Institut für Rechnergestützte Ingenieursysteme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3741

Extraktion, Analyse und grafische Anzeige von Jobdaten

Kai Hatje

Studiengang:	Informatik
Prüfer:	Prof. Dr. D. Roller
Betreuer:	Dipl.-Inf. F. Baumann
Beginn am:	22. Februar 2016
Beendet am:	23. August 2016
CR-Nummer:	H.3.3

Kurzfassung

Arbeitsstellen werden im Internet meist über Jobbörsen gesucht. Diese bieten in der Regel allerdings nur sehr grundlegende und begrenzte Such- und Anzeigemöglichkeiten. Insbesondere die Möglichkeit, sich weitere Informationen anzeigen zu lassen, ist meistens nicht gegeben. Außerdem ist die Art, wie Suchergebnisse angezeigt werden, nicht besonders übersichtlich.

In dieser Arbeit wurde ein System implementiert, welches aufbauend auf gesammelten Jobbörsendaten Arbeitsstellen übersichtlich anzeigen und weiterführende Informationen dazu einblenden kann. Dazu wird eine Karte verwendet, auf der die Arbeitsstellen als Punkte geographisch angezeigt werden. Zudem werden weitere Informationen eingeblendet, direkt als Werte und als Barcharts.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1. Einleitung

Bisher gibt es noch wenige Tools im Bereich der Arbeitsmarktanalyse. Dies gilt auch für den Bereich Business Process Management (BPM). Dabei ergeben sich viele Vorteile aus einem solchen Tool, sowohl für Firmen als auch für den Privatanwender.

Zum Beispiel könnten Firmen sehen, welche Arbeitsstellen in welchem geographischen Bereich eine hohe Chance haben, besetzt zu werden. Dies würde für Firmen die Entscheidung erleichtern, wie und wo eine Arbeitsstelle ausgeschrieben werden soll. Die Ausschreibungskriterien könnten entsprechend angepasst werden und die Wahrscheinlichkeit, die Arbeitsstelle zu besetzen, würde steigen. Weitere Entscheidungshilfen wären verschiedene Statistiken, die zu den Arbeitsstellen zugänglich gemacht werden könnten.

Privatanwendern könnte dies unter anderem helfen, eine Arbeitsstelle in einem bestimmten Bereich zu finden. So könnten sich diese auch an der geographischen Verteilung von Arbeitsstellen orientieren, was übersichtlicher wäre als dies über mehrere Seiten angezeigt zu bekommen. Zudem wäre ein interessantes Kriterium, wie lange eine Arbeitsstelle besetzt war, also wie lange Arbeitnehmer bei der entsprechenden Firma geblieben sind. Dies kann Aufschluß über die Qualität der Arbeitsstelle geben.

Zuletzt könnten die Jobbörsendaten, die oft ein großes Volumen haben, auf übersichtliche Weise dargestellt werden. Damit würde der Zugang für den Anwender erleichtert, was die Daten zugänglicher machen würde und somit aufwerten würde.

1.1. Problemstellung

Die Suche von Arbeitsstellen und die Darstellung der Ergebnisse bei Jobbörsen unterliegen gewissen Einschränkungen. So kann meist nur über entsprechende Eingabeformulare gesucht werden (siehe Abbildung 1.1). Das Ergebnis der Suche wird meist als Liste der gefundenen Arbeitsstellen zurückgegeben. Dabei ist die Anzahl der Suchergebnisse oft eingeschränkt, zum Beispiel auf 200 Einträge (siehe [Bun16b]).

Laut Sebrechts et al. [SCL+99] wird das Scrollen und Scannen von Text ab einer bestimmten Anzahl von Dokumenten unpraktisch. Zu nennen sind hier unter anderem die Zeit, die hierfür benötigt wird, als auch die mangelnde Übersichtlichkeit.

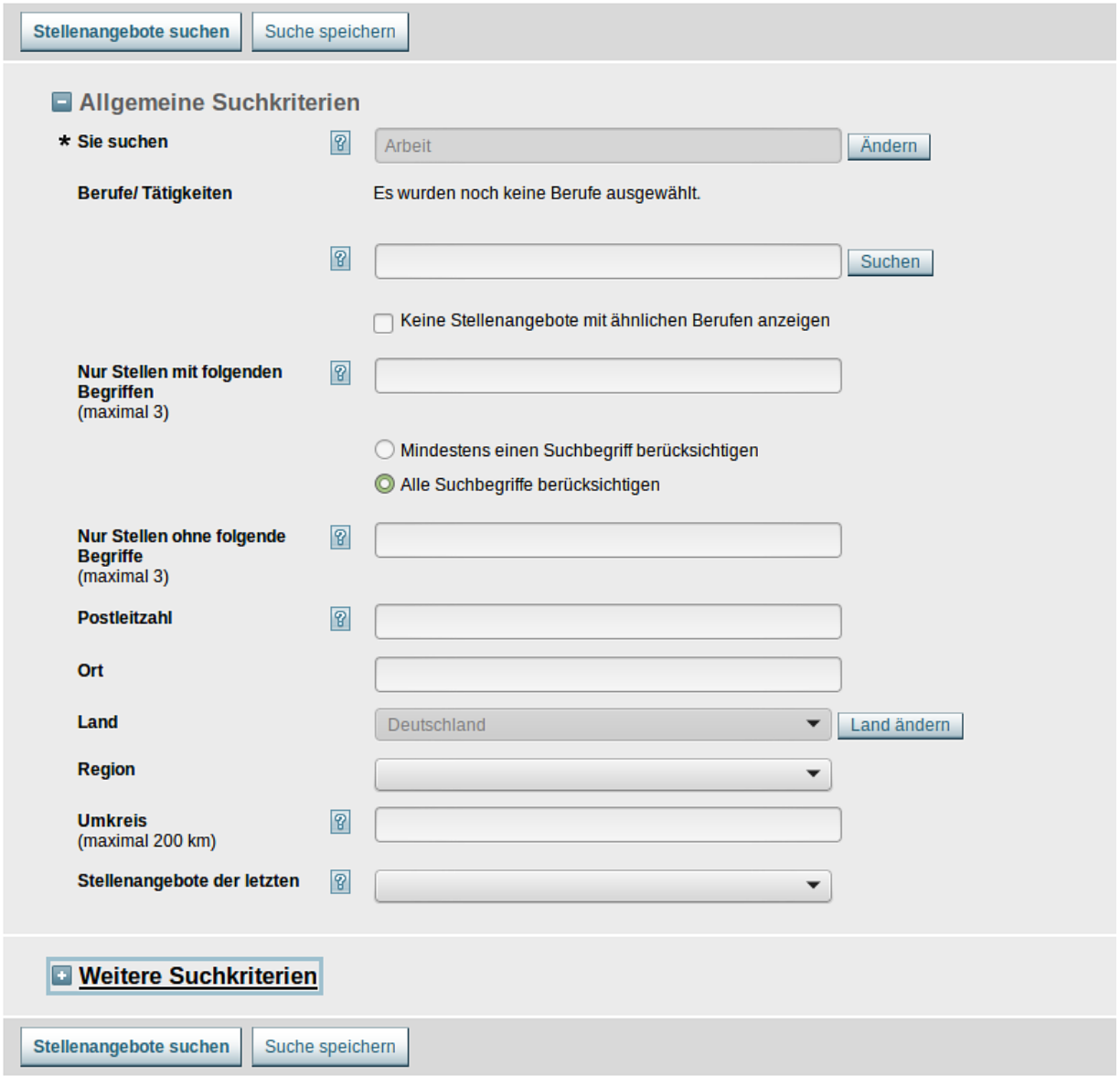

Abbildung 1.1.: Suchmaske der Jobbörse [Bun16b]

Außerdem ist es einfacher für den Benutzer, wenn alle benötigten Informationen mit einem Blick überschaut werden können, wie in Wästlund, Norlander und Archer [WNA08] beschrieben. Dies senkt die mentale Anstrengung, was das Finden der gesuchten Informationen erleichtert.

So wäre es hilfreich, Arbeitsstellen übersichtlich suchen zu können bzw. einfach herausfinden zu können, wo solche verfügbar sind und welchen Kriterien sie unterliegen. Hierfür wäre es gut, auf begleitende Statistiken und Fakten zugreifen zu können, die die Suche bzw. Erstellung von Arbeitsstellenangeboten erleichtern.

1.2. Lösungsansatz

In dieser Arbeit wird ein ein System vorgeschlagen, das dem Benutzer Informationen in einem vorbereiteten Zusammenhäng präsentiert. Da Arbeitsstellen immer geographisch verknüpft sind, bietet sich als Rahmen eine Landkarte an. Auf dieser können dann alle Arbeitsstellen in einem bestimmten Kontext visualisiert werden. Der Benutzer kann den Kontext entsprechend verändern und so zu seinem gewünschten Ergebnis gelangen.

Begleitend werden Statistiken und Fakten zu den gewählten Punkten ausgegeben, um die Entscheidungen des Benutzers zu vereinfachen und diesem mehr Kontext zu bieten.

1.3. Verwandte Arbeiten

Eine der Grundlagen für die Arbeit ist in Baumann und Roller [BR15] beschrieben. Hier wurden über einen Zeitraum von zwei Jahren Stellenausschreibungen zu einer Auswahl von Suchwörtern in Abhängigkeit zu BPM gesammelt. Diese Daten sind bereits strukturiert und müssen nur noch extrahiert werden, um benutzbar zu sein. In der Arbeit wurde auch die Frage nach weiteren Zusammenhängen gestellt, welche hier teilweise beantwortet wird.

FlipDog war ein spezialisiertes Suchsystem für Jobangebote (siehe Cohen, McCallum und Quass [CMQ00]). Hierbei wurden diese automatisch von Firmenwebseiten extrahiert und mit Methoden des Natural Language Processing (NLP) aufbereitet. Zudem wurde eine Taxonomie erstellt, um die Einträge leichter zugänglich zu machen. Dieser Ansatz ist dem obigen von Baumann und Roller sehr ähnlich. Allerdings wurde auch hier noch kein Schwerpunkt auf die Anzeige der Ergebnisse gelegt.

Es gibt bereits Datenbanken, die für die Speicherung von geographischen Punkten optimiert wurden, wie zum Beispiel nach Zongyao und Fuling [ZF02]. Hier wurde eine relationale Datebank erweitert, um einfachen und schnellen Zugang zu temporalen und spatialen, also geographich räumlichen Informationen zu ermöglichen. Aufbauend auf dieser Forschung gibt es für herkömmliche relationale Datenbanken Plugins, die diese um die Möglichkeit erweitern, spatiale Informationen performant und leicht abrufbar zu speichern. Ein Beispiel hierfür wäre Furieri [Fur16]. Natürlich gibt es inzwischen auch, aufbauend auf dem NoSQL-Paradigma, entsprechende Datenbanken bzw. Plugins. Eine Übersicht über zwei solche Datenbanksysteme ist in Baas [Baa12] zu finden.

