

Masterprojekt
Entwicklung einer webfähigen, indexbasierten
Suche, am Beispiel von Mitfahrgelegenheiten

Christian Weckel

16. März 2016

Inhaltsverzeichnis

1	Einleitung	2
2	Vorstellung des Programms	2
3	Umsetzung	6
3.1	Elasticsearch	6
3.2	Backend	7
3.3	Frontend	8
4	Fazit	10
	Abbildungsverzeichnis	I
	Listings	II

1 Einleitung

Ziel dieses Projekts ist das Erstellen eines Vergleichsportals für Mitfahrgelegenheiten.

Dieses soll als Webapplikation realisiert werden. Dazu müssen verschiedene Mitfahrgelegenheitsportal ausgelesen werden. Deren Ergebnisse werden dann zwischengespeichert, um anschließend über eine eigene Weboberfläche abgerufen werden zu können. Diese Oberfläche soll über einen Browser erreichbar sein und dort sollen sich Start, Ziel, sowie Datum der zu suchenden Mitfahrgelegenheiten angezeigt werden. Bestätigt ein Nutzer seine Eingabe, so werden ihm alle passenden Mitfahrgelegenheiten mit näheren Informationen angezeigt.

2 Vorstellung des Programms

Abbildung 1 zeigt die Weboberfläche des Programms, direkt nach ihrem Aufruf. Somit wurden noch keine Sucheingaben getätigt.

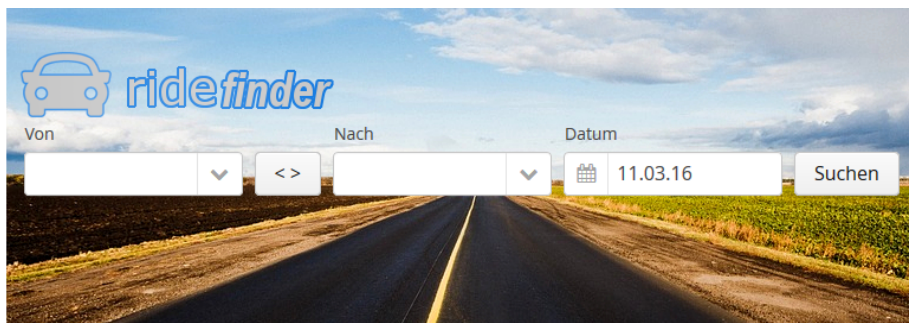


Abbildung 1: Weboberfläche direkt nach dem Laden

Im oberen Bereich der Oberfläche befinden sich die Komponenten, welche der Eingabe von Suchkriterien dienen. Umrahmt werden diese von einem Hintergrundbild und dem Logo der Anwendung.

Erstes Eingabefeld trägt den Bezeichner 'Von'. Es handelt sich dabei um die Eingabe des Startorts. Besonderheit dieses Felds ist die Autovervollständigung. Fängt der Nutzer an einen Ort einzugeben, so werden unter dem Feld Orte vorgeschlagen, welche mit seiner aktuellen Eingabe beginnen. Diese kann er durch einen Klick oder mit Enter auswählen. Diese Funktionalität ist in Abbildung 2 dargestellt.

Ähnlich der Startort-Eingabe, ist das Feld mit dem Bezeichner 'Nach'. Dieses ist für die Eingabe des Endorts zuständig.

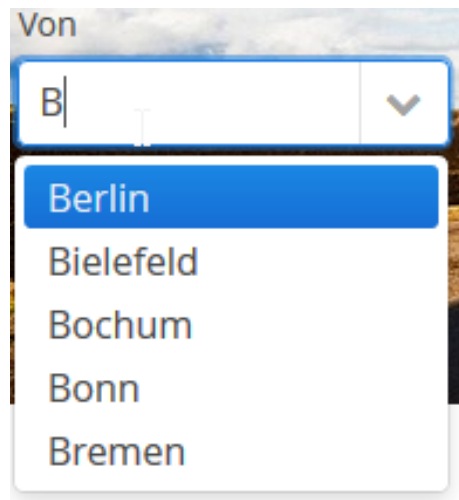


Abbildung 2: Suchvorschläge

Zwischen den beiden Ortseingabe-Feldern befindet sich noch ein Knopf. Dieser ist für den Tausch von Start- und Endposition vorgesehen. Nützlich ist dies, da oft nach der Suche einer Fahrt, auch eine Rückfahrt gesucht wird.

