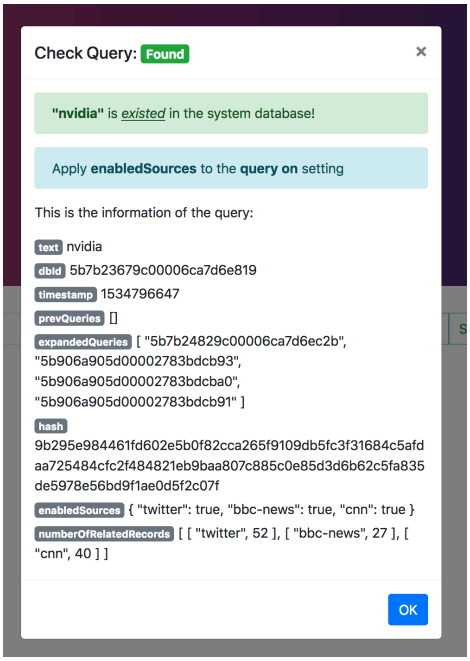
Appendix B Screenshots

Index page



Index page

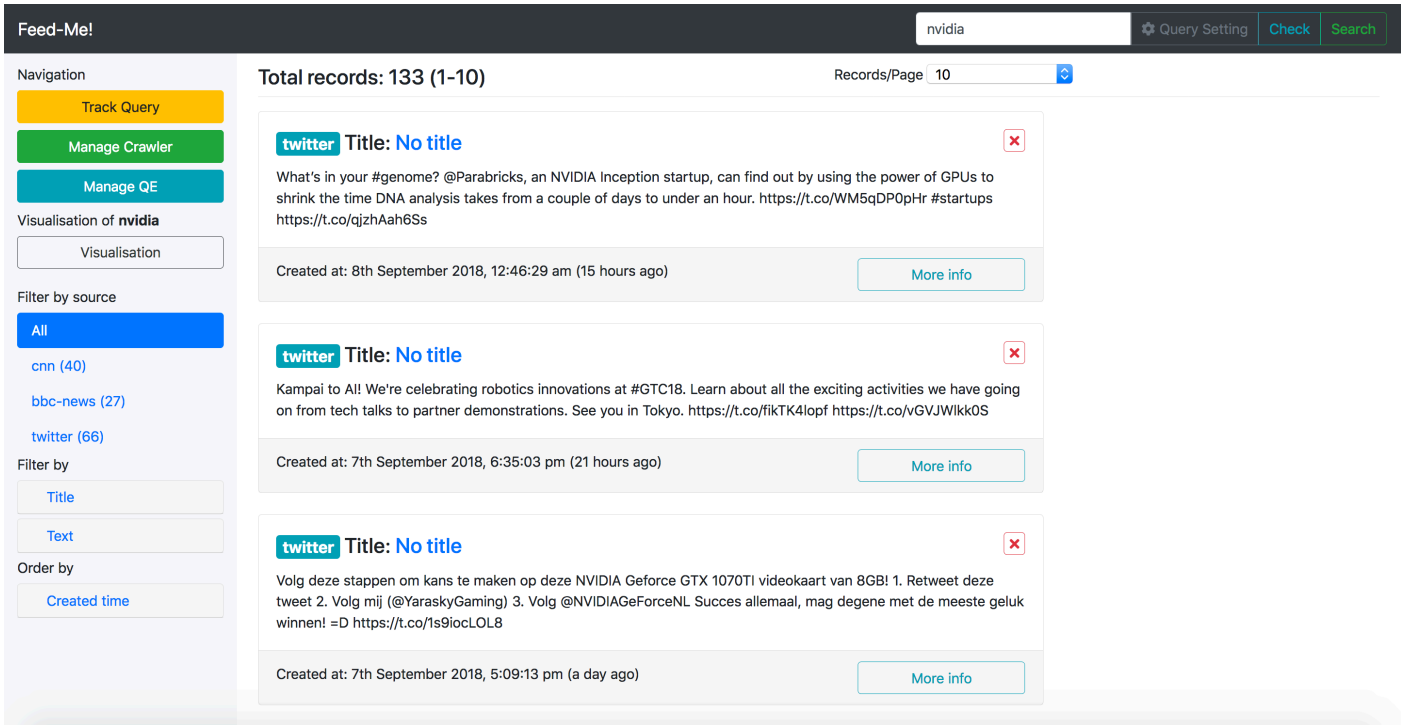
Users can navigate to other main views using the top navigation, checking if the query is existed in the database using check button, change query setting from the blue hyperlink and querying by entering text in search box then click search.



