

Faceted Navigation for Open Data: Using Elasticsearch for offenesparlament.at

Group 5 Mathias Blum, Matthias Frey, Daniel Lamprecht and Nelson Silva

706.041 Information Architecture and Web Usability WS 2015
Graz University of Technology

01 Feb 2016

Abstract

This report describes an example implementation of a faceted user interface. The implementation makes a large-scale open government dataset from the Austrian parliament accessible and explorable by means of facets. Facets include information about the speech (such as text or date) and about the speaker (such as name and political party affiliation). Combining facets makes it possible to do research in the dataset, which is further aided by several visualizations. Besides demonstrating the power of facets for data exploration, this project also makes available an extendable framework, demonstrating state-of-the-art technologies to build a faceted interface. The implementation relies on Elasticsearch as a server and provides two approaches for the client side: a version based on ElasticUI (a set of powerful AngularJS directives) and a version implemented with elasticsearch.js (the API of Elasticsearch). Finally, this report includes a manual explaining how to set up the project.

Contents

Contents	i
List of Figures	iii
List of Tables	v
List of Listings	vii
1 Introduction and Motivation	1
1.1 Demonstration of a Faceted Search User Interface	1
1.2 Demonstration of Available Technologies	1
1.3 Extensibility for Future Work	1
2 Dataset and Faceted Interface	3
2.1 Dataset: Open Government Data from the Austrian Parliament	3
2.2 A Faceted User Interface for Data Exploration	5
2.2.1 Front-End Application with ElasticUI	5
2.2.2 Front-End Application with elasticsearch.js	8
3 Implementation	11
3.1 Application Architecture	11
3.2 Search Server: Elasticsearch	11
3.3 Front-End Application	11
3.3.1 ElasticUI Client	12
3.3.2 Elasticsearch.js Client API	12
3.3.3 Widget Model	14
3.3.4 Visualizations	14
4 User Manual	17
4.1 Requirements	17
4.2 Building and Using the Application	17
4.2.1 Running And Configuring Elasticsearch	17
4.2.2 Installing Frontend Requirements	18
4.2.3 Building and Running the Frontend	18
4.2.4 Using a Different Webserver	18
4.3 Indexing Data	18
4.3.1 Importing Data Into an Index	18
4.3.2 Usage of a Mapping File	19
Bibliography	21

List of Figures

2.1	The Front-End Application with ElasticUI	6
2.2	The Front-End Application using elasticsearch.js	7
2.3	Date Histogram and and Speech Diagram	8
2.4	Visualization of the Most Frequently Used Words	9
3.1	Application Architecture	12
3.2	Widget Model	15

List of Tables

2.1	Overview of the Features Supported by the Two Approaches	5
-----	--	---

List of Listings

2.1	Example of a Speech from Dataset	4
3.1	ElasticUI Code Example	13
3.2	Elasticsearch JavaScript Client Code Example	14
3.3	Request to Backend Containing QueryDSL	15
4.1	Initialise and Run Frontend Project	18
4.2	Elasticsearch Bulk API Import Statements	19
4.3	Elasticsearch Mapping File Example	19
4.4	Import Data Into Elasticsearch Using cURL	20

Chapter 1

Introduction and Motivation

Over the last years, faceted user interfaces have become more and more prevalent. Web sites with faceted user interfaces now span a broad range of topics, from web stores (such as Amazon or Nordstrom) to price comparison (Geizhals) and a wide variety of others (e.g., Yelp, Airbnb or the TU Graz library). This report describes an example implementation of a faceted search user interface and describes the motivation, dataset, the features and capabilities and technologies used to build it.

1.1 Demonstration of a Faceted Search User Interface

Accessibility and exploration of large and complex datasets can be a daunting task for users, especially for novices. Early search interfaces often used Boolean expressions, which, however, required users to be familiar with the syntax. In recent years, the complexity of Web search has generally been decreasing. Many (if not most) users now expect to be able to use a search interface the way they use a typical Web search engine such as Google, that is, by typing in one or more keywords and not using any special syntax.

The driving factors behind this project were the wish to make a large open-government dataset accessible and the desire to build an example implementation of a faceted search user interface. By making a range of facets explicitly available in the interface, this project makes a large collection of open-government data accessible and explorable.

Faceted user interfaces offer a simple method to overcome the complexity threshold and offer users a more simplistic way of exploring large datasets [Tunkelang, 2009, Chapter 3.1]. These interfaces offer guidance by showing facets—features of the dataset along multiple, orthogonal dimensions—that allow users to narrow down the search space. Chapter 2 describes the features and capabilities of the interface implemented for this project in detail.

1.2 Demonstration of Available Technologies

An important motivation for this project were the demonstration of available state-of-the-art technologies to build faceted user interfaces. A first phase of the project therefore included surveying technologies, the results of which are detailed in [Blum et al., 2015]. This previous work formed the basis for the decision of what technology to use for this project. Chapter 3 details the technologies and frameworks used to implement this project.

1.3 Extensibility for Future Work

Finally, this project also placed importance on extensibility for future work. For this to be possible, the project makes use of AngularJS [Google, 2016], an *MVVM* (Model View View Model framework), many times referred also as an *MVW* (Model-View-Whatever works for you framework). AngularJS allows to structure application

functionality by taking data as a first-class citizen. The combination of AngularJS and elasticsearch.js (an API to allow the handling of the Elasticsearch main functionalities) allowed for the application to include all the features required by a faceted user interface. At the same time, AngularJS and elasticsearch.js make it very straightforward to extend the application by allowing to easily write and plug in new methods and capabilities. To facilitate setup and development, Chapter 4 contains a detailed setup and usage guide that allows to get the project installed and running with as little effort as possible.

Chapter 2

Dataset and Faceted Interface

This project was fortunate to work with a large and novel dataset, which stems from preliminary work of an Austrian open government project. The faceted search Web application implemented for this project offers an interface that makes the dataset accessible and explorable. As an example, the interface allows the investigation and visualization of terms that sparked public interest at certain points in time, such as *BAWAG*, a term which skyrocket in popularity after the affair concerning the Austrian bank became public.