Im letzten Eingabefeld wird das Datum der zu suchenden Mitfahrgelegenheit eingegeben. Dabei kann das Datum entweder direkt eingegeben oder über einen Kalender ausgewählt werden. Bei der direkten Eingabe muss ein gültiges internationales Format eingehalten werden, sonst wird die Eingabe nicht erkannt und das Feld rot hervorgehoben.

Der Kalender lässt sich durch einen Klick auf das zugehörige Symbol öffnen. Dort kann ein einzelner Tag mit einem Doppelklick auf diesen ausgewählt werden. Abbildung 3 zeigt diesen Kalender.

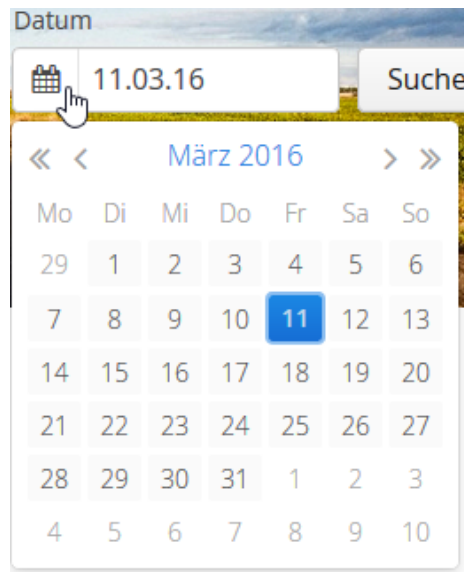
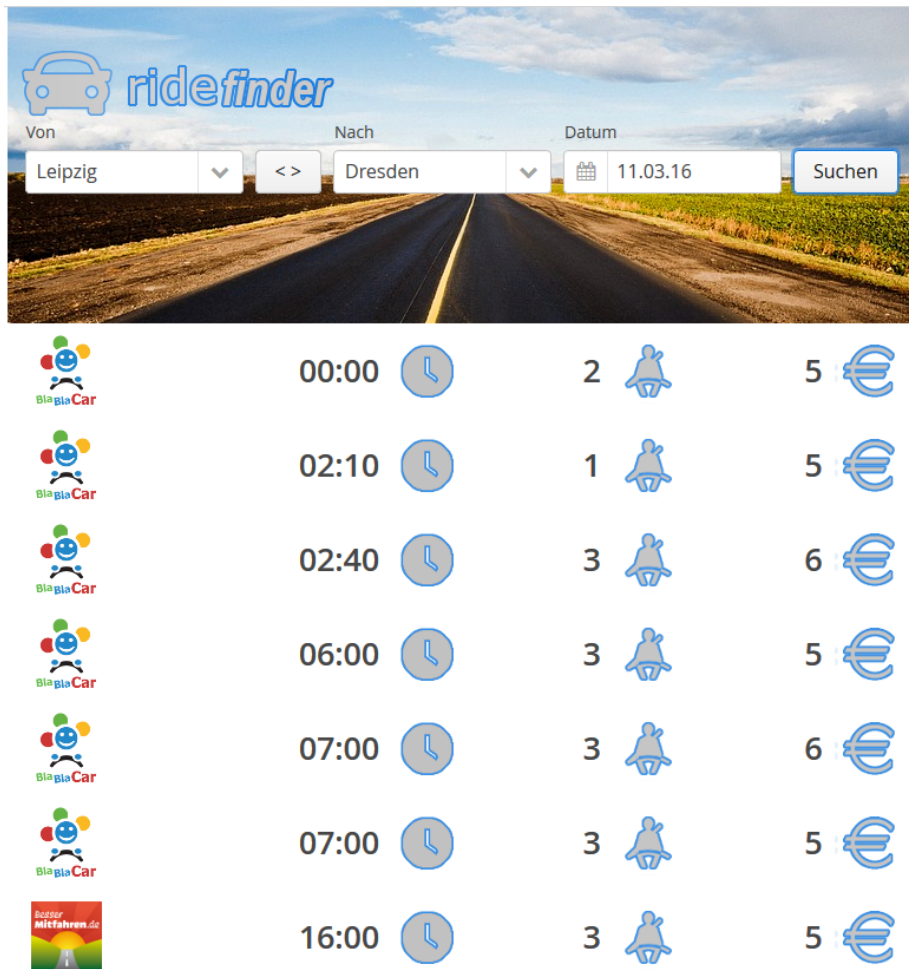