Für die Visualisierung wurde unter anderem Keim et al. [KPSN04] herangezogen. Diese Arbeit geht auf die verschiedenen Möglichkeiten zur Darstellung von großen geographischen Datensätzen ein. Eine weitere Quelle ist Andrienko und Andrienko [AA99]. Hier wurden verschiedene Ansätze zur Abfrage und anschließenden Darstellung von Datensätzen erschlossen. Außerdem wurden verschiedene Darstellungsformate verglichen und auf Tauglichkeit überprüft. Schließlich verfügt der Arbeitsmarktmonitor der Agentur für Arbeit über eine Visualisierung für verschiedene Zusammenhänge [Bun16a]. Hierbei werden Statistiken zu

den verschiedenen Ländern eingeblendet, wie zum Beispiel die Arbeitslosenquote über einen bestimmten Zeitraum.

Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 1 – Einleitung Gibt einen Überblick und eine kurze Einführung in die Arbeit.

Kapitel 2 – Verwendete Daten und Technologien Beschreibt die Daten, die verwendet werden und schneidet die eingesetzten Technologien und Methoden kurz an.

Kapitel 3 – Anforderungen Erklärt, wie ein System aussehen soll, das die Problemstellung dieser Arbeit löst.

Kapitel 4 – Entwurf Geht auf die verschiedenen Möglichkeiten, das System zu strukturieren und die Entwicklungsmethode ein.

Kapitel 5 – Implementierung Führt durch die wichtigsten Details der Implementierung sowie gefundene Probleme und deren Lösung.

Kapitel 6 – Ergebnis Zeigt verschiedene Charakteristika des Systems sowie das Erreichte auf.

Kapitel 7 – Zusammenfassung und Ausblick Gibt eine kurze Zusammenfassung der Arbeit und nennt mögliche Entwicklungsbereiche, aufbauend auf den erreichten Ergebnissen.

2. Verwendete Daten und Technologien

Hier sollen die Grundlagen für das Verständnis der folgenden Kapitel gelegt werden. Es wird zuerst auf die Art, die Herkunft und den Aufbau der verwendeten Daten eingegangen. Anschließend werden kurz die verwendeten Technologien vorgestellt, die nicht als selbstverständlich vorausgesetzt werden können.

2.1. Jobbörsendaten

Diese Arbeit baut unter anderem auf Daten von verschiedenen Jobbörsen auf, die über einen Zeitraum von einem Jahr gesammelt wurden (siehe auch Baumann und Roller [BR15]). Hierbei wurde eine Methode verwendet, die über einen längeren Zeitraum hinweg Abfragen an verschiedene Jobbörsenseiten stellt und die Ergebnisse speichert. Anschließend wurde noch eine Extraktion aller möglichen Informationen aus den extrahierten Daten vorgenommen, wobei die extrahierten Informationen strukturiert gespeichert wurden. Diese liegen nun als große XML-Datei vor, aus denen Arbeitsstellen ausgelesen werden können.

2.1.1. Quelle der Daten

Die Daten stammen aus fünf verschiedenen Jobbörsen, nämlich jobboerse.de [1] (die Seite der Arbeitsargentur), stepstone.de [2], monster.de [3], monster.at [4] (die Österreichische Version von monster.de) und karriere.at [5]. Dabei wurde das frei zugängliche Interface der Webseiten verwendet, das auch von einem Privatbenutzer verwendet werden kann.

[1] http://jobboerse.arbeitsagentur.de
[2] http://www.stepstone.de
[3] http://www.monster.de
[4] http://www.monster.at
[5] http://www.karriere.at

2.1.2. Daten Sammelmethode

Die Daten wurden mit einem so genannten Crawler gesammelt. Ein Crawler ist ein Programm, das Webseiten automatisch aufrufen kann. In diesem Fall hat es mit bestimmten Keywords auf den oben genannten Seiten gesucht. Die Ergebnisse, in der Regel HTML-Seiten, wurden zwischengespeichert und gefiltert. Eventuell wurde noch weiteren Links in den Ergebnissen gefolgt, zum Beispiel um eine Jobbörsen-externe Ausschreibung zu bekommen oder weitere Informationen über die Firma, die die Ausschreibung gemacht hat. Anschließend wurde das Ergebnis der Suche, also die zuerst erhaltene Seite und die zusätzlichen Seiten, gespeichert. Diese wurden anschließend verarbeitet.

2.1.3. Verarbeitung der gesammelten Daten

Ein HTML-Dokument, das von einer der Jobbörsenseiten heruntergeladen wurde, folgt meist einer bestimmten Struktur. So zeichnen bestimmte HTML-Tags oder eine bestimmte Abfolge an HTML-Tags Daten in der Webseite aus. Ein Beispiel hierfür ist im Folgenden zu sehen:

```
<span class="labelText ohneFeldhilfe">Arbeitsorte</span>
<p>70173 Stuttgart, Baden-Württemberg, Deutschland</p>
```

Mit diesem Ansatz kann erkannt werden, welche Information an welcher Stelle in einem Dokument sind. Das erlaubt, diese zu extrahieren. Da nur von einer begrenzten, bekannten Anzahl an Seiten extrahiert wird, konnte der Crawler auf diese Webseiten angepasst werden.

Auf diese Art und Weise wurden also verschiedene Informationen aus den erhaltenen Dokumenten extrahiert. Anhand der erhaltenen Informationen wurden durch andere Programme weitere ermittelt und den bereits vorhandenen hinzugefügt. So können zum Beispiel anhand einer Adresse, also Straße, Ort und Postleitzahl, die Koordinaten dieser Adresse gefunden werden. Diesen Vorgang nennt man Ortsbestimmung oder Positionsbestimmung. Diese können dann später verwendet werden, um die Informationen auf einer Karte anzuzeigen, wie Google Maps oder Ähnlichen. Außerdem werden Abfragen über bestimmte geographische Bereiche vereinfacht, zum Beispiel welche Arbeitsstellen in einem bestimmten Bereich liegen.

Die extrahierten Daten wurden anschließend im XML-Format gespeichert, pro Suchlauf eine. Alle Dateien wurden zu einer großen XML-Datei zusammengefügt, für die einfachere Verarbeitung. Diese enthält also viele Suchläufe, die ihrerseits unterschiedlich viele Einträge enthalten.

2.1.4. Aufbau der XML-Datei

Die XML-Datei enhält alle Suchläufe bis zu einem bestimmten Datum. Pro Suchlauf sind alle gefundenen Einträge angegeben, mit allen Daten die extrahiert werden konnten. Ein Beispiel für einen Eintrag ist in Listing 2.1 zu sehen. Zu sehen ist unter anderem, dass ein Eintrag eine Nummer hat, die mit dem `entry`-Tag gekennzeichnet ist. Außerdem sind weitere Tags zu erkennen, die unter anderem `titel`, der Name der Stellenausschreibung, und `lat` sowie `lng` unter `ort`, welches ein Ergebnis der Ortsbestimmung ist und die geographische Lage anzeigt.

Listing 2.1 Auszug eines Eintrages in der XML-Datei

```
<entry no="36">
   <url><![CDATA[http://jobboerse.arbeitsagentur.de/ ... ]]></url>
   <glob_no>36</glob_no>
   <uuid>0282fc89-6c72-4d66-af12-61063ad73397</uuid>
   <source>jobboerse.de</source>
   <term><![CDATA[geschaeftsprozess-management]]></term>
   <titel>
      <orig><![CDATA[BI Analyst (m/w) / Kennziffer: IuK-209-15-WW
         (Informatiker/in (Hochschule))]]></orig>
   </titel>
   <datum>
      <orig>08.12.2015</orig>
      <formatted>2015-12-08</formatted>
   </datum>
   <firma><![CDATA[Alpha-Engineering GmbH & Co. KG]]></firma>
   <ort>
      <orig><![CDATA[Montabaur]]></orig>
      <lat>52.42452</lat>
      <lng>10.7815</lng>
   </ort>
   <beginn>sofort</beginn>
   <status>old</status>
   ...
</entry>
```

Am Ende eines Suchlaufes sind außerdem noch verschiedene statistische Daten zu finden, zum Beispiel die Länge des Suchlaufes und die Anzahl der gefundenen Einträge.

2.2. Verwendete Technologien

Die in dieser Arbeit verwendeten Technologien sind alle Open Source. Daher sind diese ohne Kosten frei verfügbar und könnten in der Regel direkt heruntergeladen werden. Außerdem können sie angepasst werden und bei Fragen oder Programmfehlern kann in den Quelltext geschaut werden bzw. die Fehler direkt behoben werden. Letztendlich sind die Technologien weit verbreitet, was das Finden von Lösungen zu verschiedenen Problemen wahrscheinlicher macht. Dies erleichtert die Benutzung.

2.2.1. Webtechnologien

Durch den Geschwindigkeitszuwachs der letzten Jahre und die stärkere Fokussierung auf Webbrowsertechnologie ist der Browser inzwischen schnell genug für eine Vielzahl an Aufgaben, die zuvor dedizierter Software vorenthalten war (siehe Nielson, Williamson und Arlitt [NWA08]). Dies ist vor allem der Programmierung von immer schnelleren Javascript-Engines zu verdanken (siehe Smedberg [Sme10]). So finden sich inzwischen viele Programme im Browser wieder, die zuvor nur auf dem Desktop direkt verfügbar waren. Beispiele hierfür sind unter anderem Officepakete, die Tabellenkalkulationen, Textverarbeitung und Weitere.

Dies hat unter anderem den Vorteil der Betriebssystemunabhängigkeit. Außerdem muss nichts auf dem System installiert werden, auf dem die Software verwendet werden möchte. Die entsprechende Webseite kann zudem von überall zugänglich gemacht werden.

Auch Geoinformationssysteme (GIS) können inzwischen performant im Browser verwendet werden (siehe Taraldsvik [Tar11]). Das in dieser Arbeit implementierte System zählt als solches.

2.2.2. lxml

lxml ist eine Bibliothek in Python für das Einlesen, die Manipulation und das Schreiben von XML [Beh16]. Sie verwendet die in C geschriebene Bibliothek libxml2 [VT16]. Damit kombiniert sie die Geschwindigkeit von lxml2 mit einem einfach zu verwenden Interface, das den Grundlagen der Einfacheit in Python folgt.

2.2.3. Leaflet

Leaflet ist eine leichtgewichtige Javascript-Bibliothek zur einfachen Verwendung von Maps auf Webseiten. Sie lädt die Landschaftskarten als Kacheln von verschiedenen Anbietern. Hiermit kann das Aussehen angepasst werden. Es werden verschiedene Grundfunktionalitäten geboten, wie das Markieren bzw. Setzen von Punkten auf der Karte sowie die Anzeige von verschiedenen

Kontrollelementen. Durch Plugins kann die Funktionalität von Leaflet erweitert werden. Ein Beispiel für ein für diese Arbeit verwendetes Plugin ist Markercluster, das mehrere Punkte zu einem zusammenfasst [Lea16].

2.2.4. Weitere Technologien

Es wurden noch weitere Programmiersprachen und Technologien verwendet, die als bekannt vorausgesetzt werden können. Trotzdem sollen diese hier kurz genannt und die Motivation für deren Verwendung beschrieben werden.

Python

Python ist eine interpretierte Sprache. Sie findet in sehr vielen Bereichen Anwendung, auch und besonders im wissenschaftlichen Bereich (siehe Pérez, Granger und Hunter [PGH11]. Die Verwendung der Sprache ist seit ihrer Entstehung im Jahr 2001 sehr gewachsen. Inzwischen findet sich die Sprache in den Top 5 der verwendeten Sprachen wieder (siehe TIOBE software BV [TIO16], Carbonnelle [Car16]).