2.1 Dataset: Open Government Data from the Austrian Parliament

One of the motivating factors behind this project was to make a large-scale dataset accessible and explorable. The dataset the user interface was implemented for was therefore chosen as an open government dataset from the *National Council* (Nationalrat), the lower chamber of the Austrian Parliament. By the Austrian constitution, the National Council is the more powerful of the two chambers that make up the Austrian parliament, with the other being the Federal Council (Bundesrat).

The dataset used for this project consisted of all speeches by members of the National Council as well as members of government from the 22nd, 23rd and the 24th legislative periods, therefore ranging from the end of 2002 through the end of 2013. This time frame is especially interesting in the context of Austrian politics, as it encompasses the first conservative and right-wing coalition in the 22nd period, a time of great public interest in the country. The period is followed by the (for Austria) more common conservative and social democrats coalition in the 23rd and the 24th periods.

The transcripts for the speeches are available from the Web site of the Austrian parliament [[Parlamentsdirektion, 2016](#)], where the raw data is available as debate protocols in HTML. Downloading and parsing these transcripts was not directly part of this project. However, one of the authors (Matthias Frey) contributes to the *OffenesParlament.at* project [[Offenes Parlament, 2016](#)] and was able to make the data available preliminarily for use in this project. The source code for the scraper is available on GitHub [[Informationsfreiheit, 2016](#)]. While great care was taken in the scraping and preprocessing of the data, the dataset does contain a few irregularities such as speech durations of zero seconds for very short speeches. With more effort, these irregularities could certainly be overcome. This was, however, out of the scope of this project.

From the SQL database of *OffenesParlament.at*, the data was then exported with a script and saved as a JSON file. For easier handling by the Web application implemented for this project, the data fields about the speech and the corresponding speaker were joined into a single JSON object.

Listing 2.1 shows an example of a speech by a member of the conservative party and displays the properties associated with the speech (such as date, duration or number of words) and the speaker (such as political party affiliation, birth date or occupation). The data also included a link to a photo of the speaker on the Web site of the parliament, which was used in the interface.

```

1
2 {
3   "current_party": "ÖVP",
4   "name": "Dr. Andrea Eder-Gitschthaler",
5   "text": "Sehr geehrte Frau Präsidentin! Sehr geehrter Herr Minister! Hohes Haus! Wenn
        wir uns hier umschauen, sehen wir überall Laptops. Wir alle sind Internet-User
        mehr oder weniger professioneller Art. Ohne WWW, E-Mails et cetera könnte ich
        mir zum Beispiel meinen Berufs- und Politikeralltag gar nicht mehr vorstellen.
        Und schon gar nicht unsere Kinder – die sind ja quasi schon mit dem Laptop und
        dem PC auf die Welt gekommen und haben Gott sei Dank keine Scheu mehr, sich
        dieser Mittel zu bedienen.\n\nDiese zunehmende Popularität des Internets, die
        steigenden Nutzerzahlen und die vermehrte Verwendung des World Wide Web zur
        Information wie auch zur Unterhaltung sind grundsätzlich positiv, haben aber zu
        diesen – wie wir heute schon mehrmals gehört haben – großen und gewaltigen
        Problemen geführt.\n\nLeider wissen diese schier unbegrenzten Möglichkeiten
        gewissenlose Geschäfte-macher und Betrüger für ihre Zwecke zu nützen. Scheinbar
        kostenlose Angebote – ich habe ein paar solche hier, zum Beispiel www.gedichte-
        heute.com, www.geldverdienen-heute.com, www.hausaufgaben-heute.com – sind alles
        andere als gratis. Teure Abos werden mit einem Mausklick abgeschlossen, und es
        drohen, wie wir heute auch schon mehrmals gehört haben, völlig ungerechtfertigte
        Zahlungen.\n\nBetroffen sind eben die Jungen. Die Eltern kommen dann zu uns,
        und wir müssen ganz gezielt Hilfestellung, Unterstützung anbieten, denn es sind
        dann meistens auch sehr prekäre finanzielle Situationen, in denen sich diese
        betroffenen Jugendlichen befinden. – Dieser Entwicklung gilt es entschieden
        entgegenzuwirken.\n\nIch bin sehr froh über den hier vorliegenden Antrag von
        Kollegem Maier und Kollegem Rädler – dem ich von dieser Stelle aus nochmals ganz
        herzlich zum Geburtstag gratulieren möchte –, Stichwort Button-Lösung, und ich
        kann Sie daher nur um Ihre Unterstützung und Ihre Zustimmung ersuchen. – Danke.
        ",
6   "wordlen": 271,
7   "llp": 23,
8   "session_nr": 56,
9   "birthdate": "1961-09-13",
10  "duration": 108,
11  "role": "abg",
12  "date": "2008-04-10T16:44:12+00:00",
13  "deathdate": null,
14  "image": "http://www.parlament.gv.at/WWER/PAD_35471/2101829_384.jpg",
15  "deceased": false,
16  "occupation": "Versicherungsangestellte"
17 }

```

Listing 2.1: Example of a speech by Dr. Andrea Eder-Gitschthaler from the dataset. The listing shows the properties associated with the speech (such as date or duration) as well as properties of the speaker (such as political party affiliation or birthdate).

Feature	ElasticUI	elasticsearch.js
Speaker Name Search	Yes	No
Speech Term Search	No	Yes
Occupation	Yes	Yes
Political Party	Yes	Yes
Speaker Role	Yes	Yes
Speech Duration	Yes	Yes
Date	No	Yes
Legislative Period	No	Yes
Visualizations	No	Yes

Table 2.1: An overview of the features supported by the two approaches for a faceted search interface. The approach based on elasticsearch.js allowed for more fine-grained control of the connection with Elasticsearch and therefore supported a larger number of features.