Abbildung 3: Kalender für Datumsauswahl

Der Knopf ganz rechts, wird dafür genutzt, die Eingaben abzuschicken und anschließend die Suchergebnisse anzuzeigen. Diese werden in einer Liste dargestellt, wie sie Abbildung 4 zeigt. Werden keine Ergebnisse gefunden, so zeigt dies eine entsprechende Nachricht an.



Von	Nach	Datum	Suchen
Leipzig	< >	Dresden	11.03.16






















	00:00		2		5 €
	02:10		1		5 €
	02:40		3		6 €
	06:00		3		5 €
	07:00		3		6 €
	07:00		3		5 €
	16:00		3		5 €

Abbildung 4: Suchergebnisse

Ein Suchergebnis besteht immer aus folgenden Komponenten.

Erstere ist das Logo des Mitfahrgelegenheitsportals, von dem die Information stammt.

Zweite ist die Uhrzeit, wann die Mitfahrgelegenheit losfährt.

Die dritte Information ist die Anzahl der freien Plätze. Ist kein Platz verfügbar, so wird die Mitfahrgelegenheit gar nicht erst in den Ergebnissen angezeigt.

Die Information ganz rechts ist der Preis, welchen eine Person für das Mitfahren bezahlen muss.

Zu guter Letzt gibt es auch noch einen Link zu der Originalseite der Mitfahrgelegenheit. Dieser wird durch einen Klick auf den Suchergebniseintrag betätigt und dient dazu, den Fahrer zu kontaktieren.

3 Umsetzung

Das Programm besteht aus drei eigenständigen Komponenten. Diese Komponenten werden in der Abbildung 5 als blaue Objekte dargestellt. Zwei der Komponenten sind ein eigens geschriebenes Backend und ein Frontend. Die dritte Komponente ist die Suchengine Elasticsearch. Der Quelltext der beiden selbstgeschriebenen Komponenten, ist im Github vorhanden. Die Links dafür werden in den jeweiligen Kapiteln genannt. Im Github befindet sich auch eine Anleitung, wie das Programm auf einem Ubuntu-System über das Terminal eingerichtet werden kann.¹

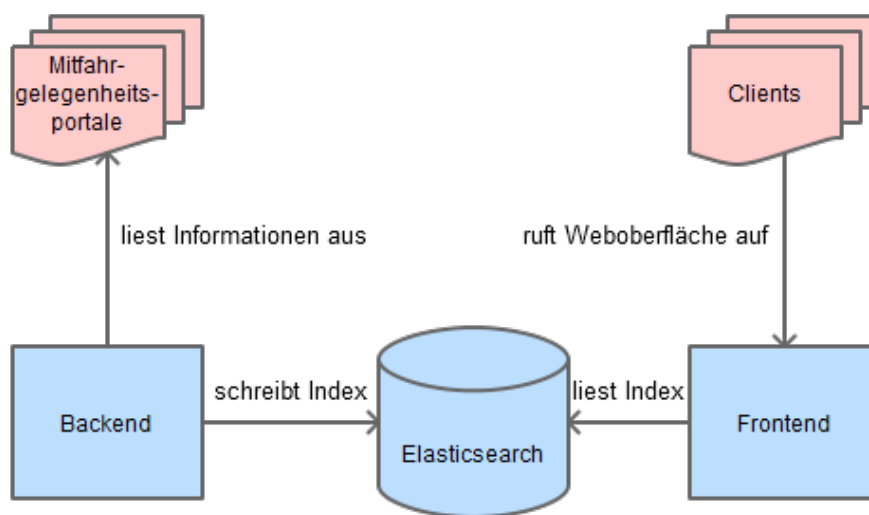


Abbildung 5: Komponenten

3.1 Elasticsearch

Bei Elasticsearch handelt es sich um eine Suchengine. Elasticsearch ist ein open source-Projekt und kann somit frei in anderen Softwarelösungen verwendet werden.