SQLite

SQLite ist ein leichtes und schnelles Datenbanksystem mit dem Motto “Small. Fast. Reliable. Choose any three“ [The16b]. Es ist server- und konfigurationslos und zeichnet sich daher durch sehr einfache Anwendung aus, was es besonders für Prototyping geeignet macht. Durch den Einsatz auf Androidgeräten, iPhones, Computern mit Windows 10, Firefox und vielen mehr gilt es als das am meisten verwendete Datenbanksystem der Welt und ist in den Top 5 der am meisten verwendeten Software der Welt.

PHP

PHP ist die am häufigsten verwendete Programmiersprache zum Erstellen von Websites [The16a]. Sie wird auf über 80 % der Webseiten weltweit verwendet (siehe Q-Success [QSu16]). Die Geschwindigkeit, mit der in PHP entwickelt werden kann als auch die große Verbreitung machen PHP zu einer guten Wahl für prototypische Projekte.

D3

D3 ist eine Javascript-Bibliothek, welche die Anzeige von graphischen Zusammenhängen erleichtert [Bos16]. Sie bietet die Grundlagen für verschiedene Visualisierungen, unter anderem Graphen und Charts.

3. Anforderungen

In der Softwareentwicklung wird zwischen verschiedenen Arten von Anforderungen unterschieden (siehe Ludewig und Lichter [LL13]). Eine Unterscheidung ist hierbei die zwischen funktionalen und nichtfunktionalen Anforderungen.

Funktionale Anforderungen beschreiben, was ein Softwareprodukt konkret leisten soll. Ein Beispiel hierfür wäre, dass das Produkt die durchschnittliche Mitarbeiterzahl aller angezeigten Firmen ausgibt.

Nichtfunktionale Anforderungen gehen über die funktionalen hinaus und werden oft als Qualitätseigenschaften verstanden. Hierunter fallen Merkmale wie die Latenz oder die Handhabung. Diese Merkmale ergeben sich oft aus anderen Eigenschaften des Systems.

3.1. Funktionale Anforderungen aus der Problemstellung

In Kapitel 1 wurden mehrere Punkte genannt, die implementiert werden sollen. Diese sind direkt über funktionale Anforderungen abbildbar. Sie werden hier der Übersichtlichkeit halber noch einmal aufgeführt.

3.1.1. Übersicht über die Informationen

Eine der funktionalen Anforderung ist, dem Benutzer auf einer Bildschirmseite einen Überblick über alle Stellenangebote zu geben. Hierfür wurde die geographische Lage der Arbeitsstellen gewählt. Somit muss der Benutzer nicht durch viele Seiten von Ergebnissen scrollen.

3.1.2. Anzeige von begleitenden Informationen

Eine weitere Anforderung ist die Anzeige von Informationen, die der Übersicht nicht ohne weiteres zu entnehmen sind. Ein Beispiel hierfür ist die durchschnittliche Größe der Unternehmen der gewählten Stellenangebote. Ein anderes Beispiel ist die Verteilung der Stellenangebote über einen gewählten Zeitraum.

3.2. Nichtfunktionale Anforderungen

Als nichtfunktionale Anforderung wurde die Latenz gewählt. Diese wurden gewählt, weil sie das System im Hinblick auf andere Systeme besonders auszeichnet.

3.2.1. Latenz

Unter der Latenz wird die Antwortzeit eines Systems verstanden, also wie lange es braucht, bis auf eine Eingabe des Benutzers ein Ergebnis gezeigt wird. Dies ist in diesem Fall besonders interessant, weil sehr große Datenmengen angezeigt werden sollen. Da die Antwortzeit oft über die Benutzbarkeit eines Systems entscheidet, wurde diese daher als eine der nichtfunktionalen Anforderungen gewählt. Hierbei werden die folgenden Zahlen angegeben, die für ein Softwaresystem gelten sollen (siehe Nielsen [Nie93]). Unter 0,1 Sekunden Reaktionszeit ist der Zeitraum, in dem der Benutzer das Gefühl hat, das System würde sofort reagieren. Unter einer Sekunde wird der Benutzer nicht in seinem Gedankengang gestört. 10 Sekunden gilt als obere Grenze, in der die Aufmerksamkeit des Benutzers gehalten werden kann. Ab dieser Grenze sollte dem Benutzer eine Rückmeldung gegeben werden, zum Beispiel über die Einblendung eines Dialoges, der ihn über den Fortschritt der im Hintergrund stattfindenden Operationen informiert.

4. Entwurf

Der Entwurf des Systems in diesem Kapitel folgt aus den Anforderungen im vorigen Kapitel.

Es werden sowohl die gewählten Entwurfsentscheidungen als auch Alternativen vorgestellt. Außerdem wird die Wahl der Entwurfsentscheidungen begründet.

Zusätzlich wird kurz auf die Entwurfs- und Entwicklungsmethode eingegangen, die verwendet wurde. Auch hier werden Alternativen vorgestellt und die Wahl entsprechend begründet.

4.1. Entwurfs- und Entwicklungsmethode

Als Entwurfs- und auch Entwicklungsmethode wurde Prototyping verwendet (siehe Bischofberger und Pomberger [BP12], Ludewig und Lichter [LL13]).

4.1.1. Prototyping

Prototyping bedeutet, die gewünschen Funktionalitäten nach Abhängigkeit voneinander oder aufsteigender Wichtigkeit zu implementieren. Hierbei können die ersten Versionen durchaus dazu dienen, sich über die Entwurfsidee an das Machbare heranzutasten.

In der hier verwendeten Variante ist dies ein explorativer Ansatz, der sich besonders anbietet, wenn fundamentale Randbedingungen noch nicht feststehen. Es wird mit den Grundfunktionalitäten begonnen. Diese werden auf Machbarkeit getestet.

Anschließend werden immer weitere Funktionalitäten hinzugefügt, bis man optimalerweise beim Endprodukt angelangt ist. Dabei gibt es mehrere Iterationen und man versucht, immer ein möglichst funktionales Ergebnis zu erhalten, bevor dieses weiter ausgebaut wird. Außerdem wird versucht, nach jeder Phase ein funktionierendes Produkt zu haben.

Dadurch, dass forschend entwickelt wird, sind die verschiedenen Phasen eines Projektes nicht so klar voneinander getrennt, wie das bei anderen Entwicklungsmodellen oft der Fall ist. Durch die explorative Natur dieses Entwicklungsmodells ist die Entwurfsphase sehr kurz ausgefallen und war direkt Teil der Implementierungsphase.

4.2. Herangehensweise Entwurf

Um das Ziel einer Oberfläche zu erreichen, die den obigen Anforderungen genügt, müssen auf jeden Fall Informationen zu Arbeitsstellen mit den entsprechenden geographischen Daten vorliegen. Daher ist die grundlegenste Funktionalität die Extraktion und wurde als erste zu implementierende Funktionalität gewählt. Die Visualisierung wurde als zweites Ziel gewählt, da sie einen Kernpunkt der Arbeit darstellt. So kann Erfahrung im Umgang mit den Daten und der Anzeige gewonnen werden.

Anschließend ergibt sich aus Anzahl und Art der Daten die Art der Speicherung. Aus den Anforderungen der Visualisierung und den extrahierten Daten wurde dann die Analyse entworfen.

4.3. Systemaufbau

Das System wurde folgendermaßen entworfen: Daten werden aus der Datenbasis extrahiert (Extraktion), in die Datenbank eingefügt (Speicherung), es werden verschiedene Zusammenhänge gewonnen (Analyse) und schließlich alle Einträge und die verknüpften Zusammenhänge angezeigt (Visualisierung).

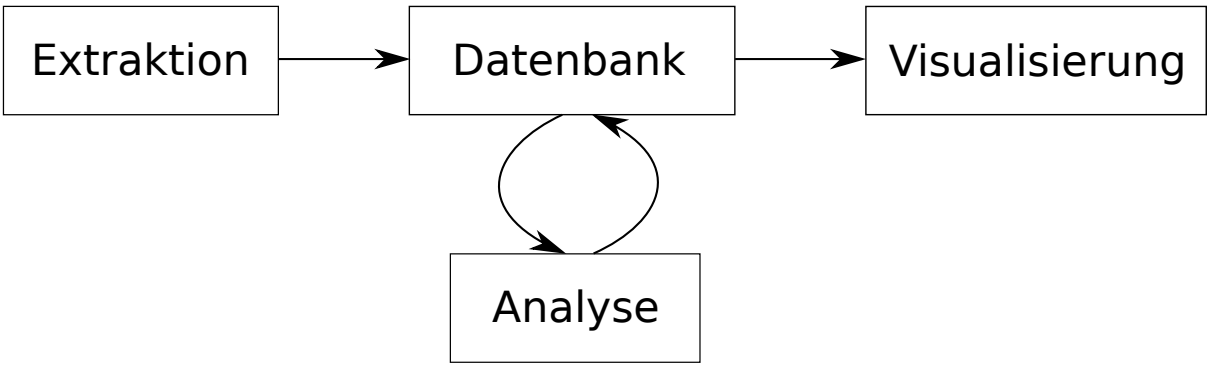

Abbildung 4.1.: Prinzipieller Aufbau des Systems

Extraktion und Speicherung hängen hierbei eng zusammen, da sich das Einlesen der Daten auf das Design der Datenbank auswirkt.

4.3.1. Extraktion

Für die Extraktion wurde ein automatischer Ansatz gewählt. Hierbei soll das Datenbankschema automatisch erstellt werden, während die Daten eingelesen werden. Dies spart den Aufwand für das manuelle Bauen des Schemas. Außerdem werden so alle Daten verwertet und es kann der Unterschiedlichkeit der Daten Rechnung getragen werden.

Trotz der obigen Vorteile ist zu bedenken, dass dieser Ansatz schwer umzusetzen sein könnte. Er könnte daran scheitern, dass die Datenstruktur, in der die Daten vorliegen, uneinheitlich ist. Dies liegt daran, dass der Crawler weiter entwickelt wurde, während des Zeitraums in dem Daten gesammelt wurden. Daher müssen eventuell viele Randbedingungen abgefangen werden. Zudem wird das Auslesen der Daten später erschwert. Aus diesen Gründen könnte statt des automatischen ein manuell erstelltes Datenbankschema notwendig werden.

4.3.2. Speicherung

Wie bereits angedeutet, soll das Schema automatisch angelegt werden. Somit folgt die Speicherung in Aufbau und Entwurf der Extraktion.

4.3.3. Analyse

Die Analyse folgt aus den Daten, die erhoben wurden sowie aus der Visualisierung. Die Daten bestimmen, was möglich ist und die Visualisierung, was davon am meisten Sinn macht bzw. am ehesten benötigt wird. Daher war geplant, die Analyse erst dann weiter voranzutreiben, wenn Extraktion und Speicherung bzw. Visualisierung in den Grundzügen abgeschlossen sind.

4.3.4. Visualisierung

Die extrahierten Punkte sowie die Zusammenhänge sollen in der Visualisierung angezeigt werden. Dies wurde als zweiter Entwicklungsschritt gewählt. Damit konnte schon im Vorfeld Erfahrung für die ausgelesenen Daten und eventuelle Probleme gewonnen werden. Außerdem konnte so die Oberfläche früh entworfen werden.

4.4. Alternative Entwurfs- und Entwicklungsmethoden

Es gab Alternativen, sowohl in der Entwicklungsmethode als auch bei den technischen Entscheidungen.