2.2 A Faceted User Interface for Data Exploration

This project includes two interfaces:

1. An interface based on ElasticUI [ElasticUI, 2016], which allowed for rapid development but less control over the interface and
2. An interface implemented based on the elasticsearch.js API, which requires more effort in development but allows for fine-grained control over the interface

Both interfaces initially show the set of all parliamentary speeches in the dataset. However, due to their large number, the speeches are not very accessible to exploration and research. For this reason, the faceted search interfaces on the left side of the pages provide help and allow to specify free-text keywords and to narrow down the search space using a range of facets. Table 2.1 gives an overview of the features supported by the two approaches. The following subsections describe the two approaches in more detail, while the details of the implementation are reported in Chapter 3.

2.2.1 Front-End Application with ElasticUI

Figure 2.1 shows the application implemented with ElasticUI. The interface is divided in two parts: a sidebar on the left shows the search interface and the facets. The content section on the right shows the speeches.

The free-text search box on the left hand side permits to search a term (in this case the complete name of a speaker), which is then queried against the names of all speakers in the dataset and in turn restricts the shown speeches on the right to those matching the searched speaker name. As such, the *speaker name* is a facet—a view of the dataset—and can be used to narrow down the search space.

In addition to the search box, the implementation includes four further facets. The *occupation* facet is able to select speeches based on the occupation of the speaker. This data field records information about a) government positions (such as chancellor) or b) additional occupations (such as lawyer or farmer) of speakers. The facet for *political party* filters by affiliation with a party (if any). The facet for *role* filters by the role the speaker had at the time of giving the speech. Roles could be a member of parliament, one of the presidents or another role (which aggregated a range of government positions). Finally, the facet for *duration of speech* filters the duration of the speeches in bins of 180 seconds. This includes a bin for zero seconds, which is mainly occupied by a range of very short introductory speeches introductions (such as "The next speaker will be Mr. so-and-so—three minutes time.") which are erroneously recorded with a length of zero seconds in the dataset.

ElasticUI

Speaker Name:

Facets

Occupation

Präsidentin des Nationalrates (17819)
Dritter Präsident des Nationalrates (11060)
Präsident des Nationalrates a.D. (8372)
Zweiter Präsident des Nationalrates (8234)
Vizekanzler und Bundesminister für Finanzen (3062)

Political Party

spö (29700)
övp (27199)
fpö (16857)
grüne (8626)
none (7128)

Role

☐ pres
☐ abg
☐ other
pres (50292)
abg (31382)
other (14044)

Duration

0 (62546)
300 (1022)
180 (1009)
240 (968)
120 (794)
360 (596)
60 (590)
420 (313)
480 (191)
179 (175)

Results Per Page


▼

Home **EUI**

Results

Previous Next


20.10.2008, 19:00 LLP 23 / Session 75 Duration: 0:00 pres



Meine Damen und Herren, die Fernsehdirektübertragung ist somit beendet. Mir liegen aber noch zwei Wortmeldungen vor: Als Erste gelangt Frau Abgeordnete Tamand
(more)

Dr. Michael Spindelegger
None Vizekanzler und Bundesminister für Finanzen
Dec 21, 1959


20.10.2008, 18:22 LLP 23 / Session 75 Duration: 0:00 pres



Der von Frau Abgeordneter Mag. Lunacek eingebrachte Entschließungsantrag ist ausreichend unterstützt und steht mit in Verhandlung.
Der Antrag hat folgenden G
(more)

Dr. Michael Spindelegger
None Vizekanzler und Bundesminister für Finanzen
Dec 21, 1959


20.10.2008, 17:47 LLP 23 / Session 75 Duration: 0:00 pres



Nächster Redner ist Herr Abgeordneter Mag. Rossmann. 6 Minuten maximale Redezeit. – Bitte, Herr Kollege.
(more)

Dr. Michael Spindelegger
None Vizekanzler und Bundesminister für Finanzen
Dec 21, 1959


20.10.2008, 16:55 LLP 23 / Session 75 Duration: 0:00 pres



Die Redezeit der nächsten fünf Rednerinnen und Redner beträgt ebenfalls je 8 Minuten.
Herr Abgeordneter Mag. Kogler gelangt als Nächster zu Wort. – Bitte.
(more)

Mag. Barbara Prammer
SPÖ Präsidentin des Nationalrates
Jan 11, 1954
† Aug 2, 2014

20.10.2008, 18:44 LLP 23 / Session 75 Duration: 0:00 pres



Nächste Rednerin ist Frau Abgeordnete Dr. Lichtenegger. 5 Minuten maximale Redezeit. – Bitte, Frau Abgeordnete.
(more)

Dr. Michael Spindelegger
None Vizekanzler und Bundesminister für Finanzen
Dec 21, 1959

Figure 2.1: The front-end application implemented using the API ElasticUI.js. The left side of the Figure show shows the search box and the facets that can be used to filter the speeches. The right side shows the content area. Screenshot taken by the authors.