Die Suchengine wird als ein eigener Service auf dem System ausgeführt. Dabei bietet Elasticsearch eine Schnittstelle an, über die andere Services mit ihr kommunizieren können. In der Regel ist dies der Port 9200. Ausnahme dafür ist eine Java-Schnittstelle. Für Java wird eine eigene Java API von Elasticsearch angeboten. Diese ermöglicht es Komponenten von Elasticsearch direkt im Java-Quelltext zu verwenden. Dadurch wird die Nutzung von Elasticsearch mit einer Java-Software stark erleichtert. In Bezug auf Java wird empfohlen den Port 9300 zu verwenden.

Der Vorteil bei der Verwendung von Elasticsearch ist, dass die Daten in einem

¹https://github.com/IVleafclover/RideFinder_Documentation/blob/master/Installationsanleitung%20Ubuntu.txt

Index verwaltet werden. Dieser ist speziell für Suchanfragen ausgelegt und beschleunigt den Suchvorgang um ein Vielfaches, im Vergleich zu herkömmlichen Datenspeichersystemen, wie Datenbanken. Die Komplexität die Daten zu pflegen nimmt dabei jedoch nicht zu.

3.2 Backend

Bei dem Backend handelt es sich um ein in Java geschriebenes Programm, welches keine Oberfläche besitzt. Es wird einmal gestartet und bleibt dann im Hintergrund aktiv. Möglich ist dies, da die einzelnen Vorgänge nach einem Zeitplan ausgeführt werden und das gesamte Programm niemals terminiert. Der Quelltext des Backends liegt vollständig im Github vor.²

Eine Aufgabe des Backends ist es die verschiedenen Mitfahrgelegenheitsportale auszulesen. Da einige der Portale mit Ajax und JavaScript generiert werden, werden auch deren Inhalte nicht sofort beim Aufruf angezeigt. Des Weiteren kann Java alleine auch kein JavaScript ausführen. Somit würde das Auslesen einiger dieser Portale keine Mitfahrgelegenheiten zurückgeben.

Aus diesem Grund verwendet das Backend einen kopflosen Browser. Dabei handelt es sich um einen eigenständigen Browser, welcher keine Oberfläche besitzt. Dieser ermöglicht es JavaScript-Funktionen auszuführen. Ein solcher Browser lässt sich in Java durch das Framework HtmlUnit verwenden. Die Implementierung ist dabei sehr einfach und kann bei Interesse im Quelltext nachgeschaut werden.

Ist durch den Browser ein Mitfahrgelegenheitsportal aufgerufen, so wird dessen Suchformular ausgefüllt und abgeschickt. Als Resultat wird die Ergebnisseite an den Browser übermittelt. Je nachdem welches Portal gerade ausgelesen wird, muss eventuell noch auf die Ausführung von JavaScript-Funktionen gewartet werden.

Anschließend kann die Suchergebnisseite ausgelesen werden. Dies geschieht über die Verwendung von XPath-Befehlen. Solche Befehle erlauben es die Struktur der Webseite auszulesen und Elemente mit gewissen Eigenschaften zu erhalten. So können z.B. alle Links mit einer bestimmten Klasse ausgegeben werden. Das Listing 1 zeigt wie die Attribute von 'BesserMitfahren' ausgelesen werden.

Listing 1: Auslesen der Eigenschaften über XPath

```
1  final List<DomNode> times = (List<DomNode>) resultPage
2      .getByXPath("//a/*[contains(concat(' ', @class, ' '), ' time
   ')]");
3  final List<DomNode> prices = (List<DomNode>) resultPage
4      .getByXPath("//a/*[contains(concat(' ', @class, ' '), ' price
   ')]");
5  final List<DomNode> seats = (List<DomNode>) resultPage
6      .getByXPath("//a/*[contains(concat(' ', @class, ' '), ' people
   ')]");
7  final List<DomNode> dates = (List<DomNode>) resultPage
8      .getByXPath("//a/*[contains(concat(' ', @class, ' '), ' date
   ')]");
9  final List<DomNode> links = (List<DomNode>) resultPage
```

²https://github.com/IVleafclover/RideFinder_Backend


```
10         .getByXPath("//li[contains(concat(' ', @class, ' '), ' clear  
        ')]/a");  
  
12     final List<DomNode> pages = (List<DomNode>)  
        resultPage.getByXPath(".*[@id='pager']/a");
```

Nachdem die Suchergebnisse ausgelesen wurden, werden diese in Java-Klassen zwischengespeichert. Anschließend wird eine Verbindung zu Elasticsearch aufgebaut und die gefundenen Mitfahrgelegenheiten in ein JSON-Format konvertiert. Dadurch können sie direkt in den Elasticsearch-Index geschrieben werden. Dies funktioniert wie in Listing 2 dargestellt wird.

Listing 2: Schreiben einer Mitfahrgelegenheit in den Index

```
1     public static void write(final Ride ride, final Client client) {  
2         final Map<String, Object> json = new HashMap<String, Object>();  
3         json.put("from", ride.getFrom());  
4         json.put("to", ride.getTo());  
5         json.put("date", ride.getDate());  
6         json.put("time", ride.getTime());  
7         if (ride.getPrice() == null) {  
8             json.put("price", "null");  
9         } else {  
10            json.put("price", Float.toString(ride.getPrice()));  
11        }  
12        json.put("seat", Integer.toString(ride.getSeat()));  
13        json.put("provider", ride.getProvider());  
14        json.put("link", ride.getLink());  
  
16        @SuppressWarnings("unused")  
17        final IndexResponse response = client.prepareIndex("rides",  
        "ride", ride.getId()).setSource(json).get();  
18    }
```

Eine weitere Aufgabe ist es alte Einträge aus dem Index zu entfernen. Dazu werden einmal alle 24 Stunden alle Mitfahrgelegenheiten des gestrigen Tages gesucht und anschließend sofort gelöscht. Dies soll dazu beitragen, dass der Index nicht ins Unendliche wächst.

3.3 Frontend

Das Frontend des Programms ist auch in Java geschrieben. Es handelt sich um eine Java-Webapplikation, welche in einem geeigneten Server, wie Tomcat, ausgeführt werden kann. Der Quelltext des Frontends ist ebenfalls im Github vorzufinden.³

Auch wenn das Frontend in Java geschrieben ist, handelt es sich bei der Weboberfläche um eine Webseite, welche durch JavaScript erzeugt wird. Zuständig dafür ist ein Framework namens 'Vaadin'. Dieses ermöglicht in Java einzelne Komponenten zusammenzufügen. Bei Ausführung des Programms generiert

³<https://github.com/IVleafclover/RideFinder>

Vaadin aus den Java-Komponenten JavaScript-Komponenten und damit eine Webseite. Das Aussehen der Webseite kann dabei noch mit eigenen Stylesheets ähnlich CSS angepasst werden. Ein Beispiel, wie Oberflächen mit Vaadin erstellt werden, zeigt Listing 3. Dabei handelt es sich um das Initialisieren des Sucheingabe-Formulars.

Listing 3: Initialisieren des Suchformulars

```
1 private void initSearchInput(final VerticalLayout layoutHeader) {
2     final HorizontalLayout layoutsearchInput = new HorizontalLayout();
3     comboboxFrom = new ComboBox("Von", new
4         ArrayList<String>(Arrays.asList(destinations)));
5     comboboxFrom.addStyleName("searchComponent");
6     layoutsearchInput.addComponent(comboboxFrom);
7
8     final Button buttonChangeFromAndTo = new Button("< >");
9     buttonChangeFromAndTo.addClickListener(new Button.ClickListener() {
10         @Override
11         public void buttonClick(final ClickEvent event) {
12             changeFromAndTo();
13         }
14     });
15     buttonChangeFromAndTo.addStyleName("searchComponent");
16     layoutsearchInput.addComponent(buttonChangeFromAndTo);
17     layoutsearchInput.setComponentAlignment(buttonChangeFromAndTo,
18         Alignment.BOTTOM_LEFT);
19
20     comboboxTo = new ComboBox("Nach", new
21         ArrayList<String>(Arrays.asList(destinations)));
22     comboboxTo.addStyleName("searchComponent");
23     layoutsearchInput.addComponent(comboboxTo);
24
25     datefieldDate = new DateField("Datum", new Date());
26     datefieldDate.addStyleName("searchComponent");
27     layoutsearchInput.addComponent(datefieldDate);
28
29     final Button buttonSearch = new Button("Suchen");
30     buttonSearch.addClickListener(new Button.ClickListener() {
31         @Override
32         public void buttonClick(final ClickEvent event) {
33             if (validateInputs()) {
34                 removeSearchResults();
35                 addSearchResults();
36             }
37         }
38     });
39     buttonSearch.addStyleName("searchComponent");
40     layoutsearchInput.addComponent(buttonSearch);
41     layoutsearchInput.setComponentAlignment(buttonSearch,
42         Alignment.BOTTOM_LEFT);
43     layoutHeader.addComponent(layoutsearchInput);
44     layoutHeader.setComponentAlignment(layoutsearchInput,
45         Alignment.TOP_LEFT);
46 }
```

Auch das Frontend besitzt eine Verbindung zu Elasticsearch. Diese benötigt es, um die Suchergebnisse zu laden. Dazu muss eine Query an die Suchengine gesendet werden. Um mehrere Suchkriterien zu kombinieren, werden BoolQueries benötigt. Dabei lässt sich einstellen, ob ein Kriterium erfüllt werden muss oder soll. Dies entspricht einer Und- bzw. Oder-Verknüpfung. Wurde so eine Query erzeugt, lässt sie sich direkt an Elasticsearch weitersenden. Als Ergebnis wird ein SearchResponse zurückgegeben, der alle Ergebnisse enthält. Listing 4 zeigt wie so eine Query erstellt und ausgeführt wird.

Listing 4: Ausführen einer Suche

```
1  final BoolQueryBuilder searchQuery = QueryBuilders.boolQuery()
2      .must(QueryBuilders.termQuery("from", from.toLowerCase()))
3      .must(QueryBuilders.termQuery("to",
4          to.toLowerCase()).must(QueryBuilders.termQuery("date",
5              date)));
6
7  SearchResponse searchResponse =
8      client.prepareSearch("rides").setScroll(new TimeValue(60000))
9      .setQuery(searchQuery).execute().actionGet();
```

4 Fazit

Die Erstellung eines Vergleichsportals im Rahmen dieses Projekts, stieß auf ein paar wenige Schwierigkeiten. Eine davon war das Auslesen der Mitfahrgelegenheitsportale, welche ihre Daten mit JavaScript nachladen. Dieses Problem lässt sich aber sehr gut umgehen, wenn ein kopfloser Browser verwendet wird.

Die Umsetzung des Frontends mit Hilfe von Vaadin funktionierte ohne weitere Probleme. Das Framework Vaadin ist schnell zu erlernen und zu benutzen. Es ist für jegliche Webanwendungen mit Java geeignet.

Am Anfang des Projekts bestand die Idee die Mitfahrgelegenheiten immer live von den Portalen zu laden. Die Wahl Elasticsearch als Zwischenspeicher für die Daten zu verwenden und die Mitfahrgelegenheiten nicht live zu laden, schien erst falsch zu sein.

Es hat den Nachteil, dass die Daten nicht immer aktuell sind. Des Weiteren entsteht auch ein großer Datentransfer bezüglich der Mitfahrgelegenheitsportale. Aus diesem Grund wurde auch ein Timer eingebaut, welcher dafür sorgt, dass die Portale nur jede Minute ausgelesen werden und das Programm nicht auf einer Blacklist landet.

Der Vorteil der Verwendung von Elasticsearch ist der, dass die Nutzer extrem schnell die Suchergebnisse angezeigt bekommen. Dies ist sehr wichtig, da der kopflose Browser einiges an Zeit brauchen kann, bis alle JavaScript-Funktionen und somit die Suchergebnisse geladen werden. Vor allem bei Portalen, wo die Ergebnisse auf mehreren Seiten vorliegen, würde diese Wartezeit für viele Nutzer zu groß sein.

Deshalb ist die Verwendung von Elasticsearch doch die bessere Lösung.

Abbildungsverzeichnis

1	Weboberfläche direkt nach dem Laden	2
2	Suchvorschläge	3
3	Kalender für Datumsauswahl	4
4	Suchergebnisse	5
5	Komponenten	6

Listings

1	Auslesen der Eigenschaften über XPath	7
2	Schreiben einer Mitfahrgelegenheit in den Index	8
3	Initialisieren des Suchformulars	9
4	Ausführen einer Suche	10