4.4.1. Alternative Entwurfsmethoden

Eine gängige Alternative zur Prototyping ist das so genannte Wasserfallmodell.

Wasserfallmodell

Das Wasserfallmodell ist ein Entwicklungsmodell, das in der Softwareentwicklung schon lange eingesetzt wird (siehe Ludewig und Lichter [LL13]). Hierbei wird der Projektprozess in mehrere Phasen unterteilt, wobei versucht wird, jeweils eine Phase abzuschließen, bevor in die Nächste übergegangen wird. Im Gegensatz zu Prototyping handelt es sich bei den Phasen nicht um Iterationen, sondern um in sich geschlossene Vorgänge. So würde man zuerst den Entwurf abschließen, bevor die Entwicklung angegangen wird. Es kann zur Not zwischen direkt aufeinander folgenden Phasen gewechselt werden, falls sich herausstellt, dass eine Phase nicht korrekt beendet wurde. Zum Beispiel kann nochmals in die Entwurfsphase zurückgegangen werden, falls in der Entwicklungsphase größere Mängel auffallen.

Dieses Modell eignet sich besonders dort, wo recht präzise die Anforderungen und die Vorgaben für das Projekt beschrieben werden können. Dies war in dieser Arbeit nicht der Fall. Daher wurde gegen das Wasserfallmodell und für das prototypische Modell entschieden.

4.4.2. Alternative Entwurfsentscheidungen

Es gab verschiedene alternative Technologien, die für die Implementierung des Systems in Frage kamen.

Normalisierte Datenbank

In der Datenbankentwicklung gibt es das Konzept der “Normalisierung“. Dies schreibt bei zusammenhängenden Daten einen bestimmten Aufbau der entsprechenden Tabellen vor. Dies hat einige Vorteile in der späteren Benutzung. Unter anderem können Informationen leichter ausgelesen werden, insbesondere was komplexere Abfragen angeht, die das Verbinden von mehreren Tabellen benötigen.

Da die Struktur zu Anfang noch nicht fest stand, wurde dieses Konzept nicht angewendet. Durch die Entscheidung, die Datenbank nicht zu normalisieren, könnten Daten leichter in die Datenbank eingelesen werden.

Big Data

Eine fundamentale Entwurfsentscheidung war die Frage, ob Konzepte aus dem Bereich des “Big Data“ verwendet werden sollen (siehe Manyika et al. [MCB+11]). Allerdings ist die Datenmenge nicht so groß, dass an das Limit von herkömmlicher Technologie gestoßen werden sollte. Daher wurden dieses Konzept nicht verwendet.

NoSQL

Ein Datenspeicherungskonzept, das in letzter Zeit immer mehr an Popularität gewonnen hat, ist das so genannte “NoSQL“, auch key-value-store genannt. Dies ist ein Alternativentwurf zu relationalen Datenbanken und hat besonders bei großen, unstrukturierten bzw. semi-strukturierten Datenbeständen Vorteile. Da im vorliegenden Fall die Daten jedoch strukturiert sind und auch die Datenmenge noch verhältnismäßig klein ist, wurde gegen diesen Ansatz entschieden.

5. Implementierung

Im Verlauf der Implementierung haben sich mehrere Probleme gezeigt. Diese sowie die Lösungen sollen hier vorgestellt werden. Außerdem konnten einige Entwurfsentscheidungen nicht umgesetzt werden. Die Alternativen werden ebenfalls hier vorgestellt.

5.1. Extraktion von Daten

Die XML-Datei mit allen Daten wurde in kleinere Dateien aufgesplittet. Diese werden anschließend der Reihenfolge nach eingelesen. Hierfür wird wie gesagt lxml verwendet, wie in Kapitel 2 angedeutet.

Das Extraktionsprogramm baut pro eingelesener Datei eine Datenstruktur auf, in die die Werte über die XML-Tags eingelesen werden. Nachdem die Datei fertig eingelesen wurde, werden die eingelesenen Dateien in der Datenbank gespeichert.

In Listing 5.1 ist ein Ausschnitt des Programms zu sehen, das für das Parsen der Informationen aus den XML-Dateien zuständig ist. Gezeigt wird ein Ausschnitt des Teiles, der die Datenstruktur aufbaut sowie des Teiles, der das Speichern in der Datenbank übernimmt.

Beim Einlesen gab es mehrere Probleme, die anschließend besprochen werden sollen.

5.1.1. Zuordnung der Felder

Teilweise waren Tags für Informationen in der XML-Datei nicht eindeutig bzw. mehrfach verwendet. So kommt es zum Beispiel vor, dass `orig` sowohl als Unterteil von `file` als auch von `titel` ist. Dies wurde gelöst, indem jeweils das vorangegangene Tag abgefragt wurde. So kann man feststellen, was der markierte Text für eine Information beinhaltet. Dies ist in Zeile 9 bis 13 von Listing 5.1 zu sehen.

Listing 5.1 Auszug aus dem Programmcode für das Einlesen von XML in die Datenbank

```
# Building the dict to insert later.
for event, el in context:
   tag = el.tag
   text = el.text.strip() if el.text else None
   attrib = el.attrib # Is a dict, needs to be processed further.
   parent = el.getparent()
   elif event == "end":
      # Here we have a lot of tags that appear in several contexts.
      if tag == "orig":
         if parent.tag == "file":
            files[e_no][f_no]["orig"] = text
         elif parent.tag == "titel":
            titel[e_no]["orig"] = text
   ...

# Now save everything :)
   # Enter titel belonging to this entry.
   commit(titel[no], "titel", "entry_id")
   # Enter company info belonging to this entry.
   comp_id = commit(comps[no], "comp", "entry_id")
   ...
```

5.1.2. Fehlende Datenfelder

Wie bereits erwähnt, gibt es verschiedene Felder, die nicht in allen Einträgen in der XML-Datei enthalten sind. Ein Beispiel hierfür ist eine `ID`, die einer Ausschreibung eine eindeutige Nummer zuweist. Mit Hilfe dieses Feldes kann eine Ausschreibung eindeutig zugeordnet werden. Dies ist ohne `ID` schwer, da nicht genau erkannt werden kann, ob es sich bei einer ähnlichen Ausschreibung zu einem anderen Zeitpunkt um dieselbe handelt.

So haben zum Beispiel die Ausschreibungen von jobboerse.de kein solches Feld. Daher kann dies bei Ausschreibungen aus dieser Quelle nicht verwendet werden.

Weiter wurde festgestellt, dass nicht alle Einträge in der Datenbank eine Quellenangabe haben, d.h. von welcher Webseite die Stellenanzeige stammt. Dieser Eintrag konnte bei Nachforschung in der XML-Datei nicht gefunden werden. Allerdings wurde sie an anderer Stelle gefunden. Dies legt nahe, dass Teile der großen XML-Datei nicht kohärent sind.

5.1.3. Einlesefehler

Ein weiteres Problem ist, dass manche Felder in der Datbank leer sind. Wie im vorigen Abschnitt beschrieben, kann dies darauf zurückgeführt werden, dass die Information in der Grunddatei nicht enthalten war. Allerdings gibt es auch leere Felder, deren Information vorhanden ist, aber offensichtlich nicht eingelesen wurde. Dieser Fehler konnte bis zum Abschluß der Arbeit nicht behoben werden.

5.1.4. Fehlerhafte XML-Struktur

An manchen Stellen ist die XML-Datei nicht wohlgeformt. Dies bedeutet, dass manche Tags nicht geschlossen wurden und daher die entsprechenden Informationen nicht ohne weiteres maschinell eingelesen werden können.

Um dies zu umgehen, musste der `recover`-Modus von lxml verwendet werden. Hierbei wird die eingelesene XML-Datei nicht auf Wohlgeformtheit geprüft. Die entsprechenden Daten können trotzdem eingelesen werden. Allerdings müssen manche Einträge deswegen verworfen werden.

Eine Alternative, die bisher nicht getestet wurde, wäre, einen ganzen Eintrag einzulesen und mit lxml zu parsen, ohne aktivierten `recover`-Modus. Ist dieser nicht wohlgeformt, wird vom Programm eine Ausnahme ausgelöst. Diese kann abgefangen werden und der entsprechende Eintrag, in dem der Fehler vorgekommen ist, verworfen werden.

5.2. Speicherung der Daten

Es haben sich mehrere Probleme gezeigt, was das automatische Erstellen der Datenbank angeht. So hat sich das Schreiben eines entsprechenden Programmes als sehr schwierig herausgestellt. Außerdem kam es durch die Inhomogenität der Daten zu einer recht inkonsistenten Struktur der Datenbank.

Deswegen wurde auf ein manuell erstelltes Schema ausgewichen, in das nur bestimmte Daten eingelesen wurden. Dieses ist in Abbildung 5.1 zu sehen.

Hierbei wurde das Datenbankschema in einem Programm namens MySQL-Workbench erstellt und mit einem Plugin nach SQLite exportiert (Oracle Corporation [Ora16], Demachi [Dem16]).