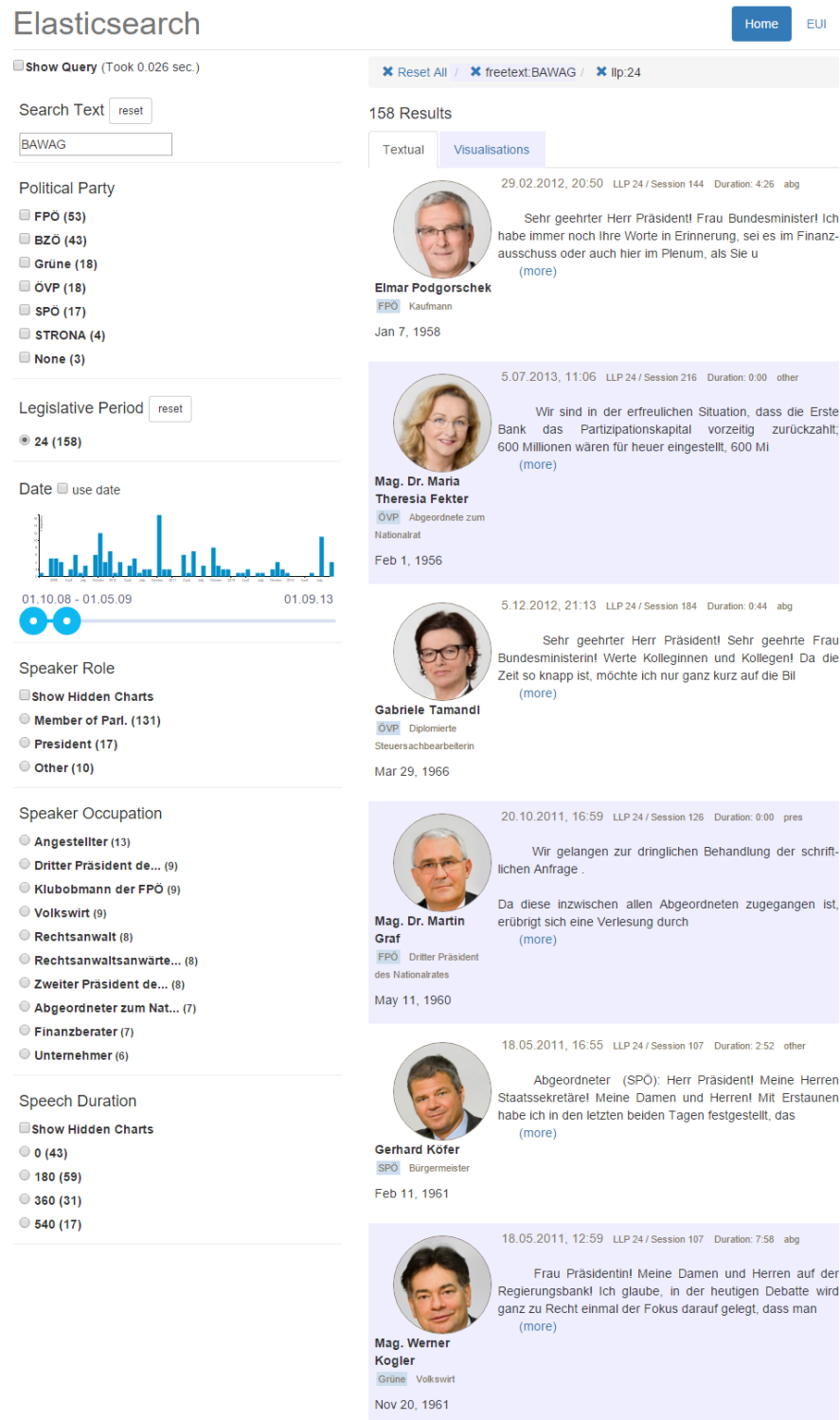


Figure 2.2: The front-end application implemented using the API `elasticsearch.js`. The left side of the Figure show shows the search box and the facets that can be used to filter the speeches. The right side shows the content area. Screenshot taken by the authors.

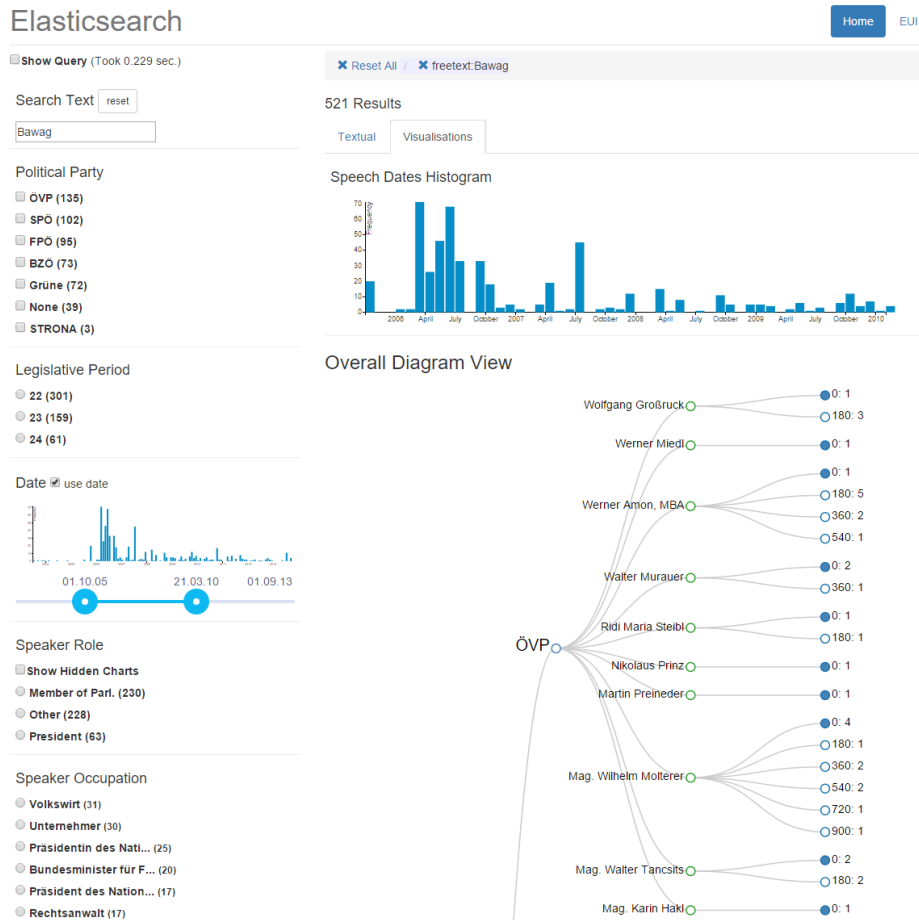


Figure 2.3: The date histogram allows a visual query by speech date and an hierarchical diagram gives a general overview of all the speeches containing the search term *BAWAG*. Screenshot taken by the authors.

2.2.2 Front-End Application with elasticsearch.js

Figure 2.2 shows the application using the API `elasticsearch.js`. For this approach, the free-text search box queries the text of speeches. Furthermore, it supports facets for *occupation*, *political party*, *role* and *duration of speech* (all of which are also supported by the ElasticUI example). In addition to these facets, the application also supports a *date* facet with a slider that enables a detailed selection of a date range. Finally, a facet for the *legislative period* is also included.