Check modal

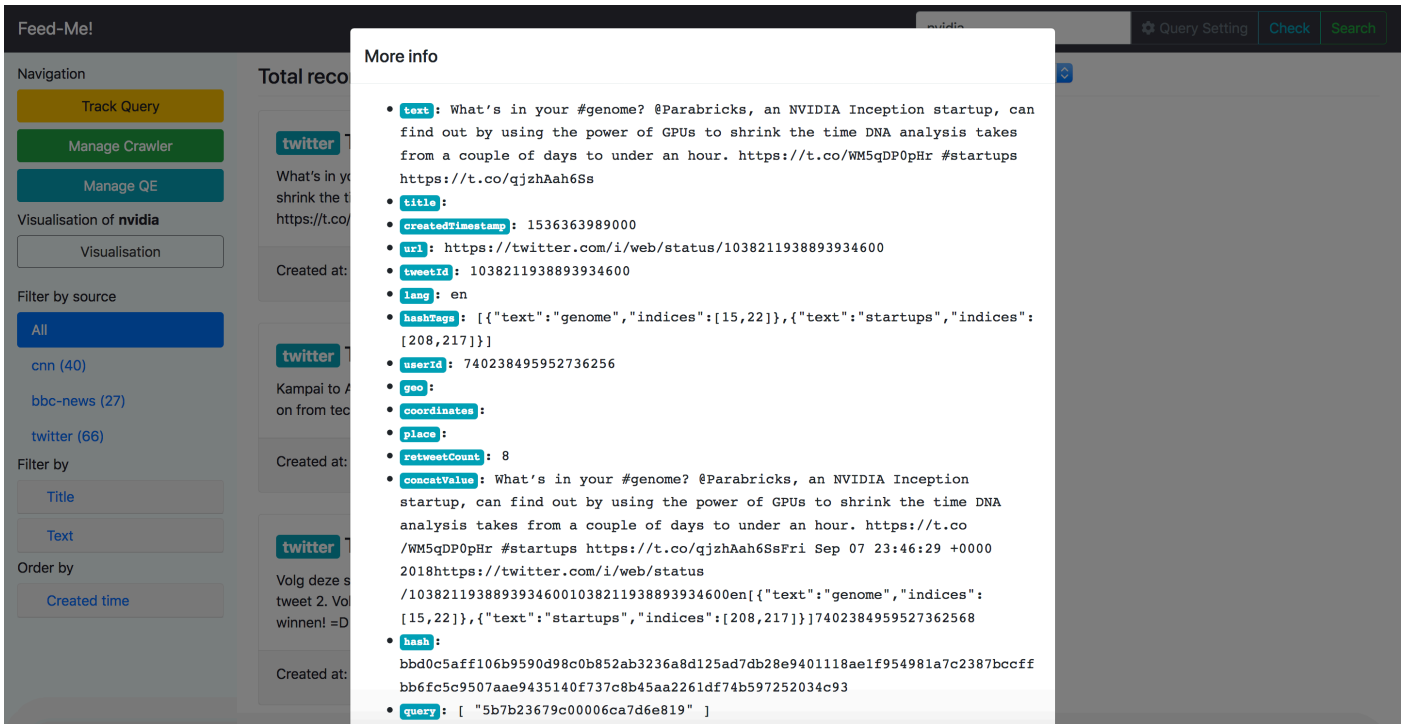
After clicking check query button, the modal appears. It shows the status of that query and some of related information.

Result page



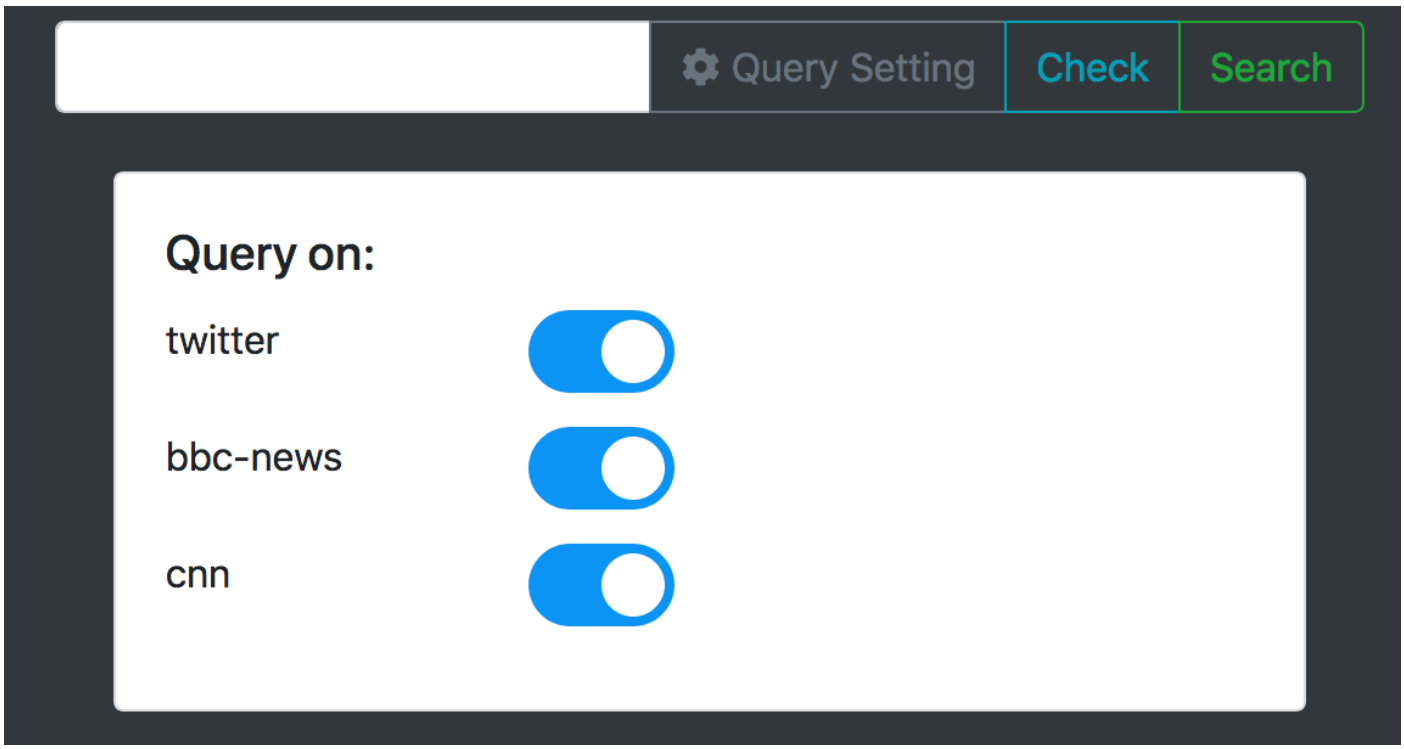
Result page

After querying, the result page is shown. There is a side bar on the left, consisting of navigation buttons, button linked to the visualisation page of a query, filtering and ordering menu. It shows the related records, each in a card, of the query. Records are queried from a new search and existing records in the database. Users can delete a record if it is duplicated or irrelevant and can click at more info button to see more information of that record. More info modal is shown after clicking.



More info modal

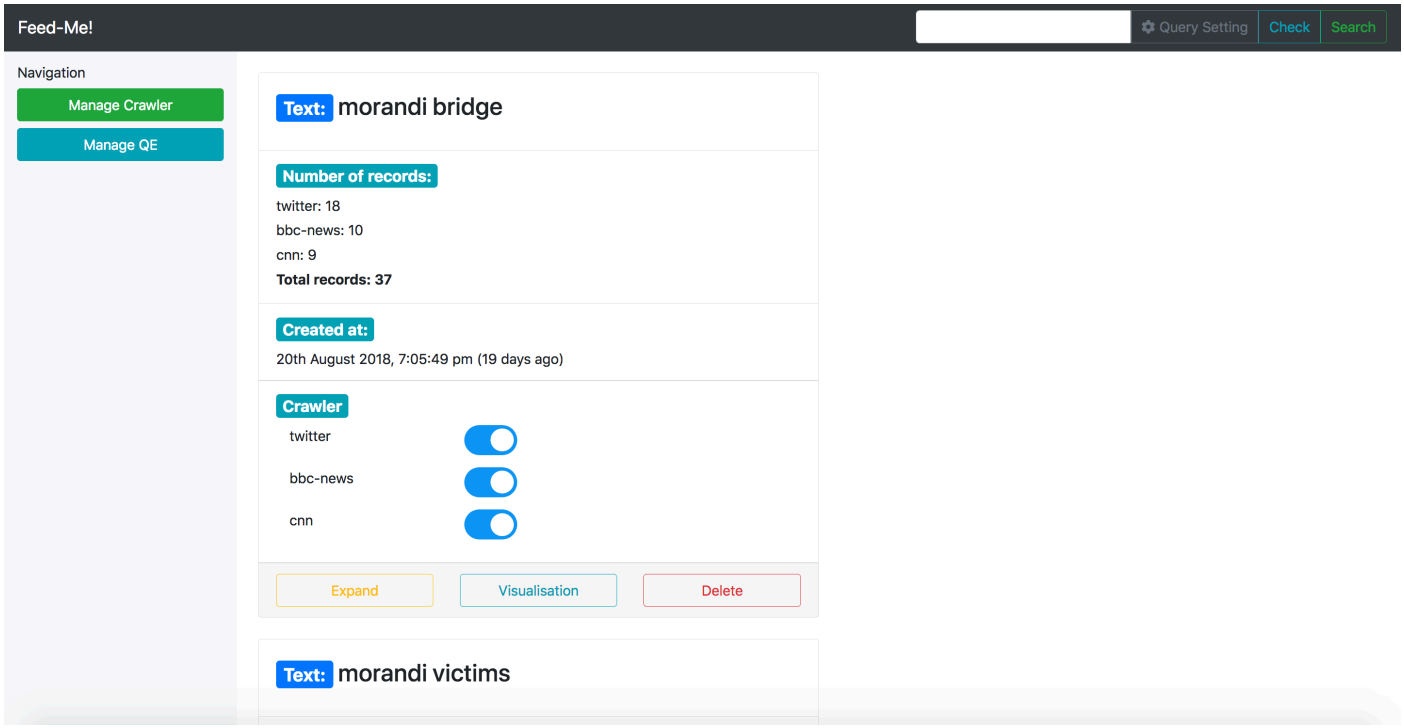
Search bar



Search bar

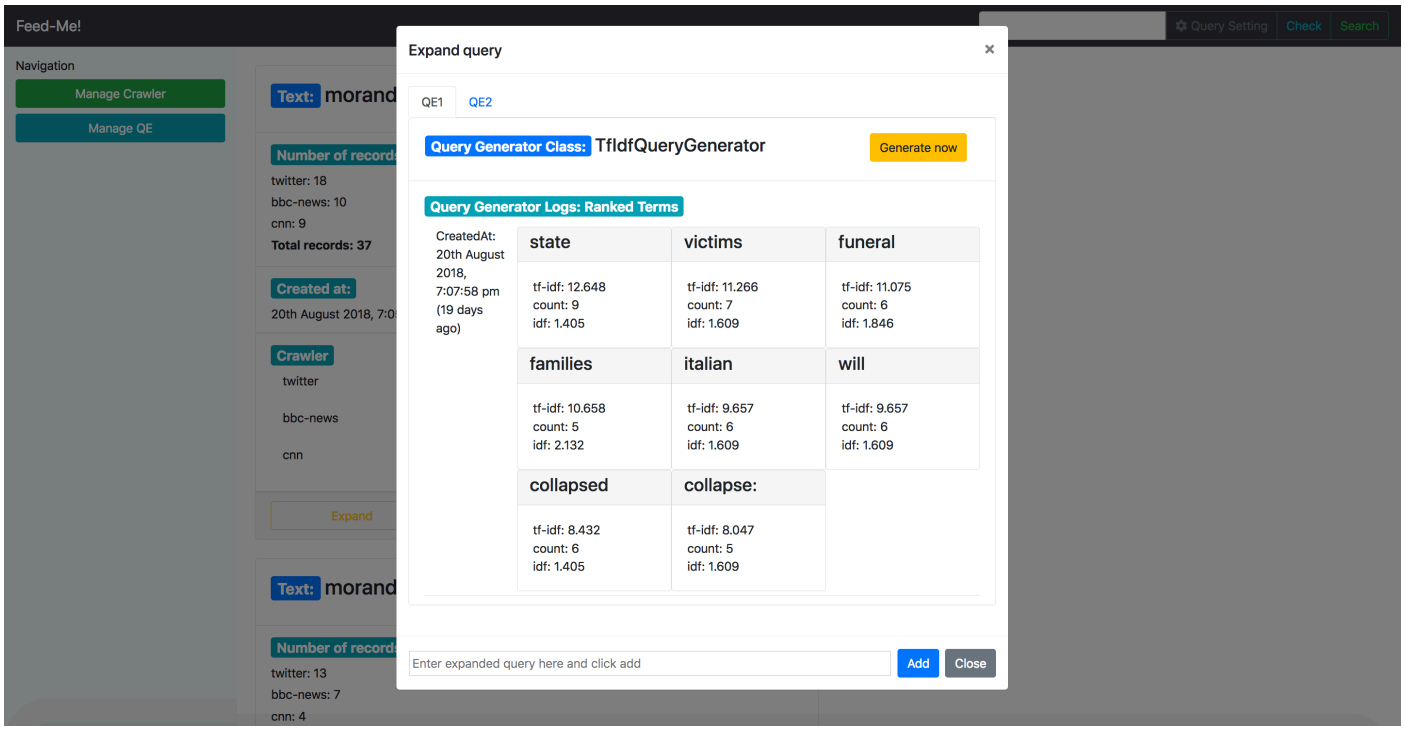
Search bar appears on the top of all pages, except index which is located in the center. Users could enter a query and click check or search from here. In addition, it shows the list of crawlers by clicking at query setting. This allows users to alter the statuses of crawlers of that query.

Track query page



Track query page

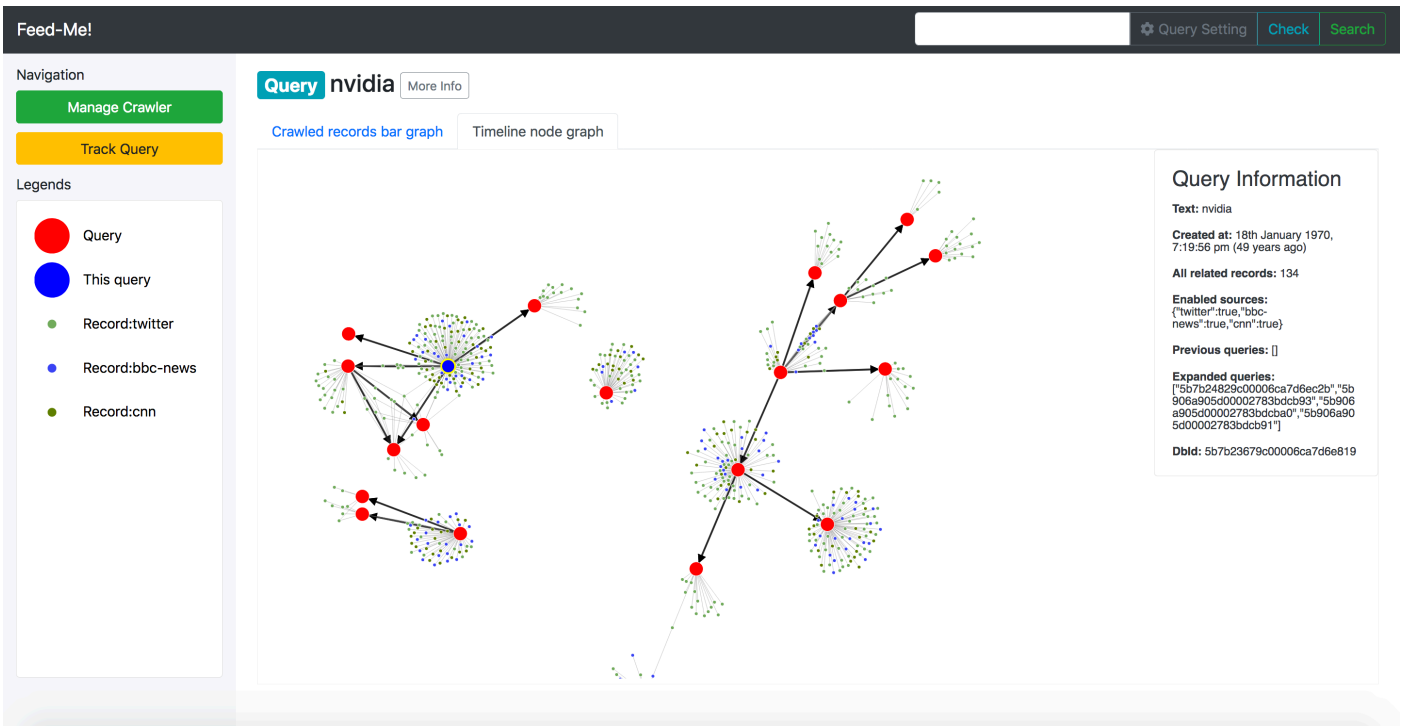
This page lets users see the list of tracked queries listed each in a card. Users can see number of records, creating date and time, and the query setting (annotated by crawler). They can also delete a query, see the visualisation page of that query and perform manual expansion on the query. By clicking expand button, the manual expansion modal will be shown. Users will see the available query expansion units listed in the tabs. They could select any of them to see helpful information generated by the unit at a specific time, showing ranked terms with their text statistics. Additionally, they can force the unit to generate information for them now. To perform a manual expansion, users enter the expanded query and click add.



Expand query modal

Visualisation page

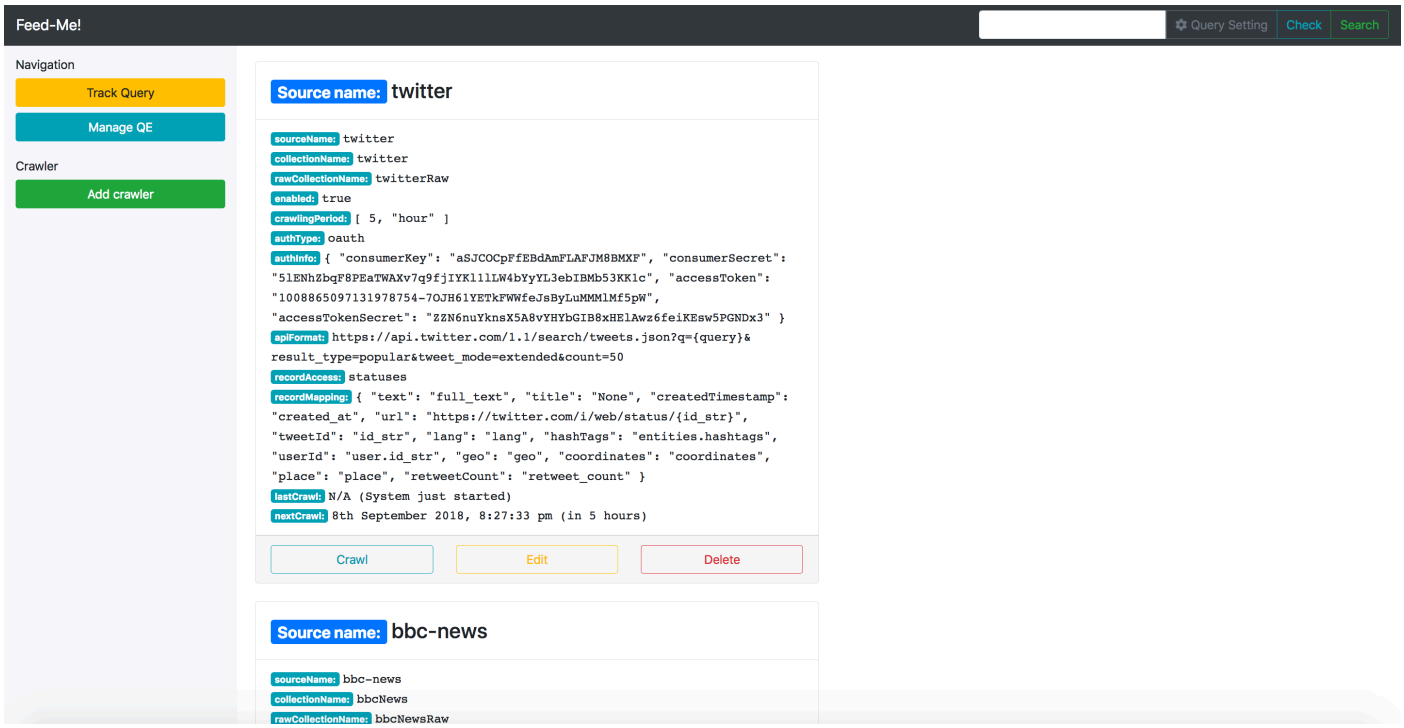
Visualisation page: Bar graph



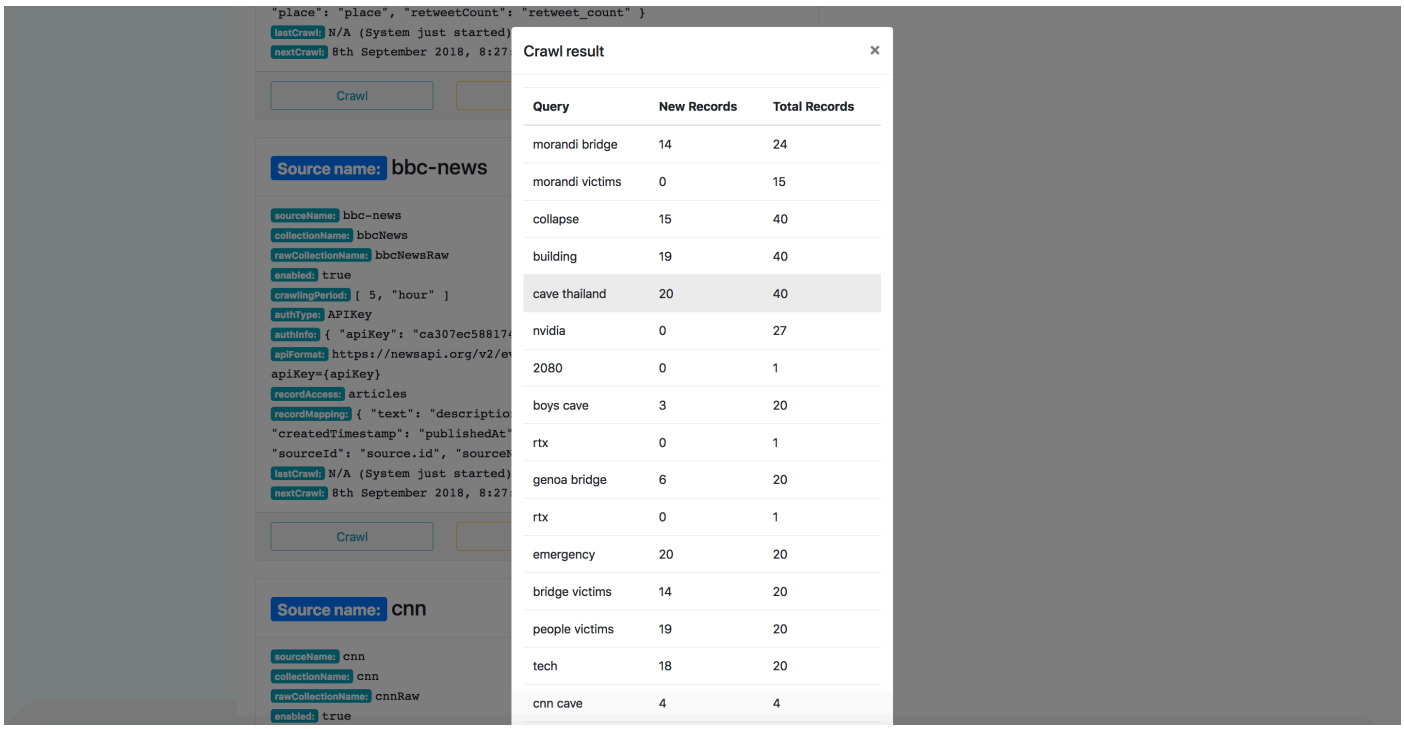
Visualisation page: Timeline graph

After clicking at the visualisation button in a result page or track query page, the users will be navigated to visualisation page of that specific query. There are 2 graphs available. Users can change the graph by clicking the tabs above. For timeline graph, there is a legend description on the left. As stated in the main report sections, the graph shows relationships between queries and records. They can also drag and zoom the graph if they want. When clicking at a node (any query or record), information box will appear on the top right of the graph.

Manage crawler page



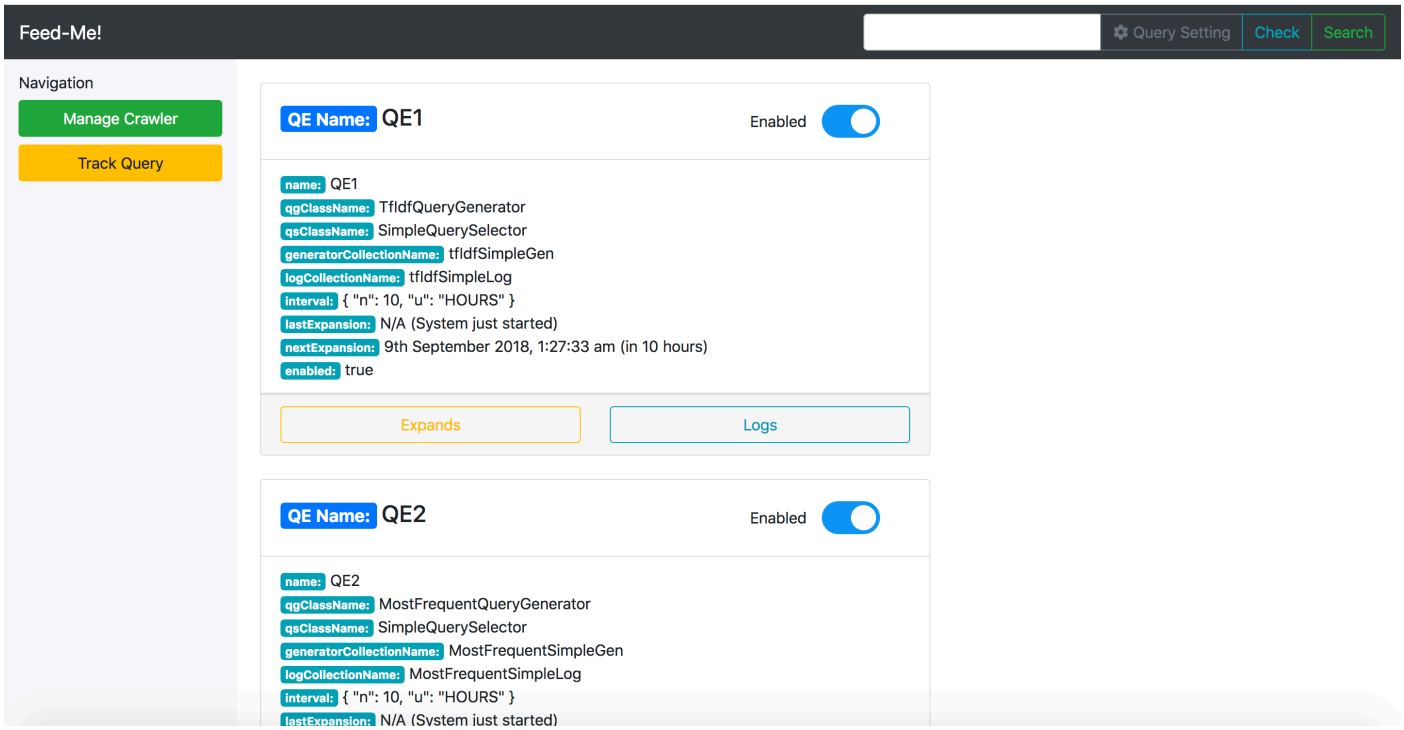
Manage crawler page



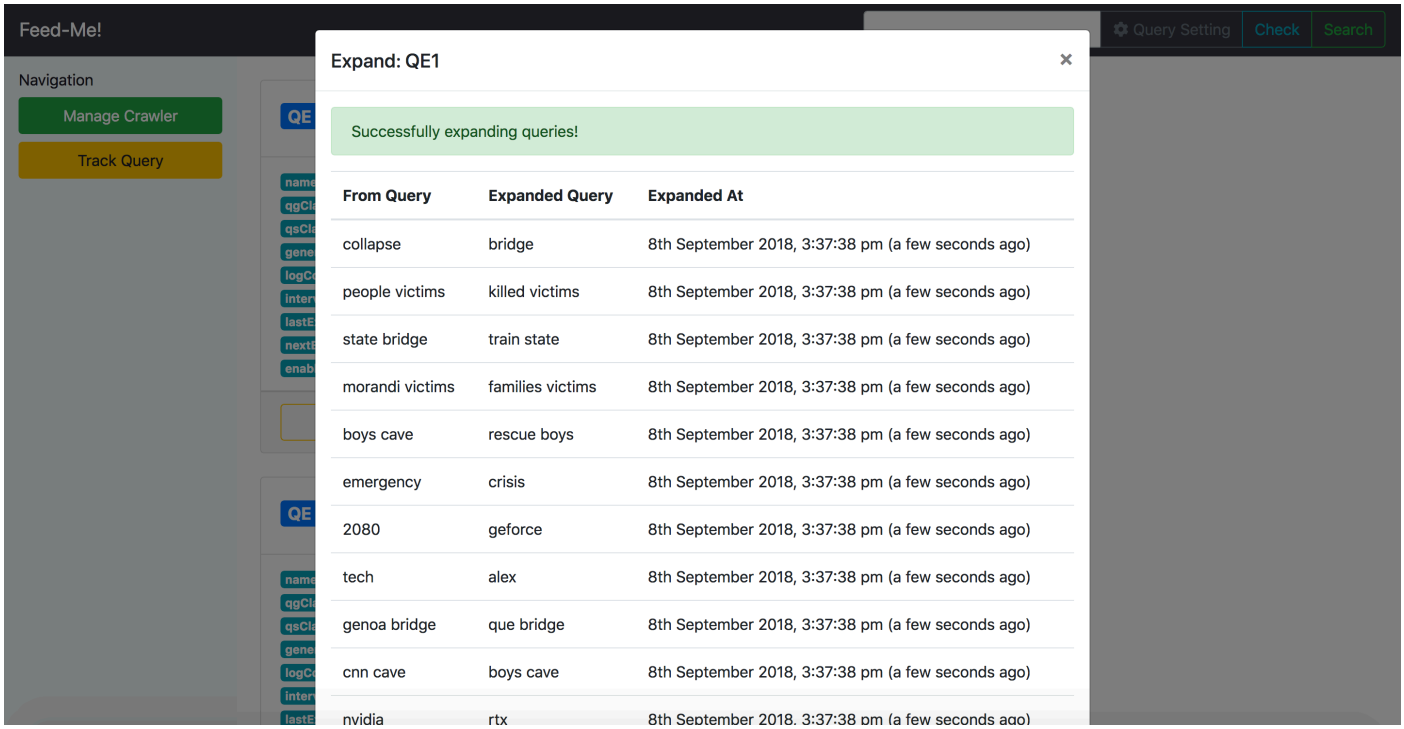
Crawling result modal

The page shows a list of crawlers, listed each in a card. A card shows information of the specific source of a crawler. Users can edit or delete a crawler and add new crawler by clicking at the green add crawler button in side bar. By clicking crawl button of a crawler, it forces the crawler to start a crawling on every tracked query immediately. The crawler would skip queries that have turned this crawler off in query setting.

Manage QE page



Manage QE page



Expansion log modal

This page is very close to a manage crawler page. Users see the list of available query expansion units. They could see the past automatic expansion logs of that unit or force the unit to perform automatic expansion immediately.

Appendix C Detail of Software Test Cases

Feature	Step	Test cases no.	Description
Query	Record mapping	1	Handle normal mapping without any special cases (accessing object or accessing array), the example is a mapping from "full_text" attribute in response to "text" attribute in mapped record
		2	Handle mapping with object accessing, the example is a mapping from "`entities.hashtags`", where "entities" is an object and "hashtags" is a field in the entities, to "hashtags" attribute in mapped record
		3	Handle mapping with array accessing, the example is a mapping from "`entities.hashtags[0]`", where [0] indicates the first element of hashtags array, to "hashtags" attribute in mapped record (the difference between this and above case is that only an element will be kept while the 2nd case kept whole array)
	Record comparison	1	No different record between new mapped records from a query and current records in database collection, so new records will be simply added to the current database collection
		2	There is a duplicate record in new mapped records from a query and current records in database collection. The query id will be added to queries field of duplicate record in collection (assume that the query is not existed in the queries field yet).
		3	There is a duplicate record in new mapped records from a query and current records in database collection. However, the query is already existed in queries field of the duplicate record. So, this duplicate record will not be updated.
Query expansion	Insert expanded query	1*	There are 2 queries existed in the database, then insert a query 3 which expanded from query 1. Query 1 is expanded to query 3.
		2*	There are 3 queries existed in the database, then insert a query that has the same text with query 3 which expanded from query 2. Query 1 and query 2 is expanded to query 3 and there should still be 3 queries in the database.
		3*	There are 3 queries existed in the database, then insert a query that has the same text with query 3 which expanded from query 1. This query is duplicated, and every query should not be changed.

* Every test case has its own setup, such as clearing and adding mock data to the test database, before starting except these 3 test cases in query expansion. They perform sequentially from one to another.

Appendix D User Evaluation Tasks

#	Requirement	Goal	Success measurement
		A. Index and Result page	
Background: You are going to find the information of event “A” and “nvidia”. However, you cannot remember whether the events are tracked in the system or not. Then, you query for “A” after knowing that “nvidia” is already existed and tracked.			
1	A4	Check if the query “A” and “nvidia” are existed in the system	Response from the check button, correct if query “A” is not existed and “nvidia” is existed
2	A1	Look for overview information of event A by querying “A”	Evaluator’s overview information is correct
Background: Result page shows list of related records, but most top records do not have title (N/A). So, you try to filter out those records to list only records with title and change the number of records per page to 100.			
3	-	Show only the records with title and change the number of records per page to 100	The number of shown records per page is changed and show only records with title
Background: There are several irrelevant and duplicated records from “twitter” source. It is a good idea to perform data cleansing by removing some of those records.			
4	A13	Remove irrelevant record(s)	Irrelevant record(s) is removed
Background: As you can see, most records from “twitter” are duplicated and irrelevant. You want to disable the “twitter” source of this query. So, there would be no records coming from this source in the next querying/crawling.			
5	A5	Disable a “twitter” crawler and search for the event “A” again	The number of disabled records must not change
		B. Track Query	
Background: You wonder how query "nvidia" is being tracked in the system and would like to see the number of related records. Since event "nvidia" is popular on Twitter, you find that there are few records from "bbc" and "cnn" sources. So, you are going to turn off "bbc" and "cnn" crawler on query "nvidia".			
1	A2, A6	Find the number of related records of query "nvidia"	Correctly identify number of related records of query "nvidia"
2	A5	Turn “bbc” and “cnn” crawlers off on query "nvidia"	"bbc" and "cnn" are turning off using the switches
Background: You have an idea to perform manual expansion on query "A". However, query "A" was just being tracked few minutes ago and it is still not the time for automatic expansion to generate the information for manual query expansion yet. You notice that there are 2 QE units and are curious to see statistics form both of them. Then, you want to force both of them to generate the information immediately.			
3	D1	Force query expansion units to generate information for manually entering expanded query	Successfully click at the "Generate now" button
Background: After seeing ranked terms from both QE units, wondering which one provides more reasonable information. Then, according to the information from both QE units, you perform manual query expansion on query "A" with query "A2".			
4	-	Compare which QE unit has better performance	Ideally more advance QE unit should be the answer but depending on the reasons given by an evaluator
5	D1	Manually expand the query "A" with query "A2" according to the generated ranked terms	Input the expanded query according to the information given by QE
Background: You find that event "cave thailand" is very old and would like to delete it.			

6	A3	Delete query "cave thailand"	Successfully delete query "cave thailand"
C. Visualisation			
Background: You want to see the visualisations of query "morandi bridge" and would like to know the number of records from each source that issued on a specific date			
1	C1	Bar graph: Find the number of records from each source on 18/08/2018	Correctly specify number of related records for each source on the specific date
Background: Switching to the timeline graph, you see that query "morandi bridge" has several expanded queries. Moreover, there are some records that links with 2+ queries and you wonder why they have such relationships. Also, you would like to see the source web page of those records.			
2	C2, C3	Timeline graph: Identify the records that have multiple links and interpret their relationship between queries	Provide any logical reasons about their relationships given the information in infobox
3	C2, C3	Timeline graph: Pick a record and link to the source url of that record	Click at the URL link in the infobox of that record
D. Manage Crawler			
Background: You would like to add a new crawler called "google-news" given the information provided on this task sheet			
1	B1	Add new crawler to the system	New crawler is added to the system
Background: You would like to know when the next crawling time and how many hours/minutes is from now. Then, you figure out that the entered crawling period is incorrect. So, you change it to the correct one. (from 3 hours to 1 hour)			
2	B4	Figure out when is the next crawling of a specific crawler	Identify a correct time of the next crawl
3	B3	Change a crawling period of a crawler	Crawling period is changed
Background: You are exciting and impatient to see the result of this crawler. So, you force google-news crawler to start before the schedule.			
4	B5	Force a crawler to start before the schedule	Number of new crawling records are shown in the modal
Background: You found that there are few records from this newly added crawler and you would like to delete it because it is not efficient.			
5	B2	Delete a crawler	Crawlers listed are changed and the crawler is removed from the list
E. Manage Query Expansion			
Background: You were impressed by the performance of "QE1" from recent manual expansion of query "A" and would like to see when the next expansion will be performed. Also, you want to look at recent expansion logs of this QE unit too.			
1	-	Figure out when is the next query expansion	Identify a correct time of the next expansion
2	-	Look for the recent query expansion information in logs	List of logs are shown in the QE logs modal
Background: You don't see any expansion from query "A" and would like to see how automatic expansion works. So, you force "QE1" to expand now.			
3	D3	Force a QE unit to perform query expansion before the schedule	Number of expansion logs are shown in the modal

Appendix E Questionnaire section 1: Rating

1. System overall
    a. *Ease of use:* Was the system easy to use?
    b. *Feasibility:* Do you think that the system would be worked in practice?
    c. *Directions/guides within the system:* Do you think that there are plenty of guides that help you easily understand the system?
    d. *Responsiveness:* Was the system fast enough to response you as expected? Do you experience any lags?
2. Visualisation: Bar graph
    a. *Insight gaining from the graph:* Do you feel that the graph provides useful information vs. raw data?
    b. *Ease of understand:* Was it easy for you to understand the graph?
3. Visualisation: Timeline graph
    a. *Insight gaining from the graph:* Do you feel that the graph provides useful information vs. raw data?
    b. *Ease of understand:* Was it easy for you to understand the graph?
4. Expansion
    a. *Feasibility:* Do you think that this feature would be worked in practice?
    b. *Manual expansion:* Did generated information help you decide the query terms for manual expansion?
5. Evaluation
    a. *Directions:* Do you think that the evaluation procedure is connected flawlessly?
    b. *Enjoyment:* Did the evaluation entertain you in any aspect?