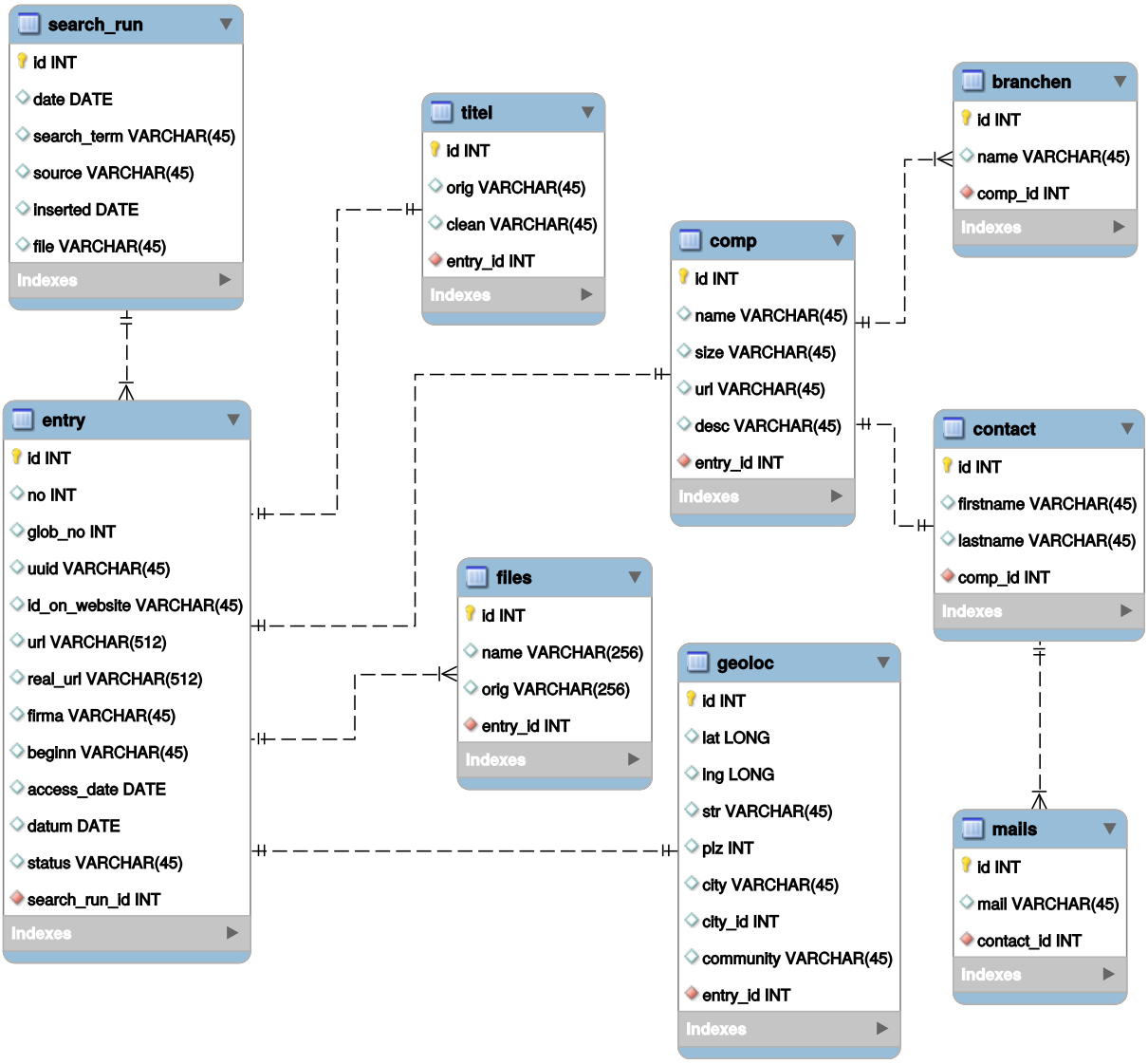

Abbildung 5.1.: Manuelles Schema der Datenbank

5.3. Erstellen von analytischen Zusammenhängen

Für die Analyse wurde sowohl ein Programm in Python geschrieben als auch eigene Tabellen in der Datenbank angelegt. In Kapitel 6 sind Werte zu sehen, die von dem Analyseprogramm errechnet wurden. Die eigenständigen Tabellen wurden für die Anzeige angelegt, da das Erzeugen der Zusammenhänge mit den schon vorhandenen Tabellen teilweise sehr lange dauern kann und diese in der Anzeige schnell verfügbar sein sollen (siehe Abschnitt 5.4.1).

Es wurden zwei verschiedene Zusammenhänge erzeugt: Erstens die Anzahl an Stellenangeboten pro Tag über einen gewählten Zeitraum, zweitens die durchschnittliche Größe der Firmen

für die ausgewählten Punkte. Die Anzahl der Stellenangebote wurde als Chart angezeigt, die durchschnittliche Größe der Firmen als Wert.

In Abbildung 5.2 ist das Schema zu sehen, das genutzt wird, um die Verteilung von Arbeitsstellen über einen bestimmten Zeitraum anzuzeigen.

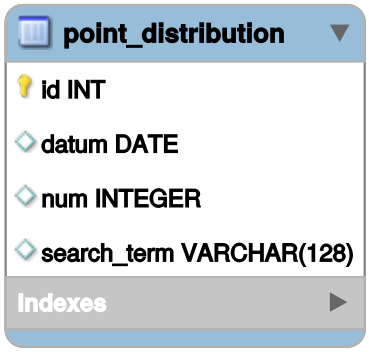

Abbildung 5.2.: Schema für die Verteilung von Arbeitsstellen über einen Zeitraum

Der entsprechende SQL-Code für die Übertragung der Daten aus den verschiedenen Tabellen in die mit obigem Schema ist in Anhang B zu finden.

In Listing B.3 ist das Schema für die Tabelle für die Berechung der durchschnittlichen Größe der gewählten Firmen zu sehen, d.h. die Anzahl der Mitarbeiter dieser Firma. Aus dieser Tabelle kann verhältnismäßig schnell ausgelesen werden, wie groß Firmen in einem gewählten Zeitraum für bestimmte Suchbegriffe sind. Daraus kann dann die durchschnittliche Größe berechnet werden.

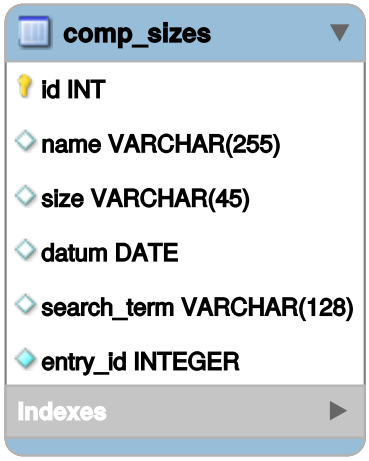

Abbildung 5.3.: Schema der Tabelle für die Berechnung der durchschnittlichen Größe von Firmen

Auch hier ist der entsprechende SQL-Code für die Übertragung der Daten aus den verschiedenen Tabellen in Anhang B zu finden.

5.4. Visualisierung durch Webseite

Auch bei der Weboberfläche gab es einige Herausforderungen, die sich im Verlauf der Programmierung gezeigt haben.

5.4.1. Lange Laufzeit von Datenabfragen

Beim Erstellen der Weboberfläche hat sich gezeigt, dass manche Datenbankabfragen übermäßig lange dauern.

Der erste Ansatz war, einen so genannten “Covering Index“ über die entsprechenden Tabellen zu erstellen. Ein “Covering Index“ ist ein Index, der alle Felder einer Tabelle überspannt. Die Erstellung dieser Indizes ist in Anhang B zu sehen.

Allerdings hat dies nicht den gewünschten Geschwindgkeitszuwachs gebracht.

Wie im vorigen Abschnitt beschrieben, wurden daher weitere Tabellen eingeführt, welche nur die Daten enthalten, die von der Weboberfläche später benötigt werden. Dieser Schritt hat die entsprechenden Abfragen stark beschleunigt.

Ein weiterer Bereich in dem eine eigene Tabelle einen Geschwindigkeitszuwachs gebracht hat, war neben der Analyse das Auslesen von vielen Arbeitsstellen aus der Datebank. Diese mussten, durch die Verteilung der Informationen auf verschiedene Tabellen, durch Queries abgefragt werden, deren Laufzeit sehr hoch war. Durch die Einführung einer eigenen Tabelle, zu sehen in Abbildung 5.4, wurde auch diese Abfrage erheblich erleichtert.

Abbildung 5.4.: Schema für schnellere Rückgabe von Arbeitsstellen

Der SQL-Code für die Transferierung der Daten von den verteilten Tabellen in die aggregierte ist wieder in Anhang B zu sehen. Dort findet sich auch ein Vergleich des SQL-Codes für die Abfrage der Daten vor und nach Erstellung der eigenen Tabelle.

5.4.2. Anzeige vieler Arbeitsstellen

Wurde durch die Zeit- oder Suchbegriffauswahl eine große Anzahl anzuzeigender Arbeitsstellen gefunden, trat das Problem auf, dass die Anzeige sehr unübersichtlich wurde. Dies wurde durch Einsatz des Markerclusterplugins behoben (Leaver [Lea16]). Dabei werden Punkte in einem geographischen Bereich gesammelt auf der Webseite angezeigt. Dies ist ein so genannter Cluster. Je mehr Marker vorhanden sind, desto dunkler ist der Cluster, von hellem Grün bis dunklem Rot. Außerdem ist die Anzahl der zusammengefassten Marker als Zahl in der Mitte des Clusters zu sehen. Bei Anklicken eines Clusters zoomt die Webseite in den Bereich hinein, in dem die zusammengefassten Punkte sich befinden und es werden entweder die Punkte oder weitere Cluster angezeigt, je nachdem wie viele vorhanden sind.

6. Ergebnis

Das System, auf dessen technische Seite und Entwicklung in Kapitel 5 eingegangen wurde, wird hier kurz vorgestellt. Außerdem werden verschiedene Eigenschaften dazu angegeben, wie zum Beispiel die benötigte Zeit für die Extraktion und die Anzahl der extrahierten Einträge. Zuletzt wird das Erreichte mit den Anforderungen verglichen und evaluiert, ob das gewünschte Ergebnis erreicht wurde.

6.1. Webseite

Nachdem die technische Seite des Systems bereits in Kapitel 4 beschrieben wurde, wird hier auf das Frontend, also die Webseite, eingegangen. Diese ist für den Benutzer hauptsächlich relevant. Die Daten müssen nur einmal eingelesen werden, bevor die Webseite verwendet werden kann. Daher wird hier größeres Augenmerk auf die Webseite gelegt.

6.1.1. Standardansicht des Systems

Die Standardansicht der Webseite ist eine Deutschlandkarte, wie in Abbildung 6.1 zu sehen ist. Links sind verschiedene Kontrollknöpfe zu sehen, zum Zoomen, für das Rezentrieren nach verschiedenen Zoombefehlen und für den Aufruf der Hilfe. Außerdem ist darunter ein Bereich für weitere Informationen über eine einzelne Stellenanzeige vorbereitet, als auch ein Bereich für statistische Auswertungen, weiter unten. Ganz unten findet sich eine Kontrolle für die Auswahl eines Zeitbereiches mit Hilfe eines Schiebereglers. Auf der rechten Seite können Begriffe ausgewählt werden, die in den Ausschreibungen vorkommen sollen. Rechts unten ist ein Bereich für Charts. Wenn man über diesen mit der Maus geht, wird die Chartanzeige geöffnet.

Bei Auswahl eines Zeitraums über den Schieberegler werden die dazu gefundenen Punkte geladen und angezeigt. Außerdem werden die Zusammenhänge neu berechnet und die Chartanzeige aktualisiert. Das gleiche geschieht bei Auswahl eines oder mehrerer Suchbegriffe. Zeitraum und Suchbegriffe können auch kombiniert werden.

Größere Mengen an gefundenen Punkten in einem Bereich werden zu einem so genannten Cluster zusammen gefasst. Dabei wird angezeigt, wie viele Punkte gefunden wurden. Durch Clicken auf einen Cluster wird in den Bereich der zusammengefassten Punkte hineingezoomt.