In addition to the facets, the `elasticsearch.js` implementation includes several visualizations. Figure 2.3 shows the two visualizations available by clicking the corresponding tab on top of the search results. Firstly, a larger version of the speech date histogram (which is also shown as part of the *date* facet) is available, allowing to associate speeches with certain dates. As an example, Figure 2.3 shows that speeches containing the search term *BAWAG* peaked in 2006, shortly after the affair concerning the bank became public.

The visualization below the histogram gives an overview of the number of speeches by the speakers. For sake of saving space, only the top section is shown in Figure 2.3; showing what members of the Austrian conservative party (ÖVP) gave speeches mentioning the *BAWAG* and the corresponding lengths.

An additional visualization is available by checking a box next to the speaker role facet and shows the top words used by the selected speeches. Figure 2.4 again shows an example of speeches mentioning *BAWAG*, which, unsurprisingly, comes out as the top word.

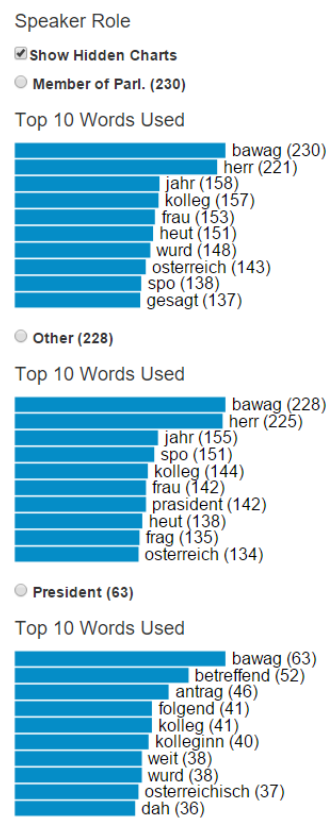


Figure 2.4: Visualization of the most frequently used words in speeches, on the example of all speeches containing the search term *BAWAG*.

Chapter 3

Implementation

This chapter describes the technology used to implement the faceted user interface project. To demonstrate the wide availability and ease of use of faceted user interface technologies, two approaches were implemented. Both implementations used Elasticsearch on the server side but differed in their use of client-side technologies. In addition, a visualization based on D3.js was made.

3.1 Application Architecture

Figure 3.1 shows the architecture of the application. It mainly comprises three components, namely the Elasticsearch server, a static Web server for serving the front-end application and a Web browser for executing the client-side JavaScript. As the JavaScript front-end is purely a client-side application, no server-side processing is necessary and only requests onto the Elasticsearch API are performed to retrieve the query results. Accessing the Elasticsearch API can be done in a RESTful manner—meaning utilizing the HTTP method verbs to express the intent of the request. The query results will be delivered in JSON format by Elasticsearch, which then can be processed on the client-side for visualization purpose.

Since the setup of this project is based on two different servers, all of which utilizing a different port, one may face a problem with the same-origin policy. A web browser only allows request to the same origin (IP address and port number) from which the current Web page has been loaded. In the case of this particular project setup, the front-end application will be loaded from the static Web server, which runs on a specific IP address and port, and then try to send an request to the Elasticsearch API, which has a different origin. Now, due to the same-origin policy this will not be possible and any Web browser will not permit this request. The solution to this problem is CORS, meaning cross origin resource sharing. With CORS, one can add a white list containing the origin of the other server to allow access. Elasticsearch offers such a feature within its configuration and to resolve the discussed issue, one only has to make sure that CORS is enabled and set up correctly.

3.2 Search Server: Elasticsearch

The search server was provided by Elasticsearch [Elastic, 2016]. Elasticsearch is an open-source solution based on Apache Lucene and provides a distributed search engine. Elasticsearch server was chosen because of its unique aggregation capabilities, that provide more powerful grouping capabilities than facets alone.

3.3 Front-End Application

For the front-end application, this project includes two demo implementations. Both are implemented using AngularJS [Google, 2016]. AngularJS is wide-spread and allows to quickly bootstrap development of a JavaScript application. However, any other framework that employs data-binding might have been used just as well.

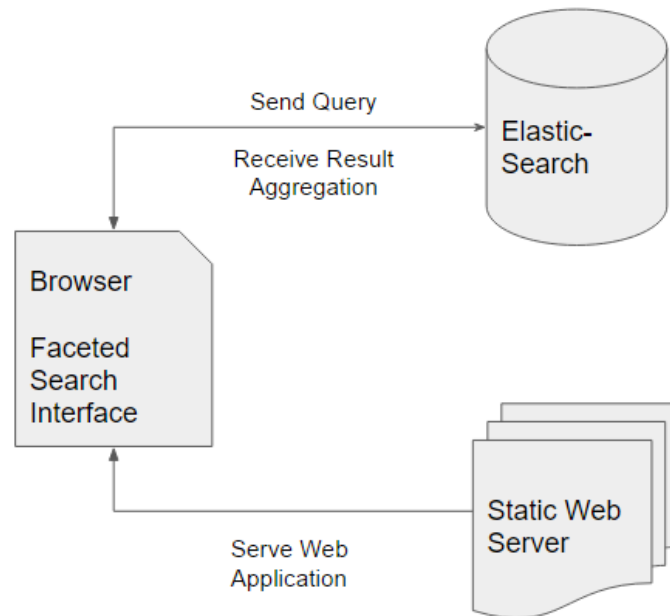


Figure 3.1: The architecture of the application consists of an Elasticsearch server, a static Web server for serving the front-end and a Web browser for executing the client-side JavaScript.

One of the implementations is based on ElasticUI [ElasticUI, 2016], which provides a set of readily available directives to connect to Elasticsearch and allows to easily get started using simple widgets that integrate with Elasticsearch. The other implementation is built using the standard `elasticsearch.js` API which in turn gives more freedom and fine-grained control in development. Finally, the visualizations used in the application were implemented with the D3 framework.