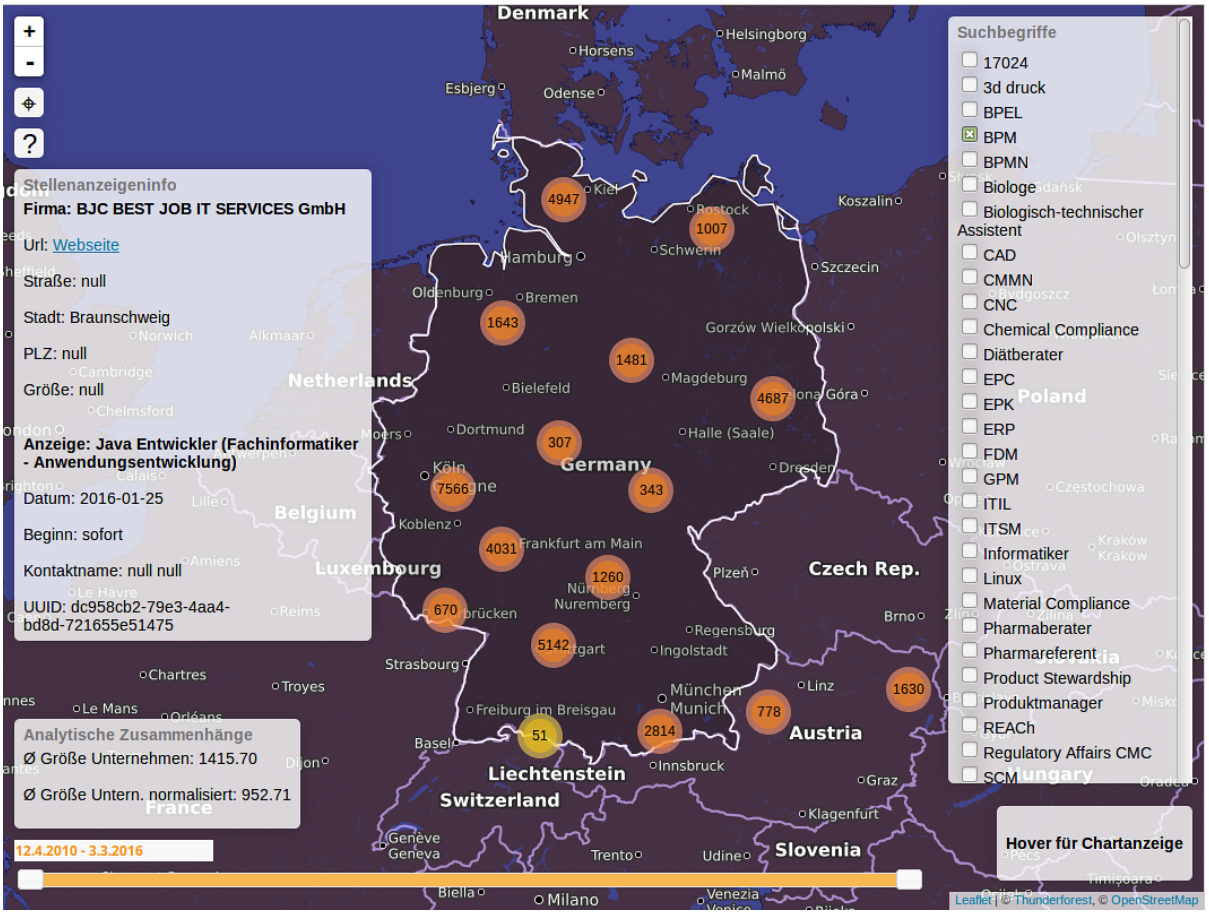

Abbildung 6.1.: Aussehen der Oberfläche der Webseite

6.1.2. Mögliche Interaktionen

Eine der Möglichkeiten, zusätzliche Informationen zu den angezeigten Daten zu erhalten, ist das Fenster links unten. Dieses zeigt die durchschnittliche Größe für Unternehmen an. Weil die Daten hierbei nicht sehr genau sind, wurden diese gewichtet normalisiert. Dies ergibt die normalisierte durchschnittliche Größe.

Um herauszufinden, wie viele Jobs es in einem bestimmen Bereich gibt, reicht es, die entsprechenden Parameter einzustellen (Zeitraum und Suchwörter) und anschließend auf der Karte zu schauen, in welchem Cluster die meisten Punkte sind. Die genauere Verteilung kann anschließend durch zoomen auf den Cluster herausgefunden werden.

Möchte man wissen, ob sich Jobs über einen Zeitraum besonders häufen, kann man die Chartanzeige herannehmen. Hierfür wählt man die über Regler und Suchbegriffe die entsprechende Menge an Punkten aus, die man angezeigt bekommen möchte. Anschließend geht man mit Mauszeiger auf die Box rechts unten. Es wird ein Chartfenster geöffnet, wie in Abbildung 6.2 zu sehen.

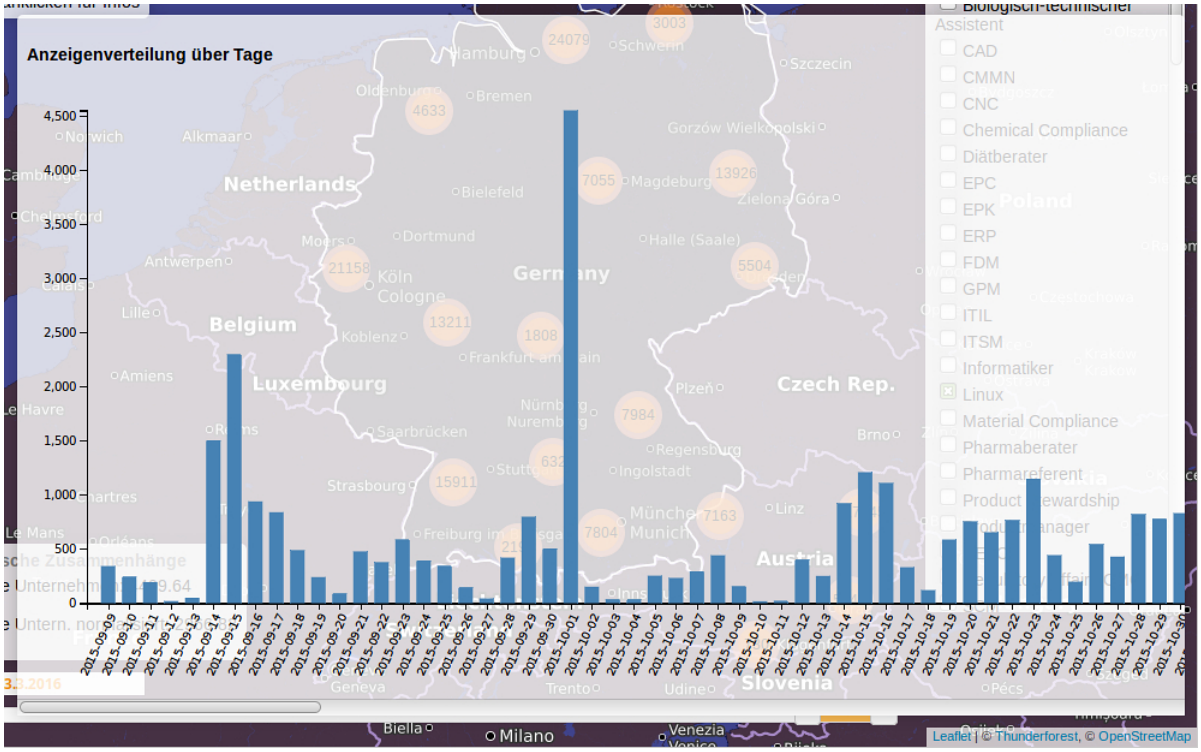

Abbildung 6.2.: Aussehen der Charts für die Anzeige der Verteilung der Stellenangebote über einen Zeitraum

6.2. Eckdaten und Statistiken des Systems

Es wurden verschiedene Messungen durchgeführt und Zahlen erhoben, die das System betreffen. Interessant ist zum einen, wie lange es braucht, um es in Betrieb zu nehmen, also die Extraktion und Analyse von Daten. Zum anderen stellt sich die Frage, wie gut die extrahierten Daten sind, im Hinblick auf die Datenbasis. Außerdem ist interessant, ob das nichtfunktionale Ziel, die Latenz gering zu halten, erreicht wurde.

6.2.1. Geschwindigkeit der Extraktion

Wie in Kapitel 2 erwähnt, wurde die ursprüngliche, sehr große XML-Datei in mehrere kleinere Dateien gesplittet. Aus diesen wurden die Einträge in die Datenbank extrahiert. Die Geschwindigkeit wurde mit unterschiedlich vielen Dateien gemessen, 1.000, 10.000 und allen. Insgesamt gibt es 43.990 aufgesplittete Dateien.

Dabei wurde die Geschwindigkeit auf verschiedenen Speichermedien gemessen: Auf einer herkömmlichen Festplatte, auch Hard Disk Drive (HDD) genannt und auf einem Halbleiterlaufwerk, auch Solid State Disk (SSD) genannt. Hierbei ist die SSD um einige Größenordnungen

schneller, vor allem was wahlfreien Zugriff (Random Access) angeht. Die Daten und die Datenbank lagen hierbei auf dem gleichen Medium, d.h. es wurde von HDD auf HDD und von SSD auf SSD gelesen und geschrieben.

Anzahl Dateien	1.000	10.000	43.990
Geschw. HDD	6m 17s	74m 57s	536m 14s
Geschw. SSD	1m 58s	24m 59s	ca. 178m

Tabelle 6.1.: Geschwindigkeit der Extraktion

6.2.2. Genauigkeit der Extraktion

Wie beschrieben, gab es Inhomogenitäten in den Daten und Fehler im Extraktionsprogramm. Daher fehlen einigen extrahierten Einträgen bestimmte Informationen. Wichtig sind die geographischen Daten (`lat` und `lng`), die für die Positionierung auf der Karte benötigt werden, sowie ob eine Ausschreibung eine `ID` hat, was für die einfache Bestimmung verschiedener Eckpunkte interessant ist, zum Beispiel die durchschnittliche Zeit die eine Anzeige online ist. Eine andere Information ist die Größe der Firma. Diese wird durch das XML-Tag `size` angegeben.

Anzahl Dateien	1.000	10.000	43.990
Anzahl Einträge	81.406	723.806	6.194.515
Einträge ohne `lat` oder `lng`	23.701	197.090	1.743.881
Einträge ohne `ID`	15.679	175.258	1.132.705
Fehlende `ID` + `lat` oder `lng`	38.517	362.405	2.811.263
Einträge mit `size`	68.75 %	78.92 %	79.20 %

Tabelle 6.2.: Eingelesene Punkte und Einlesefehler

6.2.3. Größe der Datenbank

In der folgenden Tabelle sind die Größen der Datenbank aufgeführt, einmal direkt nach dem Einlesen einer bestimmten Anzahl von Dateien und dann nach dem Hinzufügen von Indizes und dem Anlegen von weiteren Tabellen.

Überraschend hierbei ist, dass die Größe nicht linear zu verlaufen scheint, da sie jeweils ungefähr um den Faktor 10 steigt, die Anzahl der Dateien aber nur um den Faktor 4 im letzten Schritt. Dies könnte an der Ungleichverteilung der aufgesplitteten Dateien liegen.

Anzahl Dateien	1.000	10.000	43.990
Nach Einlesen	266 M	2,5 GB	24,1 GB
Indizes + Analyse	284 M	2,6 GB	25,6 GB

Tabelle 6.3.: Größe der Datenbank

Schaut man die Anzahl der Einträge an (Tabelle 6.2), so ist zu sehen, dass die Anzahl der Punkte ebenfalls um den Faktor 10 steigt. Dies bestätigt die Annahme der ungleichen Verteilung von Einträgen auf die aufgesplitteten Dateien.

6.3. Gefundene Zusammenhänge

Hier werden einige Ergebnisse der Analyse aufgeführt. Diese können auf der interaktiven Webseite für eine Auswahl an Arbeitsstellen angezeigt werden. Hier sind die Zusammenhänge für alle verfügbaren Punkte in der Datenbank angegeben.

In Tabelle 6.4 ist die durchschnittliche Größe von Firmen angegeben, für alle Einträge die über ein `size`-Feld verfügen. Hierbei werden wieder die Ergebnisse für verschieden viele eingelesene Dateien angezeigt (Anzahl Dateien).

Anzahl Dateien	1.000	10.000	43.990
Anzahl Einträge	81.406	723.806	6.194.515
`size` verf.	68.75 %	78.92 %	79.20 %
Gemittelt	945.34	1088.90	748.88
Normalisiert	636.29	732.86	504.12

Tabelle 6.4.: Durchschnittliche Größe der Firmen

Da die Größe meistens im Format “50-500“ angegeben ist, konnte die genaue Größe nicht bestimmt werden. Daher wird die Größe einmal gemittelt und einmal normalisiert angegeben. Gemittelt bedeutet hierbei der Durchschnitt der zwei Werte. Für die Normalisierung wurde eine Gewichtung auf die Werte angewendet, wobei als Faktoren 0,7 auf die untere und 0,3 auf die obere Grenzen gewählt wurde.

Ein weiterer Zusammenhang ist, wie in Tabelle 6.5 gezeigt, die durchschnittliche Zeit, die eine Anzeige online ist.

Anzahl Dateien	1.000	10.000	43.990
Zeit in Tagen	4.71	12.20	32.56

Tabelle 6.5.: Durchschnittliche Onlinezeit der Stellenangebote

Hierbei sind die Schwankungen sehr stark. Warum dies so ist, konnte bis zum Ende der Arbeit nicht herausgefunden werden.

6.4. Abgleich Anforderungen

Die Anforderungen wurden in Kapitel 3 definiert. Hier wird evaluiert, ob diese erfüllt wurden.

6.4.1. Funktionale Anforderungen

Die funktionalen Anforderungen waren:

- Übersicht in einer Dimension
- Anzeige von weiteren Informationen

Diese wurden erfüllt. Ein Überblick über alle Arbeitsstellen wurde in der geographischen Dimension gegeben. Außerdem werden weitere Informationen angezeigt, wie über die Chartanzeige (siehe Abbildung 6.2) und die durchschnittliche Größe (siehe Abbildung 6.3).

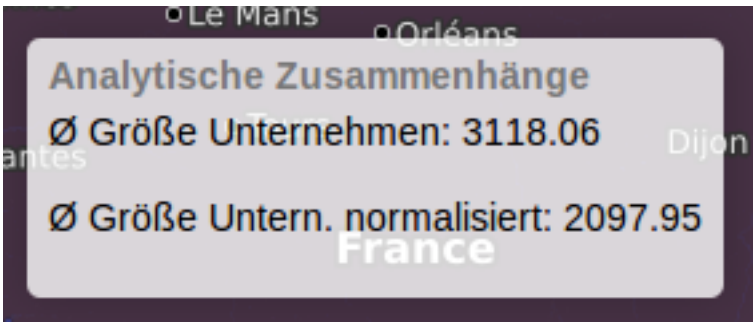

Abbildung 6.3.: Anzeige der durchschnittlichen Firmengröße

6.4.2. Nichtfunktionale Anforderung

Die nichtfunktionale Anforderung war:

- Antwortzeit (Latenz)

Diese wurde teilweise erfüllt. Die Webseite ist langsamer als gewünscht, was auf die Art der Abfrage zurückzuführen ist. Hierbei muss beachtet werden, dass für die Anzeige die Arbeitsstellen so gefiltert werden, dass eine Anzeige, die für mehrere Zeitpunkte vorliegt, nur einmal angezeigt wird. Dies hat eine stark negative Auswirkung auf die Ladezeit.

Anzahl Arbeitsstellen	10.220	101.068
Größe	630 KB	6,45 MB
Ladezeit Browser	9,9 s	10,6 s
Ladezeit Terminal	9,9 s	10,6 s

Tabelle 6.6.: Ladezeit für Arbeitsstellen auf der Webseite

Tabelle 6.6 zeigt, dass die Ladezeit für die Punkte recht konstant ist. Dies ist dem Aufbau des SQL-Queries geschuldet (siehe Anhang B). Dieses verwendet für die Filterung eine so genannte `GROUP BY` Klausel, welche die Anfragen stark verlangsamt, da sie sehr aufwändig ist.

Um der Anforderung gerecht zu werden, dem Benutzer beim Laden vieler Arbeitsstellen Rückmeldung zu geben, wurde ein Fortschrittsbalken in die Webseite integriert (siehe Abbildung 6.4). Allerdings wird dieser erst angezeigt, nachdem die abgefragten Stellenangebote von

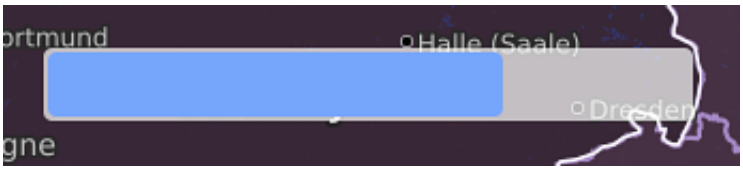

Abbildung 6.4.: Ladebalken beim Laden vieler Stellenanzeigen

dem Webserver in den Browser geladen wurden, was den Großteil der Zeit in Anspruch nimmt. Dies konnte bis zum Abschluß der Arbeit nicht geändert werden und kann noch erforscht werden.

7. Zusammenfassung und Ausblick

Es hat sich gezeigt, dass ein System für die Anzeige von Arbeitsstellen und deren Zusammenhänge grundsätzlich implementierbar ist. Dieses bringt Wissen aus den verschiedensten Bereichen zusammen. Sowohl die Extraktion als auch die Darstellung von Daten sind als Forschungsfelder sehr interessant.

7.1. Zusammenfassung

Jobbörsen haben bisher nicht die Eigenschaft, den Benutzer in seiner Entscheidungsfindung besonders zu unterstützen, sei es dabei ein Stellenangebot zu finden, aufzugeben oder Zusammenhänge auf dem Arbeitsmarkt herauszufinden. Die Suche ist verhältnismäßg beschränkt und nicht interaktiv. Außerdem sind Jobbörsen oft unübersichtlich, was die Darstellung der Ergebnisse angeht. Dies wurde in dieser Arbeit geändert, indem die Arbeitsstellen übersichtlich angezeigt werden, die angezeigten Arbeitsstellen interaktiv veränderbar sind und zusätzliche Informationen zu den angezeigten Arbeitsstellen zur Verfügung gestellt werden.

Um die nötigen Grundlagen für den Leser zu schaffen, wurde die Datenbasis, die eine der Grundlagen der Arbeit ist, erklärt. Außerdem wurden die verwendeten Technologien vorgestellt.

Es wurde definiert, welchen Anforderungen ein System genügen muss, um die genannten Problematiken zu beheben. Hierbei gab es konkrete funktionale Anforderungen, die sich direkt aus der Problemstellung ergeben haben, und nichtfunktionale Anforderungen, die wünschenswert in einem System wie diesem sind.

Es wurden Entwurfs- und Entwicklungsmethode bestimmt, die in diesem Fall eng miteinander zu tun haben. Anhand dieser Methode wurde ein Basisentwurf für das System gemacht. Es wurde erklärt, warum die Entscheidungen im Entwurf getroffen wurden sowie Alternativen vorgestellt. Auch wurden alternative Entwicklungsmethoden kurz angesprochen.

Die generellen Module des implementierten Systems wurden vorgestellt, aus rein technischer Sicht. Die einzelnen Schritte, Probleme und Lösungen im Laufe der Implementierung wurden erläutert.

Die implementierte Webseite wurde gezeigt und eine Einführung gegeben, was Aussehen, Fähigkeiten und Bedienung betrifft. Dazu wurden verschiedene Eckpunkte zum Verhalten des

Systems genannt, wie die Dauer eines Extraktionsvorganges und das Verhalten der Webseite. Es wurde geklärt, ob die Implementierung den ursprünglichen Anforderungen gerecht werden konnte.

Ausblick

Es kann in allen Bereichen, die in dieser Arbeit vorgestellt wurden, noch geforscht und weiter entwickelt werden. Dies gilt sowohl für den speziellen Fall der Jobbörsendaten als auch allgemein.

Die Extraktion könnte verbessert werden, um noch mehr Informationen aus den Daten zu holen. Außerdem könnte mit Hilfe von Named Entity Recognition versucht werden, Namen, Orte und gewünschte Technologien aus den Freitextausschreibungen zu extrahieren, um diese dann auch in die Datenbank zu speichern. Zu den angegebenen Ortsnamen könnte, falls nicht vorliegend, mit einem Geolocationservice die Geolocation auf der Karte gefunden werden, um dies auch noch anzuzeigen. Falls weder Ortsname noch Geolocation vorliegen, könnte versucht werden, diese Informationen über den Firmennamen zu bekommen.

Im Bereich der Speicherung könnte man noch forschen, wie solche Daten schnell vorgehalten werden können, insbesondere in Bezug auf die besonderen Anforderungen der geographischen Verteilung und der Suche nach Zusammenhängen. Auch hier wurde bereits einiges gemacht, allerdings kaum speziell für das Thema Jobsuche.

Im Bereich der Analyse können noch weitere Zusammenhänge gefunden werden. Die Genauigkeit der gefundenen Zusammenhänge kann verbessert werden. Zudem wurden wenig externe Tools eingebunden, die weitere Möglichkeiten eröffnen würden.

Auch bei der Visualisierung können noch weitere Forschungsfragen gestellt werden. Die Visualisierung könnte beschleunigt werden, sodass dem Benutzer die Ergebnisse seiner Anfragen noch schneller zur Verfügung stehen. Hierbei gibt es verschiedene alternative Techniken, die dies ermöglichen. Ein Beispiel hierfür ist das Laden aller verfügbaren Arbeitsstellen in den Browser, damit diese bei weiteren interaktiven Abfragen verfügbar sind. Außerdem könnte die Visualisierung noch genauer auf den Benutzer ausgerichtet werden, d.h. es könnte geschaut werden, was ein Benutzer an Informationen braucht, um bessere Entscheidungen zu treffen. Dies könnte zum Beispiel mit einer Umfrage erreicht werden. Eine weitere Möglichkeit in der Ausrichtung auf einen Arbeitsstellen-suchenden Benutzer wäre eine Bookmark-Funktion, mit Hilfe derer bereits angeschaute Arbeitstellen markiert oder ausgeblendet werden.

Auch wenn man das Finden von Zusammenhängen betrachtet, so können, ausgehend von einer konsistenteren Datenbasis, noch mehr Abhängigkeiten analysiert und gefunden werden. Auch hierfür kann die Visualisierung erweitert werden, um diese Zusammenhänge dem Benutzer einfach visuell erschließbar zu machen.

A. Glossar

BPM Business Process Management.

NLP Natural Language Processing.

NER Named Entity Recognition.

Datum Ein Wert oder ein Faktum (zum Beispiel ”7,13”).

Daten Mehrzahl von Datum.

Information Ein Datum mit Zusammenhang (z.B. ”7,13” und ”€”).

Dateien Auf dem Dateisystem gespeicherte Daten.

Benutzer Person die mit Hilfe eines Systems einen Nutzen bekommt.

Stellenangebot Beschreibung einer Arbeitsstelle mit dem Ziel, einen Arbeitnehmer zu finden, der diese besetzt.

Stellenausschreibung Siehe Stellenangebot.

Jobbörse Hier: Webseite die Stellenangebote veröffentlicht.

strukturiert Hier: Daten mit Semantik, also dem Wissen was sie bedeuten. Siehe Information.

(semi)strukturiert Hier: Teilweise strukturierte Daten.

B. Anhang

B.1. SQL-Queries für die Übertragung von Daten in aggregierte Tabellen

Wie in Kapitel 5 beschrieben, mussten weitere Tabellen eingeführt werden, um bestimmte Anfragen zu beschleunigen. Hier werden die verschiedenen SQL-Anweisungen gezeigt, die genutzt wurden, um die Daten von den verteilten Tabellen in die neu angelegten zu übertragen.