3.3.1 ElasticUI Client

ElasticUI [ElasticUI, 2016] builds on AngularJS and consists of a set of AngularJS directives to connect with Elasticsearch. As such, ElasticUI adds an additional layer to the project and brings with this many shortcuts to simplify development. For example, ElasticUI makes available directives to directly return the values of a facet. Listing 3.1 shows the general architecture of querying Elasticsearch for any general data aggregation and the returning of results to the client interface. The downside of using the framework is the limited degree of control over the interface. For example, the writing of multi queries, where one needs to run multiple queries in parallel, or the act of searching immediately on every key typing, or even the control of a query by using a range visualization, are good examples of current limitations when using the ElasticUI API. This project includes an implementation of a faceted search interface based on ElasticUI that was very quick to set up. However, for a more detailed implementation with custom widgets, the interface based on the `elasticsearch.js` API provided more functionality.

3.3.2 Elasticsearch.js Client API

Developing the client using the `elasticsearch.js` API requires more effort than using ElasticUI. However, it brings with it the advantages of being able to exercise fine-grained control over the entire search interface and the widgets. For this reason, this project includes a faceted search interface built using the `elasticsearch.js` API and featuring a number of widgets that were easily achievable in the ElasticUI interface (e.g., the date interval slider). Listing 3.2 shows a short example of how to utilize the `elasticsearch.js` library. In order to connect to the Elasticsearch server, one has to first create a new client object. During the instantiation, the connection

```

1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>ElasticUI Demo</title>
5     <script src="angular.js"></script>
6     <script src="elasticsearch.angular.js"></script>
7     <script>
8       angular
9         .module('facetedsearch', ['elasticui'])
10        .constant('euiHost', 'http://localhost:9200');
11    </script>
12  </head>
13  <body ng-app="facetedsearch">
14    <div class="sidebar">
15      <h3>Search</h3>
16      <eui-searchbox field="'search'"></eui-searchbox>
17      <h3>Single select facet</h3>
18      <eui-singleselect field="'occupation'" size="5"></eui-singleselect>
19      <h3>Multi select facet</h3>
20      <eui-checklist field="'political-party'" size="10"></eui-checklist>
21    </div>
22    <div class="results">
23      <h1>Results</h1>
24      <ul>
25        <li ng-repeat="doc in indexVM.results.hits.hits">
26          {{doc._source | json | limitTo: 500}}
27        </li>
28      </ul>
29      <eui-simple-paging></eui-simple-paging>
30    </div>
31  </body>
32 </html>

```

Listing 3.1: Example of how the ElasticUI directives are used in template code. The directives that are used here are eui-searchbox, eui-singleselect, eui-checklist and eui-simple-paging.

```

1 // connect to elasticsearch
2 var elasticsearch = require('elasticsearch');
3 var client = new elasticsearch.Client({
4   host: 'localhost:9200',
5   log: 'trace'
6 });
7
8 ...
9
10 client.search(query).then(function (resp) {
11   var results = resp;
12 }, function (err) {
13   console.trace(err.message);
14 });

```

Listing 3.2: This is a basic example of how to use the client object to issue a request to the backend. After the client object was initialised, a search query is issued. The call is asynchronous and will return a promise. Using the promises-API, the result is handled in the `then()` method.

parameter are specified. This can be examined in the lines 3 to 6. After successfully connecting the client object to the Elasticsearch API, one can call various query methods on the client object as seen in line 10. Due to the fact that all API calls take place in an asynchronous way, promises are used to solve dependencies on the query result.

3.3.3 Widget Model

Figure 3.2 shows the widget model of the Web application using the Elasticsearch JavaScript client. The JavaScript/AngularJS part keeps a 'model' for every widget that is rendered. The model data will mostly be a set of values, and their corresponding document counts. Together with result of items, this data is contained in the result received from the backend in the form of 'buckets' or 'bins'. To receive this data from the Elasticsearch backend, it first has to be requested as aggregations. In addition, the contents of the query (the currently selected or otherwise specified values and ranges have to be added to the request to the backend as well - the current query context determines the result set from which the aggregations are derived. Both parts are transmitted to the backend using the *queryDSL* of Elasticsearch. For a full documentation of that DSL, see [Elastic, 2016]. For every user interaction that changes the query context then, the request to the backend is compiled from aggregations and query-clauses. In the project, a helper component to facilitate that task was implemented as an angularjs-service (`QueryBuilderService`, in `query-service.js`). In the main application controller (`FacetedDemoController`, in `app.js`), configuration regarding i) the aggregation and ii) the query clause is passed on to the `QueryBuilderService`. The query-builder component then provides a method to assemble the queryDSL out of the configuration, and the current query context. An example of the queryDSL of such a request is given in Listing 3.3. Parsing the result, the model gets updated, which will in turn (through the framework's object binding) trigger the update of the widgets and the DOM elements involved.

3.3.4 Visualizations

For the visualizations several approaches were tried. Firstly, a general overview diagram was introduced, based on the JavaScript API `d3.js`. Figure 2.3 shows this general overview diagram. This visualization is created by aggregating data based on the current search query (facets widgets and search term). The visualization shows which parties are returned by the current search. also for each party, the name of the speakers is represented as well as the duration of each speech made by each speaker. Next to the speech duration, the number of documents found for each speaker and in each duration bin is added.

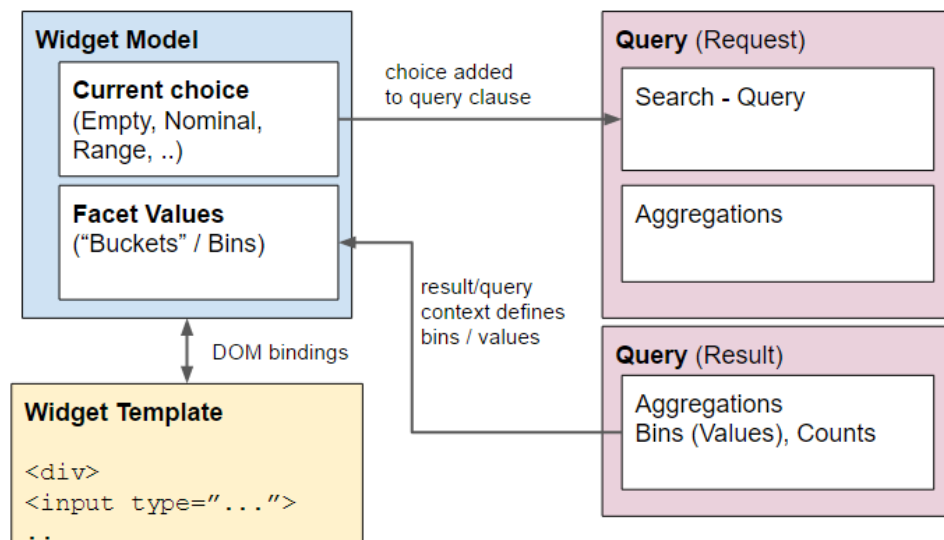


Figure 3.2: The widget model and the lifecycle of the application: the query to the backend contains clauses to define the current query context, and aggregations. The aggregations (facet values that are available under the current query context, and the corresponding document counts) will be parsed from the result and become the underlying data for the widgets' models. The AngularJS framework is responsible for updating the DOM elements that correspond to the widgets.

```

{"body":
  {"aggregations":{
    "party":{"terms":{"field":"current_party"}},
    "llp":{"terms":{"field":"llp"}},
    "duration":{"histogram":{"field":"duration","interval":180}},
    ...
  },
  "query":{
    "filtered":{"filter":{
      "and":[
        {"term":{"llp":"22"}},
        {"range":{"date":{"gte":1064212277060,"lt":1088480517360}}}
      ]
    }}
  }
}}
  
```

Listing 3.3: The contents of a request to the backend in queryDSL. Two main blocks exist: the query-block and the aggregations-block.

Secondly, a date histogram was created. Figure 2.3 shows this date overview chart, on top of the hierarchical diagram. The date chart represents all speeches dates along the period returned by the current query.

Thirdly, a date widget histogram was created. Figure 2.3 shows this date range histogram on the left side, where the user can use the range sliders to easily define a date constraint. This widget allows the user to specify a search date range on top of the current query. The widget serves as an example on how to perform a facet query using a visualization. For this visualization the JavaScript API `dangle.js` together with `jquery.js` (for the sliders) was used, because it was more easily adaptable for this kind of widget visual representation. We also give an example on how to achieve the creation of a similar range widget, by using the JavaScript APIs `d3.js`, `crossfilter.js` and `dc.js`, these libraries are usually used to aggregate data (here this function is easily replaced by the native Elasticsearch aggregation capabilities), making the usage of simpler APIs like `dangle.js` possible.

Finally, a significant terms analysis aggregation example was implemented. It analyzes the speech text field, and returns only the top 10 most significant terms found on the returned speeches (depending on the current query). Figure 2.4 shows the most frequently used words split by speaker role. The significant term analysis, must be configured (as mentioned before) in the mapping file used to load the data into the Elasticsearch server. An additional settings section must be prepared, where the analyzer and the filters to be used must be defined, eg., German stop-words and a German text analyzer. A list of additional terms to be excluded can also be referred during the creation of the json query string. This analysis serves as an example on how to use an analyzer and it would need of course to be improved in a real case scenario. More information about this configuration can be found on the project files: `notes.md` and `quick_elasticsearch_reference.md`.

Chapter 4

User Manual

This chapter describes how to install, set up and run the project. The required software packages are named, after which instructions on how to build frontend application and how to configure the parts of projects are given.

4.1 Requirements

To run the project, at the bare minimum, a version of the Elasticsearch server software and a web server process to serve the HTML, CSS and JavaScript files are required. Probably the best way to build and serve the project for demonstration and development purposes is to use nodejs and it's development tool-chain that also includes a simple webserver. The Elasticsearch server can be obtained from [\[Elastic, 2016\]](#). In the development of this project, version 2.1.0 was used. To build the project, a set of JavaScript/NodeJS-based [\[Joyent, 2016\]](#) build-tools is required. All of these tools can best be installed using the NodeJS package-manager npm [\[npm, 2016\]](#). However, the following software packages are pre-requisites and have to be downloaded manually (or installed using the os' packaging system):

- Elasticsearch server [\[Elastic, 2016\]](#)
- Nodejs [\[Joyent, 2016\]](#)
- NPM [\[npm, 2016\]](#)

4.2 Building and Using the Application

This section contains information on how to run and configure the Elasticsearch backend. It further describes how to install and run (serve) the developed JavaScript application and lastly how to import data into the index.

4.2.1 Running And Configuring Elasticsearch

Elasticsearch can be downloaded as a zipped archive. The contents of the archive should be extracted, after which the suitable executable (depending on the operating-system) can be run from the `bin`-folder. Configuration of the server can be provided in the file `config/elasticsearch.yml` in the Elasticsearch directory, should it be necessary. At least the CORS-settings should be configured, another common option would be to set specific directories for Elasticsearch to keep the index and the log-files. The project contains the file `elasticsearch-example.yml`, in which directives to configure just these two options are included.

```
$ node install
$ bower install
$ gulp serve
```

Listing 4.1: Initialise and run (serve) the frontend project: fetch JavaScript dependencies for development/tools and for frontend third party code; then use the gulp build tool to serve the project.

4.2.2 Installing Frontend Requirements

Given that NodeJS and npm have been installed, npm should be run from the project directory to first fetch requirements for building the project. Among others, these include the development server, gulp [gulp.js, 2016] as the build tool, and BowerJS [Bower, 2016] to manage the dependencies of the actual JavaScript frontend app (the full list of packages that will be install is kept in `package.json`). As the next step, BowerJS is used to install the third party libraries the web application is built upon, such as AngularJS and the Elasticsearch client-library (full list in `bower.json`). Listing 4.1 depicts all the necessary steps to initialize and run (serve) the frontend application.

4.2.3 Building and Running the Frontend

As mentioned before, gulp is used to build the project, and optionally build and start the development server and serve the project. Thus, the most important gulp-tasks are `gulp build` and `gulp serve`. The first task, `gulp build`, will compile `.scss` files to `.css`, minify and concatenate JavaScript source code, and eventually copy the finished project into the `dist`-folder. With the second task, `gulp serve`, after the build, node's development webserver is employed to serve the web-page on `http://localhost:9000`.

4.2.4 Using a Different Webserver

Once the project was built to the `dist`-folder, of course any webserver can be used to serve the contents of the directory.

4.3 Indexing Data

This section describes means of importing data into an Elasticsearch index, along with fine-tuning some of the data's fields using a mapping file.

4.3.1 Importing Data Into an Index

To be able to query and use the Elasticsearch backend, it first has to be given some data to index, that means a collection of documents (in JSON format) has to be added to an Elasticsearch *index*. In this project, the documents were referenced to as 'statements', while the index was named 'opendata'. Documents are added to an Elasticsearch index again using the software's REST/http interface. A convenient way when indexing a large dataset is to use the bulk API. The REST API endpoint is for the bulk API is `/_bulk`. The expected JSON structure is a repeating sequence of first an action- or metadata-statement, then followed by an optional data-row. Listing 4.2 shows an example of a JSON structure prepared to import documents. If such data was stored in a file named 'import.json', and an Elasticsearch server was running on localhost, port 9200, it could then be imported using cURL [cURL, 2016] as an http client with the following command-line statement:

```
curl -XPOST localhost:9200/opendata/statement/_bulk --data-binary @import.json
```

For this project, the data was fetched using the code available from [Informationsfreiheit, 2016], then preprocessed so to have it in the aforementioned format, with each import file containing statements of one legislative period.

```

{"index ":{}}
{"party ":" greens ", " date ":"2015-05-13", ... }
{"index ":{}}
{"party ":" lif ", " date ":"2014-11-07", ... } ,
...

```

Listing 4.2: Example of Elasticsearch bulk API import statementst - a document is preceeded by the relevant action such as adding the document to the index.

```

{
  "statement": {
    "properties": {
      "birthdate": {
        "type": "date",
        "format": "strict_date_optional_time||epoch_millis"
      },
      "deceased": {
        "type": "boolean"
      },
      "llp": {
        "type": "long"
      },
      "occupation": {
        "type": "string",
        "index": "not_analyzed"
      },
      "role": {
        "type": "string"
      },
      ...
    }
  }
}

```

Listing 4.3: Excerpt of the mapping file that was used in the project. For the field occupation, analysing was explicitly turned off, to leave the contents of the field intact and not have it tokenized.

4.3.2 Usage of a Mapping File

With the import action in the previous section, no specific configuration was applied regarding the mapping and analyzing of the data when it is indexed. The means to specify such configuration is by a mapping-file. Somewhat similar to a schema-definition of a relational database, the mapping defines how a document and its fields are stored and indexed. The mapping defines data types (such as numbers, dates, text) and for each field, an analyzer can be specified - however, in some cases an explicit mapping file can be omitted, with the backend's attempts to automatically detect data types and suitable analyzers being reasonably good. A more in-depth explanation of available data types and indexing mechanisms can be found in the official Elasticsearch documentation [Elastic, 2016]. Building on the example of the previous section, the existing mapping (here the default, auto-detected mapping) can be queried with the following request: `GET /opendata/_mapping/statement`

Listing 4.3 depicts an excerpt of the mapping file that was used in the project. In large parts, the default mapping received with the statement above was unchanged. However, for instance for the field containing the occupation of the speakers, the analyzer was explicitly turned off to leave the contents of the field intact and not have it tokenized.

The steps for importing data with an explicit mapping file are shown in listing 4.4. To continue with the

```
curl -XPUT localhost:9200/opendata
curl -XPUT localhost:9200/opendata/_mapping --data-binary @mapping.json
curl -XPOST localhost:9200/opendata/_bulk --data-binary @import.json
```

Listing 4.4: Import of documents into Elasticsearch using the bulk API and cURL as a client. A mapping file is used to explicitly define the document structure. The commands in rows one to three create the index, set a mapping definition and import data, respectively.

example from above, the current index would have to be deleted first - a task that can be achieved with the following request to the Elasticsearch backend : `DELETE localhost:9200/opendata` . The three necessary steps are i) creating the index ii) uploading / defining the mapping of the index, followed by iii) importing the actual data file(s).

Bibliography

- Blum, Mathias et al. [2015]. *Faceted User Interfaces*. Survey handed in as part of 706.041 Information Architecture and Web Usability in the winter term of 2015 at Graz University of Technology. 9th Dec 2015 (cited on page 1).
- Bower [2016]. *Bower*. 2016. <http://bower.io> (cited on page 18).
- cURL [2016]. *cURL and libcurl*. 2016. <https://curl.haxx.se> (cited on page 18).
- Elastic [2016]. *Elastic*. 2016. <http://elastic.co> (cited on pages 11, 14, 17, 19).
- ElasticUI [2016]. *ElasticUI*. 2016. <http://elasticui.com> (cited on pages 5, 12).
- Google [2016]. *AngularJS - Superheroic JavaScript MVW Framework*. 2016. <https://angularjs.org> (cited on pages 1, 11).
- gulp.js [2016]. *gulp.js - the streaming build system*. 2016. <http://gulpjs.com> (cited on page 18).
- Informationsfreiheit, Forum [2016]. *Forum-Informationsfreiheit/OffenesParlament...GitHub*. 2016. <https://github.com/Forum-Informationsfreiheit/OffenesParlament> (cited on pages 3, 18).
- Joyent, Inc. [2016]. *Node.js*. 2016. <https://nodejs.org> (cited on page 17).
- npm, Inc. [2016]. *npm*. 2016. <https://npmjs.com> (cited on page 17).
- Offenes Parlament [2016]. *OffenesParlament.at*. 2016. <http://offenesparlament.at> (cited on page 3).
- Parlamentsdirektion [2016]. *Österreichisches Parlament*. 2016. <http://parlament.gv.at> (cited on page 3).
- Tunkelang, Daniel [2009]. *Faceted Search*. Synthesis Lectures on Information Concepts, Retrieval, and Services. 2009. ISBN 1598299999. doi:10.2200/S00190ED1V01Y200904ICR005 (cited on page 1).