Der SQL-Code für die Übertragung der Daten aus den vorhandenen Tabellen in die neu angelegte ist in Listing B.1 zu sehen.

Listing B.1 SQL-Code für das Übertragen der entsprechenden Informationen in die Tabelle "points"

```
INSERT INTO "points"
   (city, lat, lng, entry_id, datum, search_term, id_on_website)
   SELECT city, lat, lng, entry_id, e.datum, s.search_term, e.id_on_website
   FROM geoloc g
      INNER JOIN entry e ON g.entry_id = e.id
      INNER JOIN search_run s ON e.search_run_id = s.id
   WHERE e.datum LIKE '____-__-__';
```

Der SQL-Code für die Übertragung der Daten aus den vorhandenen Tabellen in die neu angelegte ist in Listing B.2 zu sehen.

Listing B.2 SQL-Code für das Übertragen der entsprechenden Informationen in die Tabelle “point_distribution“

```
INSERT INTO "point_distribution"
   (datum, num, search_term)
   SELECT datum, count(*) AS ‘num‘, s.search_term
   FROM entry e
      INNER JOIN search_run s ON e.search_run_id = s.id
   WHERE datum LIKE ’____-__-__’ GROUP BY e.datum, s.search_term;
```

In Listing B.3 ist der Code für die Übertragung der entsprechenden Informationen zu sehen, für die Tabelle “comp_sizes“.

Listing B.3 SQL-Code für das Übertragen der entsprechenden Informationen in die Tabelle “comp_sizes“

```
INSERT INTO "comp_sizes"
   (name, size, datum, search_term, entry_id)
   SELECT c.name, c.size, e.datum, s.search_term, e.id
   FROM comp c
      JOIN entry e ON c.entry_id = e.id
      JOIN search_run s ON e.search_run_id = s.id
   WHERE size LIKE ’%-%’;
```

B.2. Vergleich der Queries vor und nach Erstellung eigener Tabellen

Listing B.4 zeigt das Query für die Abfrage aller Punkte in einem bestimmten Zeitraum und nach bestimmten Suchbegriffen vor Einführung einer eigenen Tabelle zu diesem Zweck. Hierbei müssen mehrere Tabellen verbunden werden (JOIN), was einen großen zeitlichen Aufwand bedeutet, vor allem bei großen Datenbeständen.

Listing B.4 SQL-Query vor Einführung einer eigenen Tabelle für “points“

```
SELECT city, lat, lng, entry_id
   FROM geoloc g
      INNER JOIN entry e ON g.entry_id = e.id
      INNER JOIN search_run s on e.search_run_id = s.id
   WHERE datum >= :d_min AND datum <= :d_max
   AND ( search_term = :term_ OR search_term = :term_ OR ... ) ;
```

Dieser Aufwand ist in Listing B.5 nicht mehr nötig, da das JOIN nicht mehr verwendet wird, weil die entsprechenden Informationen direkt in der Tabelle verfügbar sind.

Listing B.5 SQL-Query nach Einführung einer eigenen Tabelle für “points“

```
SELECT city, lat, lng, entry_id
   FROM points
   WHERE datum >= :d_min AND datum <= :d_max
      AND ( search_term = :term_$i OR search_term = :term_$i OR ... )
   GROUP BY id_on_website;
```

B.3. Indizes für die Beschleunigung

Es wurden verschiedene Indizes verwendet, um manche Abfragen zu beschleunigen. Diese sind in Listing B.6 zu sehen.

Listing B.6 SQL-Code für die Erstellung von Indizes zur Beschleunigung von Abfragen

```
CREATE INDEX "entry.datum_idx" ON "entry" ("datum");
CREATE INDEX "search_run.search_term_idx" ON "search_run" ("search_term");
CREATE INDEX "geoloc.entry_id_idx" ON "geoloc" ("entry_id");
CREATE INDEX "geoloc.city_idx" ON "geoloc" ("city");
CREATE INDEX "geoloc.lat_idx" ON "geoloc" ("lat");
CREATE INDEX "geoloc.lng_idx" ON "geoloc" ("lng");

CREATE INDEX IF NOT EXISTS "points.full_idx"
   ON "points" ("city", "lat", "lng", "entry_id", "datum", "search_term",
       "id_on_website");

CREATE INDEX IF NOT EXISTS "point_distribution.full_idx"
   ON "point_distribution" ("datum", "num", "search_term");

CREATE INDEX IF NOT EXISTS "comp_sizes.full_idx"
   ON "comp_sizes" ("id", "name", "size", "datum", "search_term", "entry_id");
```

B.4. Testsystem

Alle Zahlen wurden auf dem folgenden System erhoben:

CPU Intel i5-4570 CPU @ 3.20GHz

RAM 16 GB DDR3 Kingston ValueRam

SSD Cynix M4 SSD 256 GB

HDD Seagate BUP USB 4 TB

OS Linux Mint 17.2 Kernel 4.4.0-lowlatency

Browser Firefox 48.0

Webserver PHP 5.5.9 - Integrierter Webserver

Literaturverzeichnis

[AA99] G. L. Andrienko und N. V. Andrienko. „Interactive maps for visual data exploration“. In: *International Journal of Geographical Information Science* 13.4 (1999), S. 355–374 (Zitiert auf S. 11).

[Baa12] B. Baas. „NoSQL spatial: Neo4j versus PostGIS“. In: (2012) (Zitiert auf S. 11).

[Beh16] S. Behnel. *lxml - Processing XML and HTML with Python*. 2016. URL: http://lxml.de/ (besucht am 20. 08. 2016) (Zitiert auf S. 16).

[Bos16] M. Bostock. *Data-Driven Documents*. 2016. URL: https://d3js.org/ (besucht am 07. 08. 2016) (Zitiert auf S. 17).

[BP12] W. R. Bischofberger und G. Pomberger. *Prototyping-oriented software development: concepts and tools*. Springer Science & Business Media, 2012 (Zitiert auf S. 21).

[BR15] F. Baumann und D. Roller. „BPM in German Companies - Information Gathering.“ In: *ZEUS*. 2015, S. 33–37 (Zitiert auf S. 11, 13).

[Bun16a] Bundesagentur für Arbeit. *Faktencheck zum Arbeitsmarkt*. 2016. URL: http://arbeitsmarktmonitor.arbeitsagentur.de/ (besucht am 21. 07. 2016) (Zitiert auf S. 11).

[Bun16b] Bundesagentur für Arbeit. *JOBBÖRSE der Bundesagentur für Arbeit*. 2016. URL: http://jobboerse.arbeitsagentur.de/ (besucht am 21. 07. 2016) (Zitiert auf S. 9, 10).

[Car16] P. Carbonnelle. *PYPL PopularitY of Programming Language*. 2016. URL: http://pypl.github.io/PYPL.html (besucht am 04. 08. 2016) (Zitiert auf S. 17).

[CMQ00] W. W. Cohen, A. McCallum und D. Quass. „Learning to understand the Web“. In: *IEEE Data Eng. Bull.* 23.3 (2000), S. 17–24 (Zitiert auf S. 11).

[Dem16] T. Demachi. *SQLite export plugin for MySQL Workbench*. 2016. URL: https://github.com/tatsushid/mysql-wb-exportsqlite (besucht am 31. 07. 2016) (Zitiert auf S. 29).

[Fur16] A. Furieri. *SpatiaLite: SQLite with Spatial SQL*. 2016. URL: https://www.gaia-gis.it/fossil/libspatialite/index (besucht am 28. 07. 2016) (Zitiert auf S. 11).

[KPSN04] D. A. Keim, C. Panse, M. Sips und S. C. North. „Visual data mining in large geospatial point sets“. In: *Computer Graphics and Applications, IEEE* 24.5 (2004), S. 36–44 (Zitiert auf S. 11).

[Lea16] D. Leaver. *Leaflet Markercluster*. 2016. URL: https://github.com/Leaflet/Leaflet.markercluster (besucht am 31. 07. 2016) (Zitiert auf S. 17, 33).

[LL13] J. Ludewig und H. Lichter. *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*. dpunkt. Verlag, 2013 (Zitiert auf S. 19, 21, 24).

[MCB+11] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh und A. H. Byers. „Big data: The next frontier for innovation, competition, and productivity". In: (2011) (Zitiert auf S. 24).

[Nie93] J. Nielsen. „Response times: The 3 important limits". In: *Usability Engineering* (1993) (Zitiert auf S. 20).

[NWA08] J. Nielson, C. Williamson und M. Arlitt. „Benchmarking modern web browsers". In: *2nd IEEE Workshop on Hot Topics in Web Systems and Technologies*. 2008 (Zitiert auf S. 16).

[Ora16] Oracle Corporation. *MySQL Workbench*. 2016. URL: https://www.mysql.de/products/workbench/ (besucht am 31. 07. 2016) (Zitiert auf S. 29).

[PGH11] F. Pérez, B. E. Granger und J. D. Hunter. „Python: An Ecosystem for Scientific Computing". In: *Computing in Science & Engineering* 13.2 (2011), S. 13–21. URL: http://scitation.aip.org/content/aip/journal/cise/13/2/10.1109/MCSE.2010.119 (Zitiert auf S. 17).

[QSu16] Q-Success. *Historical trends in the usage of server-side programming languages for websites*. 2016. URL: https://w3techs.com/technologies/history_overview/programming_language (besucht am 04. 08. 2016) (Zitiert auf S. 17).

[SCL+99] M. M. Sebrechts, J. V. Cugini, S. J. Laskowski, J. Vasilakis und M. S. Miller. „Visualization of Search Results: A Comparative Evaluation of Text, 2D, and 3D Interfaces". In: *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '99. Berkeley, California, USA: ACM, 1999, S. 3–10. URL: http://doi.acm.org/10.1145/312624.312634 (Zitiert auf S. 9).

[Sme10] F. Smedberg. „Performance analysis of javascript". In: (2010) (Zitiert auf S. 16).

[Tar11] M. Taraldsvik. „Exploring the future: is html5 the solution for gis applications on the world wide web". In: *Norvegian University of science and technology, Department of civil and transport technology* (2011) (Zitiert auf S. 16).

[The16a] The PHP Group. *PHP: Hypertext Preprocessor*. 2016. URL: https://www.php.net (besucht am 04. 08. 2016) (Zitiert auf S. 17).

[The16b] The SQLite Development Team. *SQLite*. 2016. URL: https://www.sqlite.org/ (besucht am 04. 08. 2016) (Zitiert auf S. 17).

[TIO16] TIOBE software BV. *Tiobe Index*. 2016. URL: http://www.tiobe.com/tiobe-index/ (besucht am 04. 08. 2016) (Zitiert auf S. 17).

[VT16] D. Veillard und The GNOME Project. *libxml2*. 2016. URL: http://xmlsoft.org/ (besucht am 20. 08. 2016) (Zitiert auf S. 16).

[WNA08] E. Wästlund, T. Norlander und T. Archer. „The effect of page layout on mental workload: A dual-task experiment“. In: *Computers in human behavior* 24.3 (2008), S. 1229–1245 (Zitiert auf S. 10).

[ZF02] S. Zongyao und B. Fuling. „Spatio-temporal data model based on relational database system“. In: *Geo-spatial Information Science* 5.2 (2002), S. 22–27 (Zitiert auf S. 11).

Alle URLs wurden zuletzt am 14. 08. 2016 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift