



Abschlussprüfung Winter 2016

Fachinformatiker für Anwendungsentwicklung  
Dokumentation zur betrieblichen Projektarbeit

# Unterstützende Suchfunktion

**für den ConSol\*-Profil-Generator**

Abgabetermin: München, den 03.11.2016

**Prüfungsbewerber:**

Florian Müller  
Rathenastr. 124  
80937 München



**Ausbildungsbetrieb:**

CONSOL\* Consulting and Solutions GmbH  
Franziskanerstr. 38  
81669 München

## Inhaltsverzeichnis

<b>Abbildungsverzeichnis</b>	<b>II</b>
<b>Tabellenverzeichnis</b>	<b>II</b>
<b>Listings</b>	<b>III</b>
<b>Abkürzungsverzeichnis</b>	<b>III</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Projektumfeld . . . . .	1
1.2 Projektziel . . . . .	1
1.3 Projektbegründung . . . . .	1
1.4 Projektschnittstellen . . . . .	1
1.5 Projektabgrenzung . . . . .	2
<b>2 Projektplanung</b>	<b>2</b>
2.1 Projektphasen . . . . .	2
2.2 Ressourcenplanung . . . . .	2
2.3 Entwicklungsprozess . . . . .	3
<b>3 Analysephase</b>	<b>4</b>
3.1 Ist-Analyse . . . . .	4
3.2 Wirtschaftlichkeitsanalyse . . . . .	5
3.2.1 Projektkosten . . . . .	5
3.2.2 Amortisationsdauer . . . . .	6
3.3 Anwendungsfälle . . . . .	6
3.4 Soll-Konzept . . . . .	6
<b>4 Entwurfsphase</b>	<b>7</b>
4.1 Risiken und Möglichkeiten . . . . .	7
4.2 Entwurf des Scoringalgorithmus . . . . .	7
4.3 Geschäftslogik . . . . .	9
4.4 Datenmodell . . . . .	9
4.5 Entwurf der Benutzeroberfläche . . . . .	9
4.6 Maßnahmen zur Qualitätssicherung . . . . .	10
<b>5 Implementierungsphase</b>	<b>10</b>
5.1 Implementierung der Datenstrukturen . . . . .	10
5.2 Implementierung der Geschäftslogik . . . . .	11
5.3 Implementierung des Scoring-Algorithmus . . . . .	11
5.4 Implementierung der Benutzeroberfläche . . . . .	12

Abbildungsverzeichnis

<b>6</b>	<b>Abnahmephase</b>	<b>13</b>
<b>7</b>	<b>Einführungsphase</b>	<b>14</b>
<b>8</b>	<b>Dokumentation</b>	<b>14</b>
<b>9</b>	<b>Fazit</b>	<b>14</b>
9.1	Soll-/Ist-Vergleich . . . . .	14
9.2	Lessons Learned . . . . .	15
9.3	Ausblick . . . . .	15
<b>10</b>	<b>Quellen</b>	<b>16</b>
<b>Eidesstattliche Erklärung</b>		<b>17</b>
<b>A</b>	<b>Anhang</b>	<b>i</b>
A.1	Detaillierte Zeitplanung . . . . .	i
A.2	JSON-Modell . . . . .	ii
A.3	Screenshots des Profil-Generators . . . . .	iv
A.4	Proof-of-concept der Korrelationen . . . . .	iv
A.5	Klassendiagramm . . . . .	v
A.6	Datenbankmodell . . . . .	v
A.7	Oberflächenentwurf . . . . .	vi
A.8	Interessante Funktionen . . . . .	vii

## Abbildungsverzeichnis

1	Aufbau der bestehenden Profil-Tabelle . . . . .	4
2	Frontend-Struktur der zu implementierenden Suchseite . . . . .	12
3	Eingabemaske für Fähigkeiten . . . . .	iv
4	Darstellung der Korrelationen zwischen den Fähigkeiten im Live-System mit einer Stärke von $\pm 0,85$ oder mehr . . . . .	iv
5	Klassendiagramm mit Beziehungen der zu implementierenden Klassen . . . . .	v
6	Datenbankmodell . . . . .	v
7	Entwurf der Suchmaske mit statischem HTML und Bootstrap . . . . .	vi

## Tabellenverzeichnis

1	Zeitplanung . . . . .	2
2	Zeitliche Einteilung in Sprints . . . . .	3
3	Soll-/Ist-Vergleich . . . . .	15

## Listings

Listings/profile_model.js . . . . .	ii
Listings/initSkillLinks.java . . . . .	vii
Listings/getProjectScore.java . . . . .	viii

## Abkürzungsverzeichnis

<b>ODT</b>	OpenDocument-Textformat
<b>API</b>	Application Programming Interface
<b>CSS</b>	Cascading Style Sheets
<b>ERM</b>	Entity-Relationship-Model
<b>POJO</b>	Plain Old Java Object
<b>HTML</b>	Hypertext Markup Language
<b>IDE</b>	Integrated Development Environment
<b>MVC</b>	Model View Controller
<b>SQL</b>	Structured Query Language
<b>UML</b>	Unified Modeling Language
<b>CDI</b>	Context and Dependency Injection
<b>DOM</b>	Document Object Model

## 1 Einleitung

### 1.1 Projektumfeld

Die ConSol\* Consulting and Solutions GmbH ist ein mittelständisches Unternehmen mit Hauptsitz in München. Weitere Niederlassungen unterhält ConSol\* in Düsseldorf, Nürnberg, Krakau, Wien, Dubai und San Francisco. Am Standort München ist dazu die Tochter für Internet Solutions und Mediendesign, die alle! GmbH, ansässig.

### 1.2 Projektziel

Ziel des Projekts ist die Entwicklung und Integration einer unterstützenden Suchfunktion für den ConSol\*-Profil-Generator. Durch dieses Feature soll es den Mitarbeitern der Vertriebsabteilung ermöglicht werden, die im ConSol\*-Profil-Generator hinterlegte Liste der Mitarbeiter, hinsichtlich deren Fähigkeiten und Projekterfahrung, adäquat durchsuchen zu können, um potentielle Bewerber für ausgeschriebene Projekte zu finden.

### 1.3 Projektbegründung

Der Prozess zur Auswahl von geeigneten Mitarbeitern für ausgeschriebene Projekte ist aktuell umständlich und zeitaufwendig. Vertriebsmitarbeiter müssen manuell Profile von Mitarbeitern im ODT-Format durchsuchen um Fähigkeiten und Projekterfahrung von Mitarbeitern zu bewerten. Der Zeitaufwand ist unverhältnismäßig, daher ist eine Suchfunktion für die archivierten Mitarbeiterprofile eine geeignete Rationalisierungsmaßnahme um die Effizienz des Auswahlprozesses zu erhöhen.

### 1.4 Projektschnittstellen

Der ConSol\*-Profil-Generator ist in der Version 1.0 bereits in Betrieb. In dieser Webanwendung werden alle Mitarbeiterprofile der Consulting- und Development-Abteilungen zentral aufgenommen und verwaltet. Jeder Nutzer kann dort die Projekte, an denen er beteiligt war, beschreiben, sowie anhand einer definierten Menge von Technologien seine Fähigkeiten bewerten. Der Profil-Generator soll daher um eine Suchfunktion erweitert werden.

Der ConSol\*-Profil-Generator ist mit Hilfe des Spring-Boot Frameworks und Java 8 realisiert worden. Es bietet sich daher an, die Suchfunktion durch Integration einer neuen Spring-Bean in den bestehenden Spring-Kontext umzusetzen. Um die Ergebnisse darzustellen wird eine neue HTML-Seite unter Zuhilfenahme von Bootstrap, Backbone.js und weiterer JavaScript Frameworks wie JQuery und Underscore.js implementiert. Die Suchfunktion wird für die Vertriebsmitarbeiter entwickelt. Sie soll außerdem von allen Mitarbeitern genutzt werden können, um einen fachkundigen Kollegen im Bedarfsfall schneller finden zu können. Die Abnahme der Umsetzung erfolgt durch eine Auswahl der

## 2 Projektplanung

---

Vertriebsmitarbeiter, die die neue Suchfunktion am häufigsten nutzen werden. Auftraggeber ist der Leiter der Delivery-Abteilung, Herr <zensiert>.

### 1.5 Projektabgrenzung

Inhalt des Projekts ist die Entwicklung einer Suchfunktion für die bereits bestehende Anwendung ConSol\*-Profil-Generator. Sie soll UND-verkettete Anfragen verarbeiten können, nicht jedoch ODER- und NICHT-Anweisungen. Der Zustand der bestehenden Features und Programmstrukturen soll soweit wie möglich unverändert bleiben. Ziel ist also nicht die Entwicklung eines eigenständigen, entkoppelten Programmes, sondern die Integration der neuen Suchfunktionalität in den bestehenden Programmkontext.

## 2 Projektplanung

### 2.1 Projektphasen

Das Projekt wird im Zeitraum vom 03.10 bis zum 14.10 durchgeführt. Die Abweichung von der im Projektantrag angegebenen Zeitspanne ist durch einen Berufsschulblock bedingt, der den vorgeschlagenen Zeitraum komplett einnimmt. Die tägliche Arbeitszeit beläuft sich auf durchschnittlich 8 Stunden.

Tabelle 1 zeigt die grobe Zeitplanung.

Projektphase	Geplante Zeit
Planung	13 h
Implementierungsphase	33 h
Kontroll- und Testphase	12 h
Erstellen der Dokumentation	10h
Pufferzeit	2 h
<b>Gesamt</b>	<b>70 h</b>

Tabelle 1: Zeitplanung

Eine detailliertere Zeitplanung findet sich im Anhang [A.1: Detaillierte Zeitplanung](#) auf Seite [i](#).

### 2.2 Ressourcenplanung

Die Ansprechpartner auf Seiten der Vertriebsabteilung sind Herr <zensiert> und Herr <zensiert>. Informationen zur Integration in die bestehende Infrastruktur sind vom Leiter der internen Technik, Herr <zensiert>, einzuholen. Für Informationen zum ConSol\*-Profil-Generator steht der Projektleiter, Herr <zensiert>, zur Verfügung. Betreuer dieser Projektarbeit ist Herr <zensiert>. Die Umsetzung erfolgt an einem Desktop-PC in den Büros der Firma ConSol\* unter Verwendung von Windows 7 und einer

## 2 Projektplanung

Instanz der IntelliJ IDEA, die bereits zu Beginn der Ausbildung lizenziert wurde und für das gesamte Verbleiben in dieser Firma genutzt werden soll. Der ConSol\*-Profil-Generator ist bereits durch die interne Technik in Betrieb genommen worden und läuft auf einem BSD-Unix-basierten Rechner der internen Infrastruktur. Alle weiteren Frameworks, die zur Umsetzung dieser Projektarbeit verwendet wurden, stehen unter einer Open-Source-Lizenz.

### 2.3 Entwicklungsprozess

Als Entwicklungsprozess wurde ein Ansatz ähnlich der Scrum-Methodik gewählt. Da das Projekt von nur einer Person aktiv umgesetzt wurde, fallen klassische Rollen wie Scrum-Master und Product-Owner weg. Es wurde sich auf das Einteilen der zu leistenden Arbeiten in einwöchige Abschnitte ('Sprints') beschränkt. Nach einem Sprint erfolgte die Abnahme der erreichten Ergebnisse durch Herrn <zensiert> als Betreuer dieser Projektarbeit. Dies bot zusätzlich die Möglichkeit der Vertriebsabteilung Zwischenstände nach dem ersten Sprint vorzustellen und die geforderten Grundanforderungen zu verifizieren. In Tabelle 2 ist die zeitliche Gliederung dargestellt.

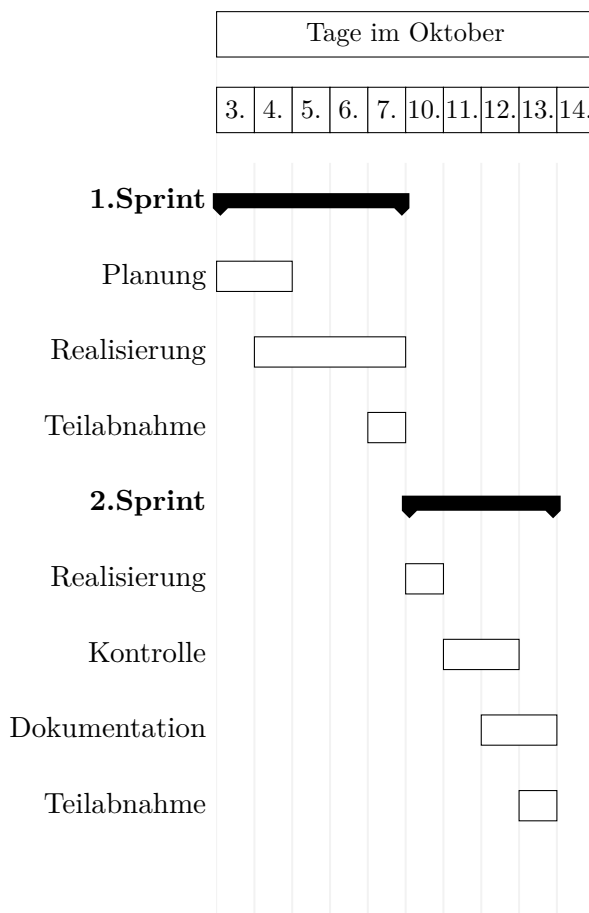


Tabelle 2: Zeitliche Einteilung in Sprints

## 3 Analysephase

### 3.1 Ist-Analyse

Für die Projekt-Akquise sichten die Mitarbeiter der Verkaufsabteilung der ConSol\* Consulting and Solutions GmbH täglich Ausschreibungen und Angebote. Ist ein attraktives Angebot gefunden, muss geprüft werden ob passendes Personal verfügbar ist. Hierzu liegen Dokumente in digitaler Form vor, die den Lebenslauf, die Projekterfahrung und die Fähigkeiten eines Mitarbeiters enthalten. Zur Zeit muss ein Vertriebsmitarbeiter umfassende Kenntnisse über seine Kollegen haben, alle statischen Dokumente durcharbeiten oder aus einer Vorauswahl eines erfahreneren Kollegen wählen, um eine adäquate Wahl treffen zu können. Dieser Arbeitsablauf ist sehr zeit- und kostenintensiv.

Zur zentralen Verwaltung solcher Profile ist der ConSol\*-Profil-Generator bereits in Version 1.0 in Betrieb. Mit Hilfe dieser Webapplikation kann jeder Mitarbeiter seine Daten eigenhändig aufnehmen. Hierzu existieren diverse Eingabemasken, die dem Nutzer erlauben Daten zu seiner Person, seinen Fähigkeiten und seinen Projekten aufzunehmen. Eine exemplarische Darstellung der Maske zur Eingabe von Kenntnissen aus der fest definierten Menge an Fähigkeiten ist im Anhang [A.3: Screenshots des Profil-Generators](#) auf Seite [iv](#) aufgeführt. Der Großteil aller Profile ist bereits erfasst und steht zur Verfügung. Diese werden intern pseudo-dokumentenorientiert in einer MariaDB persistiert. Das bedeutet, dass die Daten im JSON-Format direkt in eine Spalte der Table 'Profil' geschrieben werden.

Column	Type
◇ UID	int(11)
◇ JSON	mediumtext
◇ DATE	datetime
◇ VERSION	int(11)
◇ VALID	tinyint(1)

Abbildung 1: Aufbau der bestehenden Profil-Tabelle

Die Spalte 'UID' beinhaltet die eindeutige Identifikationsnummer des Profils, 'Date' speichert das letzte Änderungsdatum, 'Version' ist der Versionszähler und 'Valid' gibt an, ob das Profil das momentan valide Profil ist. Es kann immer nur ein Profil pro 'UID' als valide markiert sein.

Interessant für die Suchfunktion ist an dieser Stelle die Struktur des JSON-Modells. Ein Listing findet sich im Anhang [A.2: JSON-Modell](#) auf Seite [ii](#). Relevant sind die Daten aus dem Array 'projects' und aus dem Array 'skills'. Dabei sind die Felder 'projects.industryField' und 'projects.toolsEnvironment' Freitextfelder. Die Fähigkeiten, repräsentiert durch das Array 'skills', sind aus einer Menge an vorher festgelegten Optionen auszusuchen und zu bewerten. Der Wertebereich erstreckt sich von 0 (keine Kenntnisse) bis 4 (Experte).

Die Kommunikation mit der Datenbank erfolgt über eine Spring-Boot-Applikation. Durch Spring-Bean-Definitionen und die Nutzung der CDI-Unterstützung kann die Geschäftslogik in diesem Framework sehr einfach von der Client-Server-Kommunikation entkoppelt werden, was die Erweiterbarkeit



### 3 Analysephase

---

der Anwendung vereinfacht. Hier wurden zur Kommunikation mit dem Frontend ein 'MainController' und ein 'ApiController', welcher als REST-Schnittstelle fungiert, implementiert. Dabei verweist der 'MainController' jede valide Anfrage auf dieselbe 'index.html' Seite. Diese ist als One-Page-Applikation entwickelt worden, was bedeutet, dass sie mit einem speziellen Framework - in diesem Fall Backbone.js - durch dynamische Manipulation des DOMs verschiedene Seitenansichten in einem einzigen HTML-Dokument darstellen kann.

## 3.2 Wirtschaftlichkeitsanalyse

Durch die Umsetzung der Suchfunktion wird eine Effizienzsteigerung im unter 3.1 beschriebenen Arbeitsablauf angestrebt. Sie soll außerdem die Einarbeitungszeit für neue Mitarbeiter, die diesen Arbeitsablauf durchführen sollen, minimieren. Da der Datenbestand bereits durch den ConSol\*-Profil-Generator sichergestellt ist, ist der Entwicklungsaufwand für die Suchfunktion überschaubar. Diese Punkte rechtfertigen die Durchführung des Projekts in wirtschaftlicher Hinsicht.

### 3.2.1 Projektkosten

Ein Auszubildender verdient im zweiten Lehrjahr tariflich im Monat 900,00€ Brutto. Nachfolgend gehe ich vereinfacht von 20 Arbeitstagen im Monat aus.

$$8 \text{ h/Tag} \cdot 20 \text{ Tage/Monat} = 160 \text{ h/Monat} \quad (1)$$

$$\frac{900 \text{ €}}{160 \text{ h/Monat}} \approx 5,63 \text{ €/h} \quad (2)$$

Daher ergibt sich ein Stundenlohn von 5,63 €. Die Projektdauer ist auf 70 Stunden ausgelegt. Für die Nutzung der Büroressourcen werden pauschal Kosten in Höhe von 10 € angesetzt. Da die Lizenzgebühren für die Entwicklungsumgebung (IntelliJ IDEA) bereits zu Beginn der Ausbildung bezahlt wurden und für die gesamte Dauer meiner Anstellung bei ConSol\* verrechnet werden müssten, werden sie in diesem Fall von der Ressourcenpauschale mit abgedeckt. Der Lohn der Mitarbeiter der Vertriebsabteilung wird pauschal auf 30 € pro Stunde geschätzt. Die zwei Ansprechpartner aus dieser Abteilung werden zusammen für 2 Stunden in einem Fachgespräch gebraucht und müssen das Projekt nach jedem Sprint abnehmen. Hierfür wird 1 Stunde pro Mitarbeiter veranschlagt.

$$70h \cdot 5,63 \text{ €/h} = 394,10 \text{ €} \quad (3)$$

$$70h \cdot 10 \text{ €/h} = 700 \text{ €} \quad (4)$$

$$6h \cdot 30 \text{ €/h} = 180 \text{ €} \quad (5)$$

$$394,10 \text{ €} + 700 \text{ €} + 180 \text{ €} = 1274,10 \text{ €} \quad (6)$$

Die Gesamtkosten für die Umsetzung des Projekts belaufen sich also auf **1274,10 €**.

### 3.2.2 Amortisationsdauer

Geht man von einer Zeitersparnis von 10 Minuten pro Suchanfrage (Schätzung aufgrund des in 3.1 beschriebenen Arbeitsprozesses) und durchschnittlich 25 Suchen pro Woche aus, so ergibt sich eine wöchentliche Einsparung von 250 Minuten pro Mitarbeiter der Verkaufsabteilung. Diese Werte können je nach Auftragslage und Auslastung der Abteilung stark schwanken und sind nur ungefähre Schätzwerte. Auf Basis dieser Werte erhält man bei nur zwei aktiven Nutzern pro Woche eine Zeitersparnis von 500 Minuten, was nach obiger Stundenpauschale einer Ersparnis von 250 € gleichkommt. Geht man weiter von vier Arbeitswochen pro Monat aus ergibt sich eine monatliche Ersparnis von 1000 € und eine jährliche Ersparnis von 12000 €.

Daher ist mit einer Amortisationsdauer von  $\frac{1274,10 \text{ €}}{12000 \text{ €/Jahr}} \approx 0,11 \text{ Jahre} \approx 6 \text{ Wochen}$  zu rechnen.

## 3.3 Anwendungsfälle

Es existiert der folgende Anwendungsfall:

- Ein Nutzer sucht einen Mitarbeiter anhand von festgelegten Fähigkeiten und/oder Branchen über ein Freitextfeld im ConSol\*-Profil-Generator.

Wegen der Einfachheit dieses Anwendungsfalls wird auf eine grafische Darstellung verzichtet.

## 3.4 Soll-Konzept

Der Arbeitsfluss zur Auswahl passender Mitarbeiter für anstehende Projekte soll durch die Integration einer Suchfunktion in die bestehende Anwendung des Profil-Generators effizienter gestaltet werden. Über ein Freitextfeld sollen logische, und-verknüpfte Suchanfragen an die Anwendung gesendet werden können. Alle relevanten Felder eines Profils müssen semantisch korrekt durchsucht werden und ein Scoring-Mechanismus soll passende Profile sortiert zurückliefern. Die Darstellung der Profile muss überschaubar in das Front-End der bestehenden Applikation integriert werden. Dabei sollen die bestehenden Softwarestrukturen so wenig wie möglich abgeändert werden.

#### 4 Entwurfsphase

---

Im Fachgespräch mit den Mitarbeitern der Vertriebsabteilung wurde festgestellt, dass die für die Suche wichtigen Informationen auf die Fähigkeiten und Projekte eines Profils beschränkt werden können. Aus technischer Sicht handelt es sich um die Felder 'projects' und 'skills' aus dem JSON-Modell (Anhang A.2: [JSON-Modell](#) auf Seite ii). Die relevanten Informationen des Arrays 'projects' wurden weiterhin beschränkt auf die Branche, die eingesetzten Werkzeuge und den Zeitraum eines Projekts. Dies entspricht im Listing den Feldern 'projects.industryField', 'projects.toolsEnvironment' und 'projects.fromDate' sowie 'projects.toDate'.

## 4 Entwurfsphase

### 4.1 Risiken und Möglichkeiten

In der Analysephase wurde festgestellt, dass die Nutzer der neuen Suchfunktion ausschließlich nach Fähigkeiten, Branchen und eingesetzten Werkzeugen suchen.

Vorteilhaft ist, dass alle Fähigkeiten aus einer fest definierten Menge gewählt werden müssen, die vom Administrator des ConSol\*-Profil-Generators bearbeitet werden kann. Die Bewertung ist jedoch immer eine subjektive Eigeneinschätzung des Mitarbeiters. Das führt zwangsläufig zum Über- oder Unterschätzen der eigenen Fähigkeiten. Vorteilhaft ist also eine Glättung der Werte in der potentiellen Suchmenge.

Die vom Mitarbeiter in den Projekten verwendeten Werkzeuge und Technologien werden hingegen als eine Liste von frei wählbaren Begriffen aufgenommen. Diese Tatsache hat zur Folge, dass z.B. für die Programmiersprache Java in unterschiedlichen Profilen abweichende Bezeichnungen auftauchen. Als Beispiel gibt Person A 'JDK 1.6' und Person B 'Java 6' an. Eine Prüfung auf lexikalische Gleichheit entfällt, denn gängige Algorithmen, wie z.B. die Levenstein-Distanz, würden diese beiden Begriffe als zu unterschiedlich einstufen, als dass man mit der gleichen Anfrage beide finden könnte. Es bietet sich also an, die undefinierte Menge an Werkzeugen mit der klar definierten Menge an Fähigkeiten zu verbinden. Für obiges Beispiel würde das bedeuten, dass die Werkzeuge 'JDK 1.6' und 'Java 6' beide auf die selbe Fähigkeit 'Java' abgebildet und somit über den selben Suchbegriff gefunden werden können. Setzt man dieses Mapping um, bietet das die Möglichkeit, einzig nach den vorher klar definierten Fähigkeitsbegriffen zu suchen und trotzdem die Projekterfahrung mit einzubeziehen.

### 4.2 Entwurf des Scoringalgorithmus

Die einfachste mögliche Suche ist im Kontext des ConSol\*-Profil-Generator die nach einer Fähigkeit. Hierzu muss man lediglich alle Profile nach dem exakten Suchbegriff durchsuchen und passende Ergebnisse zurückliefern. Sortiert man diese Ergebnisse nach den Eigenbewertungen der gesuchten Fähigkeit erhält man ein erstes einfaches Scoring. Dieser Prozess stellt sicher, dass ein Mitarbeiter, der sich als Experte einstuft immer vor einem Mitarbeiter, der Grundkenntnisse angegeben hat, eingestuft wird. Diese erste Iteration reicht für die vorgesehene Nutzung der Suchfunktion aus.

Interessanter ist das Einbeziehen von angegebenen Projekten. Hier bietet sich die Möglichkeiten Über-

#### 4 Entwurfsphase

---

und Unterschätzungen der Mitarbeiter bei der Bewertung ihrer Fähigkeiten zu glätten. Anders als die eigene Einschätzung transportiert der Datensatz eines Projekts Metadaten, die rein objektiv zu betrachten sind. Dazu gehören die verwendeten Werkzeuge, die Länge des Projekts und die Branche, in der das Projekt umgesetzt wurde.

Durch die Datenbasis der Profile bietet sich die Möglichkeit Beziehungen zwischen den Fähigkeiten herzustellen. Grundgedanke ist, wer sich mit Fähigkeit X gut auskennt, wird sich mit hoher Wahrscheinlichkeit auch mit Fähigkeit Y in bestimmten Maße auskennen. In der empirischen Statistik kann ein solcher Zusammenhang mit einem Korrelationskoeffizienten ausgedrückt werden. Man durchläuft also alle Profile und sucht dabei nach Fähigkeit X und Fähigkeit Y. Sind beide Fähigkeiten in einem Profil aufgeführt dient dieser Datensatz zur Berechnung des Korrelationskoeffizienten zwischen Fähigkeit X und Y. Da die Bewertungsoptionen, wie in der Ist-Analyse beschrieben, einen Wertebereich von [0..4] einnehmen ist die Bestimmung eines solchen Korrelationskoeffizienten möglich. Hierzu wurde vor der Durchführung des Projekts ein Proof-of-concept (nicht Bestandteil der Projektarbeit) angefertigt, welcher die Korrelationen aus dem Datenbestand des Live-Systems analysiert und Korrelationen mit ausreichender Stärke in einem gewichteten Punktdiagramm darstellt. Eine Darstellung ist im Anhang [A.4: Proof-of-concept der Korrelationen](#) auf Seite [iv](#) zu finden. Hat man Korrelationen mit ausreichendem Gewicht ist es möglich unter Zuhilfenahme der empirischen Regressionsanalyse eine Vorhersage über eine Fähigkeit Y auf Basis einer Fähigkeit X zu treffen. Dazu werden die Werte der Selbsteinschätzung genutzt. Kombiniert man das Mapping der Werkzeuge auf die Fähigkeiten mit der Regressionsanalyse kann man also einen wahrscheinlichen Wert für eine Fähigkeit aus den Werkzeugen eines Projekts ableiten.

**Beispiel:** Ein Nutzer sucht Profile mit dem Suchbegriff 'Java'. 'Java' korreliert ausreichend stark mit 'Spring'. Hat ein Mitarbeiter in einem seiner Projekte ein Werkzeug angegeben, dass auf die Fähigkeit 'Spring' abgebildet ist kann der Wert der Eigeneinschätzung für 'Spring' in die Regressionsgleichung von 'Spring' und 'Java' eingesetzt werden und ein wahrscheinlicher Wert für die Fähigkeit 'Java' in diesem Projekt errechnet werden. Der errechnete Wert repräsentiert damit den wahrscheinlichen Durchschnittswert der Java-Bewertung für Mitarbeiter, die 'Spring' mit Wert A und 'Java' mit Wert B bewertet haben. Weiterhin kann für jedes Projekt und seine Werkzeuge der Projektzeitraum mitbezogen werden. Dieser Effekt hilft dabei die Ausreißer in den Selbsteinschätzungen zu glätten und die Projekte eines Profils adäquat zu bewerten.

Diese Überlegungen berücksichtigt folgende Formel zur Berechnung des Wertes eines Profils. Hier sind x die gesuchte Fähigkeit, xb die Bewertung dieser Fähigkeit, p das Profil, i die Anzahl der Projekte des Profils, y das Werkzeug und j die Anzahl der Werkzeuge eines Projekts. Die Funktion f(x,y) berechnet den linearen Regressionswert von x in Beziehung zu y und die Funktion b(p) berechnet den Branchenwert, falls dieser Teil der Suchanfrage ist.

$$100 \cdot x_b + \sum_{i=1}^{NumProjekte} \left( \left[ \sum_{j=1}^{NumWerkzeuge(Projekt(i))} f(x, y_j) \cdot \frac{Projektdauer}{365} \right] + b(p_i) \right) \quad (7)$$

### 4.3 Geschäftslogik

Die Wahl des Architekturdiseigns orientiert sich am bereits bestehenden Programmkontext. Hier wird durchgehend ein Model-View-Controller Muster verwendet. Daher wird der oben beschriebene Bewertungsalgorithmus und die Steuerung der Suchanfrage mit einer Spring-Bean realisiert, welche durch CDI in den ApiController injiziert wird. Außerdem wird ein neues DAO als weitere Spring-Bean umgesetzt, die ihrerseits in die neue Controller-Bean injiziert wird (Anhang [A.5: Klassendiagramm](#) auf Seite [v](#)).

Das bestehende Model 'profile.java' wird um ein Feld 'searchValue' erweitert, das den vom Bewertungsalgorithmus errechneten Wert für die jeweilige Suche speichert. Zur Kommunikation mit dem Frontend wird ein Wrapper-Model benötigt. Es muss die zu suchende Zeichenkette zum Server und die gefundenen Profile zurück zum Client transportieren können. Eine eintreffende Suchanfrage wird also über den ApiController an den SearchController weitergeleitet, welcher mit Hilfe des neuen DAO zutreffende Profile findet, sie der berechneten Bewertung entsprechend ordnet und an den ApiController zurück gibt, welcher dann die Daten im JSON-Format an den Client sendet. Weiterhin muss die Initialisierungsfunktion für die Synchronisierung der Daten in der Controller-Bean so implementiert werden, dass sie einmal am Tag zu einem Zeitpunkt ausgeführt wird, an dem das System am wenigsten ausgelastet ist.

### 4.4 Datenmodell

Die Aufbereitung der Datenbasis erfolgt durch die Umwandlung der JSON-Daten (siehe Anhang [A.2: JSON-Modell](#) auf Seite [ii](#)) der bereits vorhandenen 'Profile' Tabelle in ein relationales Datenbankmodell, das alle Daten abdeckt, die in der Analysephase als relevant eingestuft wurden. Dabei handelt es sich um die Werkzeuge, die in den Projekten verwendet wurden, die zu bewertenden Fähigkeiten und die Branchen, denen ein Projekt zuzuordnen ist. Außerdem werden Zwischentabellen benötigt, die die Zuordnungen von Werkzeugen, Branchen und Fähigkeiten zu Profilen abbilden. Es wird bewusst auf die komplette Überführung des JSON-Modells in ein relationales Datenbankmodell verzichtet um die Redundanz der Daten so gering wie möglich zu halten. Eine Abbildung der Projekte an sich ist nicht notwendig, da man die relevanten Felder direkt einem Profil zuordnen kann. Daraus ergibt sich das im Anhang [A.6: Datenbankmodell](#) auf Seite [v](#) abgebildete Datenbankmodell. Zur Modellierung wurde das Werkzeug 'MySQL Workbench' verwendet.

### 4.5 Entwurf der Benutzeroberfläche

Da bereits eine Benutzeroberfläche zur Eingabe und Administration der Daten existiert, wurde die neue Suchmaske an das bestehende Design angepasst. Ein Entwurf konnte durch statisches HTML und die Verwendung von Bootstrap erstellt werden und ist im Anhang [A.7: Oberflächenentwurf](#) auf Seite [vi](#) zu finden. Dieses Vorgehen hat den Vorteil, dass das daraus entstandene Konzept mit wenig

## 5 Implementierungsphase

---

Änderungsarbeit für die Implementierung genutzt werden kann. Neben einem Freitextfeld mit automatischer Vervollständigung und einem Knopf zum Absenden der Suchanfrage wurde eine Checkbox eingeplant, mit der die Einbeziehung der Projekterfahrung in das Suchergebnis eingestellt werden kann. Das Suchergebnis wird tabellarisch dargestellt. Um einen Überblick zu gewährleisten werden die Anzahl der getroffenen Fähigkeiten und Projekte sowie das letzte Änderungsdatum des Profils ausgegeben. Am linken Ende jeder Zeile wird ein Knopf positioniert, mit dem man eine Darstellung der Projekte und Fähigkeiten des zugehörigen Profils auf- und zuklappen kann. Mit einem Klick auf den Link hinter einem Profilnamen öffnet sich ein neuer Reiter, der alle Daten des Profils detailliert darstellt. Zwischen den Reitern kann navigiert werden um mehrere Profile einzusehen. Offene Reiter können durch einen Klick auf das 'X'-Symbol des Reiters geschlossen werden.

### 4.6 Maßnahmen zur Qualitätssicherung

Um eine ausreichende Qualität sicherzustellen wird vor dem Beginn der Implementierungsphase eine Testumgebung im Spring-Boot-Kontext eingerichtet, in der automatisierte Tests angestoßen werden. Dazu bringt Spring-Boot eine In-Memory H2 Datenbank mit. Diese kann zur Ausführung der Testfälle mit einem Schema und dazugehörigen Testdaten über zwei Dateien ('schema.sql', 'data.sql') befüllt werden. Das schafft eine immer gleiche Ausgangssituation und ermöglicht die Vorhersage von Ergebnissen. Ein weiterer Vorteil ist, dass man keine Komponenten simulieren muss und sowohl Unit- wie auch Integrations- oder End-to-End-Tests im selben Test-Kontext implementieren kann.

Die neue Suchfunktion soll in zweierlei Hinsicht getestet werden. Zum einen müssen die korrekte Ausführung der täglichen Datensynchronisation und die richtige Funktionsweise des Suchalgorithmus getestet werden: Dazu kann der neue Controller in den Test-Kontext injiziert und seine Funktionen in Unit-Tests angestoßen werden. Zum anderen müssen die Client-Server-Kommunikation und die Darstellung der Suchergebnisse auf der Benutzeroberfläche getestet werden. Um dies zu realisieren wird Phantom.js - ein headless Browser - verwendet, der durch einen Selenium Webdriver gesteuert wird. Dadurch kann eine Benutzerinteraktion auf dem Frontend simuliert und deren Ergebnis getestet werden. Dazu werden relevante Elemente des DOM ausgelesen und deren Inhalt überprüft.

## 5 Implementierungsphase

### 5.1 Implementierung der Datenstrukturen

Die Implementierung der Datenstrukturen erfolgt auf Grundlage des in 4.4 erarbeiteten Datenbankmodells. Hierzu werden alle Tabellen durch manuelle SQL-Anweisungen auf dem bestehenden Schema der Entwicklungsdatenbank erstellt. Dabei wird eine exakte Abbildung des im Anhang A.6: [Datenbankmodell](#) auf Seite v erarbeiteten Modells realisiert, um zu gewährleisten, dass die Anforderungen an die Datenstrukturen erfüllt sind.

## 5.2 Implementierung der Geschäftslogik

Zunächst wurde ein neues Java-Paket definiert, um die neue Suchfunktion bereits auf Paketebene vom Profil-Generator zu entkoppeln. Anschließend wurden die zwei neuen POJO's ('SearchController.java', 'SkillSearchDao.java') als Klassen erstellt. Über die Klasse 'WebConfiguration', die mit der Annotation '@Configuration' markiert ist, werden die neuen Klassen mit Hilfe der Annotation '@Bean' einer Getter-Funktion, die den jeweiligen Standardkonstruktor der Klassen aufruft, im CDI-Kontext der Applikation bereitgestellt. Etwaige Felder, die mit dem Datentyp der Klassen, die als Spring-Bean definiert sind, ausgezeichnet sind können anschließend mit der Annotation '@Autowired' automatisch befüllt werden. Auf diese Weise werden der SearchController im ApiController und der SkillSearchDao im SearchController bekannt gemacht. Dieses System hat den Vorteil, dass es immer nur ein Laufzeitobjekt einer Spring-Bean geben soll, das eindeutig identifiziert und somit in anderen Spring-Beans verwendet werden kann, ohne dass jedes Mal neue Objekte erstellt werden müssen.

Nachdem die neuen Spring-Beans erstellt und registriert waren, wurden die Initialisierungsfunktionen erstellt, die die Tabellen 'skills', 'tool' und 'field\_of\_buisness' auf Grundlage der Datenbasis aus dem Feld 'JSON' der Tabelle 'Profile' und den Informationen der Tabelle 'skilloption' befüllen. Dazu werden die Spring-Beans 'ProfileDao' und 'SkillOptionsDao' nach den oben beschriebenen Verfahren im 'SearchController' bekannt gemacht. Über diese beiden Data-Access-Objekte werden die Profil- und Fähigkeitsdaten bezogen. Der 'SearchController' iteriert über die jeweilige Datenbasis und ruft die passende, im 'SkillSearchDao' implementierte Funktion zur Persistierung des Datensatzes in der zugehörigen neuen Tabelle auf.

Anschließend wurden die Funktionen erstellt, die die Beziehungen von Fähigkeiten, Werkzeugen und Branchen zu den Profilen herstellen. Hierzu wurden alle Profile durchlaufen, die Beziehungen ausgelesen und in die neuen Zwischentabellen geschrieben.

Der letzte Schritt der Initialisierung der Datenbasis ist die Berechnung der Korrelationen zwischen den Fähigkeiten. Dazu wurde eine Funktion erstellt, die alle Profile nach allen möglichen Fähigkeitspaaren durchsucht. Sind alle Profile für ein Fähigkeitspaar durchlaufen wird die Korrelation und die Regressionsgleichung errechnet und über den 'SkillSearchDao' in die Tabelle 'skill\_link' geschrieben. Diese Funktion ist exemplarisch im Anhang [A.8: Interessante Funktionen](#) auf Seite [vii](#) gelistet. All diese Funktionen wurden letztlich in einer öffentlichen Funktion zusammengefasst, die mit der Annotation '@Scheduled(cron = "0 0 3 \* \* \*")' versehen wurde, um die Funktion jede Nacht um 03:00 Uhr auszuführen und damit die Aktualität der Datenbasis zu gewährleisten.

Die Verbindung der Projektwerkzeuge mit Fähigkeiten erfolgt manuell durch einen Administrator über eine Administrationsseite. Die Realisierung dieses Teils der Benutzeroberfläche ist nicht Teil dieser Projektarbeit.

## 5.3 Implementierung des Scoring-Algorithmus

Als letzter Schritt bei der Implementierung des Backends wurde die eigentliche Suchfunktion umgesetzt. Diese nimmt als Argument eine Zeichenkette entgegen, die mehrere durch ein kaufmännisches 'und' ('&') getrennte Suchbegriffe enthalten kann und trennt diese Begriffe. Anschließend ruft sie die



## 5 Implementierungsphase

im 'SkillSearchDao' implementierte Funktion mit den ermittelten Suchbegriffen auf. Diese identifiziert über einen Zeichenkettenvergleich auf der Datenbank die ID's der passenden Fähigkeiten und/oder Branchen und gibt die Profile zurück, die über die oben genannten Zwischentabellen mit diesen Branchen oder Skills verbunden sind. Ist das Flag zur Berücksichtigung der Projekterfahrung nicht gesetzt, werden die Profile nach dem im ersten Absatz von 4.2 beschriebenen Verfahren bewertet und zurück an den Client gesendet.

Zur Bewertung der Projekterfahrung wurde die Funktion 'getProjectScore' im 'SearchController' implementiert. Sie berechnet nach der Formel aus dem letzten Absatz von 4.2 den für die Bewertung relevanten Wert für alle vorher gefundenen Profile. Sie ist im Anhang A.8: [Interessante Funktionen](#) auf Seite viii gelistet.

### 5.4 Implementierung der Benutzeroberfläche

Da der Fokus dieser Projektarbeit auf der Umsetzung des Scoring-Algorithmus liegt, wird die Umsetzung des Frontends nur kurz erläutert.

Die Implementierung der Benutzeroberfläche erfolgte auf der Basis des HTML-Codes, der im Abschnitt 4.5 entwickelt wurde. Das Grundgerüst dieses Entwurfes wurde in statische und dynamische Inhalte eingeteilt. Die statischen Elemente des DOMs wurden in einem Template abgelegt. Zur Manipulation des Templates und zur Kommunikation mit dem Server wird das Framework 'Backbone.js' verwendet. Durch asynchrone Anfragen an den 'ApiController' können zur Laufzeit Daten an der REST-Schnittstelle abgefragt und ins DOM eingefügt werden. Backbone.js arbeitet nach einem vereinfachten 'Model-View-Controller' Muster. Die Darstellung einer Seite besteht aus einer HTML-Vorlage (View), einem JavaScript-Objekt, das die kommunizierten Daten hält (Model) und einem JavaScript-Objekt, welches die Nutzerinteraktion auswertet und das Modell-Objekt manipulieren kann (Controller).

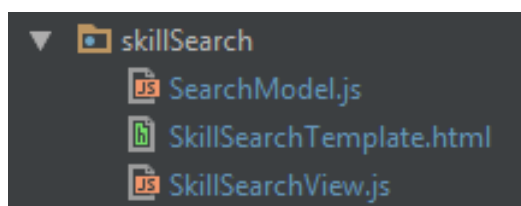


Abbildung 2: Frontend-Struktur der zu implementierenden Suchseite

Hierbei wird die Datei 'SkillSearchTemplate.html' nicht direkt vom Browser interpretiert. Zunächst werden Platzhalter im HTML-Code mit Realdaten gefüllt. Anschließend wird via JavaScript das Ziel-DOM-Objekt, welches in der Datei 'SkillSearchView.js' definiert ist, geleert und mit dem neuen Inhalt gefüllt. Die 'SkillSearchView.js' fungiert also als Controller, hat ihren Namenszusatz 'View' aber aus Gründen der Konvention. Im 'Backbone.js' Kontext muss eine Klasse namens 'View' erweitert werden, um einen solchen Controller zu implementieren. Hier wurden alle möglichen Interaktionen im 'events'-Array definiert und die zugehörigen Funktionen ausformuliert. 'Backbone.js' arbeitet mit einem 'Router'-Objekt, das auf Basis von URL-Pfaden zugehörige Controller aufruft. Wird der Controller der Suchseite auf diese Weise aufgerufen, initialisiert er sein Model-Objekt. Wird ein Suchbegriff



eingegeben und bestätigt, wird das Modell damit angereichert und an den Server gesendet. Kommt eine valide Antwort am Client an, wird das 'sync'-Event des Models getriggert und eine daran gebundene Funktion ausgelöst, die das Template mit den erhaltenen Daten anreichert und letztlich dargestellt.

## 6 Abnahmephase

Alle Tests werden während des Build-Prozesses der Maven Umgebung angestoßen und sollen sicherstellen, dass das Projekt nur gebaut werden kann, wenn alle Testfälle erfolgreich durchlaufen.

Die Funktionsfähigkeit der 'SearchController'-Strukturen wurde im unter 4.6 beschriebenen Kontext getestet. Dazu wurde nach der Implementierung einer Teilfunktion der täglichen Datensynchronisierung, der zugehörige Unit-Test implementiert. Diese Tests folgen zunächst alle dem selben Schema: Die entsprechende Funktion wird in einem Unit-Test angestoßen. Nach der Ausführung wird mit Hilfe eines Test-Data-Access-Objects, der vom 'SkillSearchDao' erbt und durch Wrapper-Funktionen, spezielle Funktionen zum Lesen von Daten in der Datenbank offenlegt, die korrekte Ausführung getestet, indem die Korrektheit der Daten, die der 'SearchController' in die Datenbank geschrieben hat, überprüft wird.

Um die korrekte Funktionsweise des Such- und Scoringalgorithmus zu verifizieren wurden die immer gleichen Ausgangsdaten auf der H2-Datenbank des Testkontext genutzt (siehe 4.6). Dazu wurde zunächst ein Schreibtischtest für die Berechnung der Korrelationen und einem ausgewählten Scoring einer Fähigkeit durchgeführt und die Ergebnisse zur Verifikation der vollautomatischen Unit-Tests verwendet. Die Suchfunktion des Searchcontrollers wird manuell mit einer vorher definierten Fähigkeit als Suchanfrage angestoßen. Diese Funktion liefert letztlich eine nach Bewertung geordnete Liste an Profilen zurück. Die Bewertungen und die Reihenfolge der Profile werden anschließend auf Korrektheit geprüft.

Um die Dargestellung der Ergebnisse zu testen, wird mit einem für Selenium implementierten Webdriver des headless Browsers Phantom.js gearbeitet. Dies ist entscheidend für die Ausführung der Tests auf Servermaschinen ohne graphischen Kontext. Auch diese End-to-End-Tests arbeiten mit der Datenbasis der H2-Datenbank. Über DOM-Selektoren können die einzelnen HTML-Elemente ausgewählt und manipuliert werden. Zum Testen der Suchfunktion werden daher in das Suchfeld ausgewählte Suchbegriffe eingegeben und die Suche über den 'Suchen'-Knopf vom Frontend aus angestoßen. Ist die Suche abgeschlossen und die Ergebnisse dargestellt, werden die DOM-Elemente, die den Inhalt transportieren auf ihre Korrektheit überprüft. Mit selbigem Verfahren kann das Verhalten von DOM-Manipulationen auf JavaScript-Basis getestet werden. Eine Aktion - z.B das Erzeugen eines neuen Tabs in der Suchseite - wird über den Webdriver angestoßen und das vorraussichtliche Verhalten überprüft.

## 7 Einführungsphase

Das Produkt wurde noch nicht eingeführt. Aufgrund der mathematischen Grundlage muss das Produkt zunächst von einer ausgewählten Gruppe der tatsächlichen Endnutzer getestet werden. Hierzu wird als Datenbasis die Datenbank des Live-Systems eingebunden, was im Spring Boot Kontext über die Änderung der zugehörigen Property-Einträge in der Konfigurationsdatei umgesetzt wird. Außerdem sind hier die Basis-URL und einige anderer Attribute anzupassen.

Die Suchfunktion läuft zur Zeit auf einem parallelen Testsystem, das obige Konfiguration nutzt. Die Tester müssen nun auf Grundlage ihrer Erfahrung beurteilen, ob der Suchalgorithmus adäquate Ergebnisse liefert.

Um die Applikation auf dem Testsystem zu installieren ist lediglich notwendig, das Projekt mit dem Tool 'Maven' zu bauen und die entstehende JAR-Datei per FTP auf das Testsystem zu deployen. Anschließend wird die JAR als Systemdienst registriert und erstmalig gestartet.

## 8 Dokumentation

Die neue Suchfunktion wurde in zweierlei Hinsicht dokumentiert. Es wurde eine Entwicklerdokumentation und eine Benutzerdokumentation angefertigt. Die Entwicklerdokumentation wurde in Form eines JavaDoc's in der Projektstruktur bereitgestellt. Das JavaDoc wurde mit Hilfe der IntelliJ IDEA direkt aus der Entwicklungsumgebung generiert und besteht aus den Kommentaren des eigentlichen Programmcodes.

Die Entwicklerdokumentation wurde im firmeneigenen Intranet über die Confluence-Plattform in Form eines Fließtexts mit leitenden Screenshots erstellt. Hier wird dem Nutzer erklärt wie er Suchbegriffe identifiziert, mit vorgeschlagenen Begriffen arbeitet und Suchbegriffe verketteten kann. Ebenfalls wird eine kurze Einführung in die Navigation durch die Suchergebnisse gegeben. Letztlich wird erklärt, wie man von einem Suchergebnis zum zugehörigen Profil navigieren kann.

## 9 Fazit

### 9.1 Soll-/Ist-Vergleich

Im Zuge dieser Projektarbeit wurden alle Anforderungen umgesetzt. Die Aussagekraft des Scoring-Algorithmus muss im Nachhinein durch die Vertriebsabteilung beurteilt werden. Für die Planung sowie die Kontroll- und Testphase wurde zu wenig Zeit eingeplant. Diese Fehlkalkulation konnte durch Zeiterparnis während der Implementierung und die Ausnutzung der eingeplanten Pufferzeit ausgeglichen werden.

Phase	Geplant	Tatsächlich	Differenz
Planung	13 h	16 h	+3 h
Implementierungsphase	33 h	28 h	-5 h
Kontroll- und Testphase	12 h	15 h	+3 h
Erstellen der Dokumentation	10 h	11 h	+1 h
Pufferzeit	2 h	0 h	-2 h
Gesamt	70 h	70 h	

Tabelle 3: Soll-/Ist-Vergleich

## 9.2 Lessons Learned

Vor allem das Spring-Boot-Framework hat beim schnellen Prototyping komplexer Algorithmen geholfen. Dieses Framework ist schnell aufgesetzt und lokale Testumgebungen können leicht erstellt und abgegrenzt werden. Für die Frontendentwicklung hat sich das Framework 'Backbone.js' hinsichtlich der MVC-Umsetzung als nützlich erwiesen. Leider traten Fehler in dem Framework zutage, deren Behebung Recherchen notwendig machten. Die Einarbeitungszeit in 'Backbone.js' ist relativ hoch. Die Projektplanung betreffend wurde die Einteilung der Aufgaben in Sprints als sehr angenehm empfunden. Auch die wöchentliche Abnahme, hat dabei geholfen, Fehler früh zu erkennen und die Anforderung der Vertriebsabteilung besser zu verstehen.

## 9.3 Ausblick

In Abstimmung mit der Vertriebsabteilung muss die Leistung des implementierten Scoringalgorithmus beobachtet und dieser gegebenenfalls verfeinert werden. Denkbar ist, die Suche auf weitere Felder der Mitarbeiterprofile auszudehnen. Auch könnten weitere logische Verknüpfungen der Suchbegriffe, wie 'ODER' oder 'NICHT', eingeführt werden.

## 10 Quellen

- Spring-Boot-Dokumentation (<http://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/>)
- Informationen zur Regressionsanalyse (<http://www.faes.de/Basis/Basis-Statistik/Basis-Statistik-basis-statistik-korrelation-re.html> und [https://de.wikipedia.org/wiki/Lineare\\_Regression](https://de.wikipedia.org/wiki/Lineare_Regression))
- Backbone.js Dokumentation (<http://backbonejs.org/>)
- Informationen zur Projektdokumentation (<https://fachinformatiker-anwendungsentwicklung.net/>)

## Eidesstattliche Erklärung

Ich, Florian Müller, versichere hiermit, dass ich meine **Dokumentation zur betrieblichen Projektarbeit** mit dem Thema

*Unterstützende Suchfunktion – für den ConSol\*-Profil-Generator*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

München, den 03.11.2016

---

FLORIAN MÜLLER

## A Anhang

### A.1 Detaillierte Zeitplanung

<b>Planung</b>	<b>13 h</b>
1. Analyse des Ist-Zustands	3 h
2. Erstellen eines Soll-Konzepts mit der Fachabteilung	3 h
3. Entwurf eines Suchalgorithmus	5 h
4. Entwurf eines Datenbankschemas	2 h
<b>Realisierungsphase</b>	<b>33 h</b>
1. Erstellen einer Testumgebung im Spring-Boot-Kontext	2 h
2. Erstellen der Datenbank	2 h
3. Implementierung der täglichen Datensynchronisierung	6 h
4. Implementierung der Such-Logik	15 h
5. Implementierung des Webfrontends	8 h
<b>Kontroll- und Testphase</b>	<b>12 h</b>
1. Erstellen eines Testplans	1 h
2. Implementierung Modularer Tests für das Backend	4 h
3. Implementierung funktionaler Frontend Tests	5 h
4. Puffer für Nachbesserungen	2 h
<b>Dokumentation</b>	<b>12 h</b>
1. Erstellung der Projektdokumentation	8 h
2. Erstellen der Anwenderdokumentation	2 h
3. Erstellen der Entwicklerdokumentation	2 h
<b>Gesamt</b>	<b>70 h</b>

## A.2 JSON-Modell

---

```
1 {
2   "lastname": "Mustermann",
3   "firstname": "Max",
4   "jobTitle": "Senior Software Developer",
5   "languages": [{
6     "value": "Deutsch",
7     "grade": "C2"
8   }, {
9     "value": "Englisch",
10    "grade": "B1"
11  }],
12  "education": [{
13    "content": "TU-Muenchen",
14    "fromDate": "01/1992",
15    "toDate": "02/1997"
16  }, {
17    "content": "Ausbildung bei ConSol*",
18    "fromDate": "01/1992",
19    "toDate": "02/1997"
20  }],
21  "certificates": [{
22    "dateOfIssue": "01/1992",
23    "validityDate": "02/1997",
24    "organisation": "PMI",
25    "nameAndLevel": "PMP"
26  }],
27  "projects": [{
28    "fromDate": "01/1992",
29    "toDate": "02/1997",
30    "projectTitle": "Audi FahrzeugManager",
31    "company": "Test",
32    "industryField": "Automotive",
33    "projectDescription": "Weiterentwicklung einer J2EE-Applikation zur
34      Verwaltung von Testfahrzeugen",
35    "position": "Software Developer",
36    "projectActivity": "Entwicklung von Java-Code und Testklassen",
37    "toolsEnvironment": ["J2EE, EJB, Spring, Oracle DB"]
38  }],
39 }
```

A Anhang

---

```
39
40     "skills": [{
41         "value": "Java",
42         "grade": "EXPERT"
43     }, {
44         "value": "Java EE",
45         "grade": "EXPERT"
46     }, {
47         "value": "C++",
48         "grade": "PRACTITIONER"
49     }, {
50         "value": "Scala",
51         "grade": "PRACTITIONER"
52     }, {
53         "value": "Cobol",
54         "grade": "PRACTITIONER"
55     }, {
56         "value": "Anforderungsmanagement",
57         "grade": "PRACTITIONER"
58     }, {
59         "value": "Ruby",
60         "grade": "PRACTITIONER"
61     }, {
62         "value": "Erstellung von Projektstrukturplaenen",
63         "grade": "PRACTITIONER"
64     }],
65     "id": 15
66 }
```

---



### A.3 Screenshots des Profil-Generators

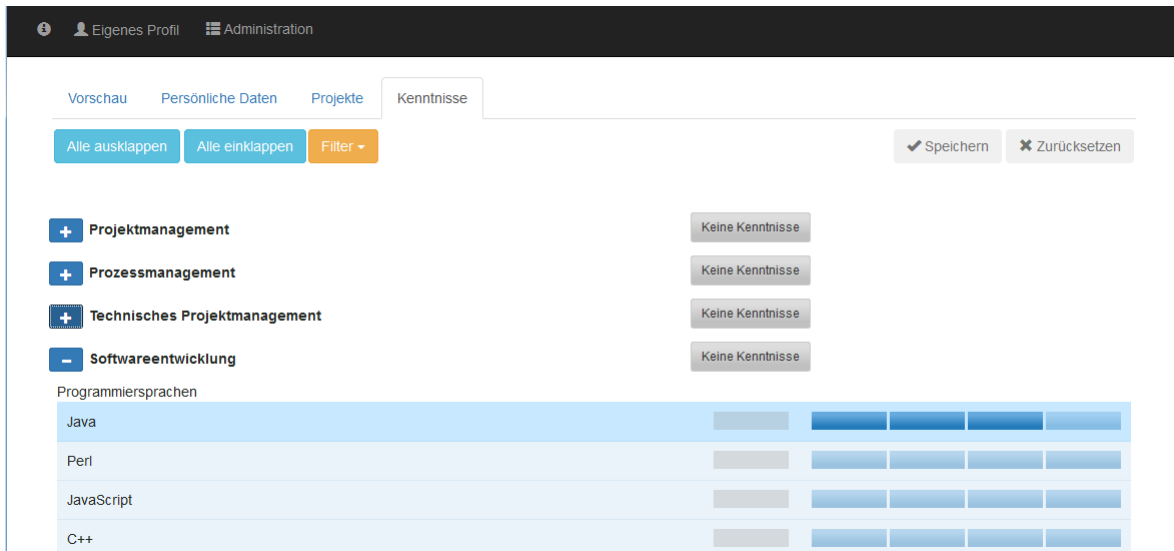


Abbildung 3: Eingabemaske für Fähigkeiten

### A.4 Proof-of-concept der Korrelationen

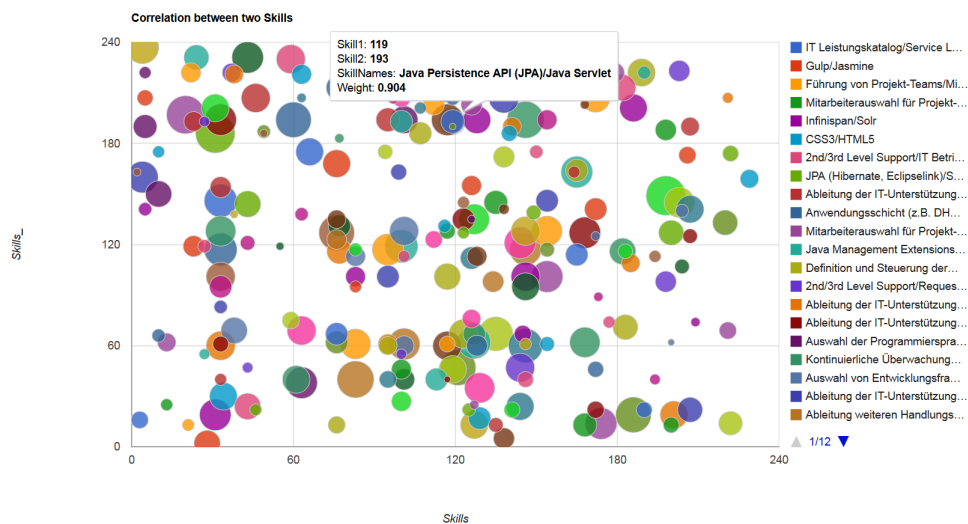


Abbildung 4: Darstellung der Korrelationen zwischen den Fähigkeiten im Live-System mit einer Stärke von  $\pm 0,85$  oder mehr

## A.5 Klassendiagramm

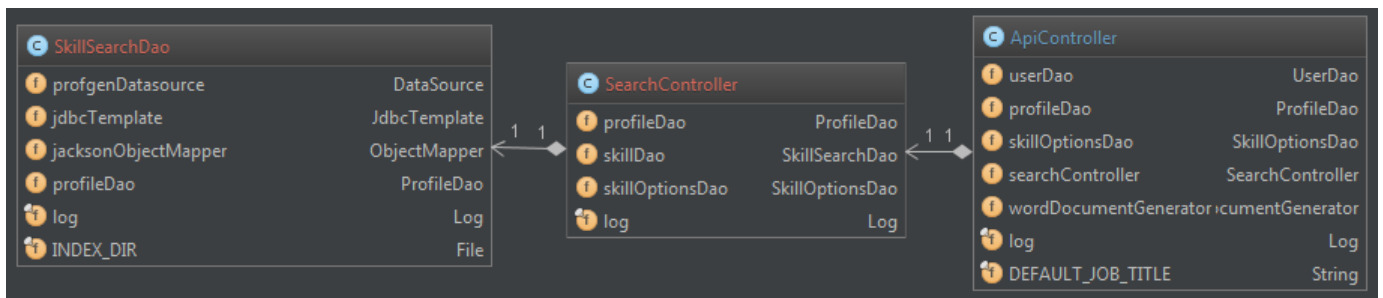


Abbildung 5: Klassendiagramm mit Beziehungen der zu implementierenden Klassen

## A.6 Datenbankmodell

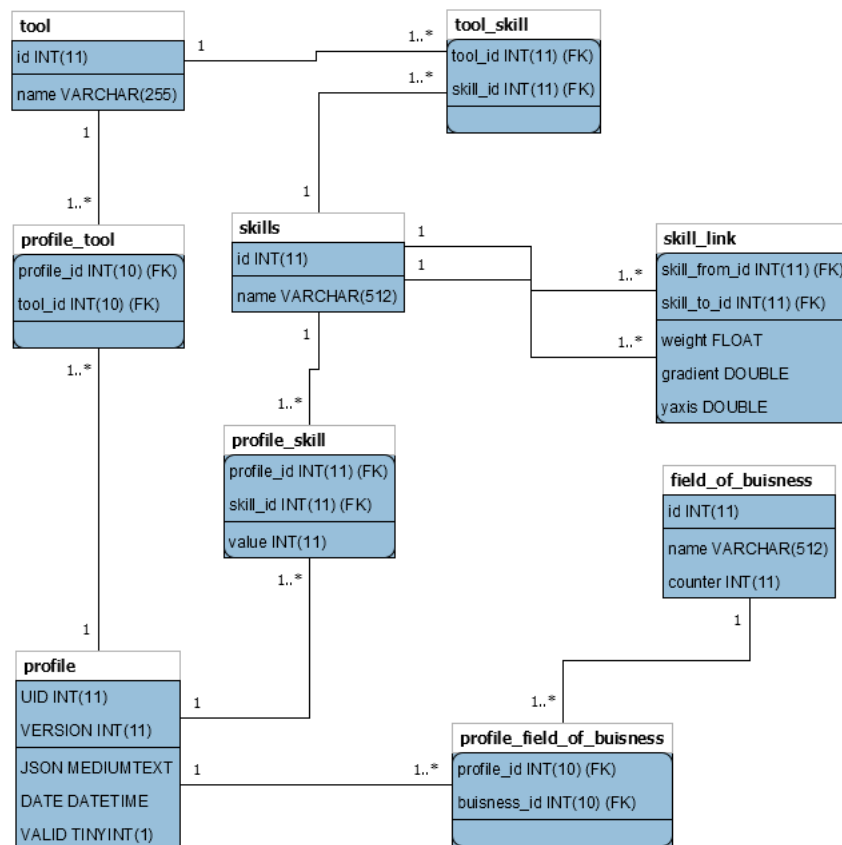
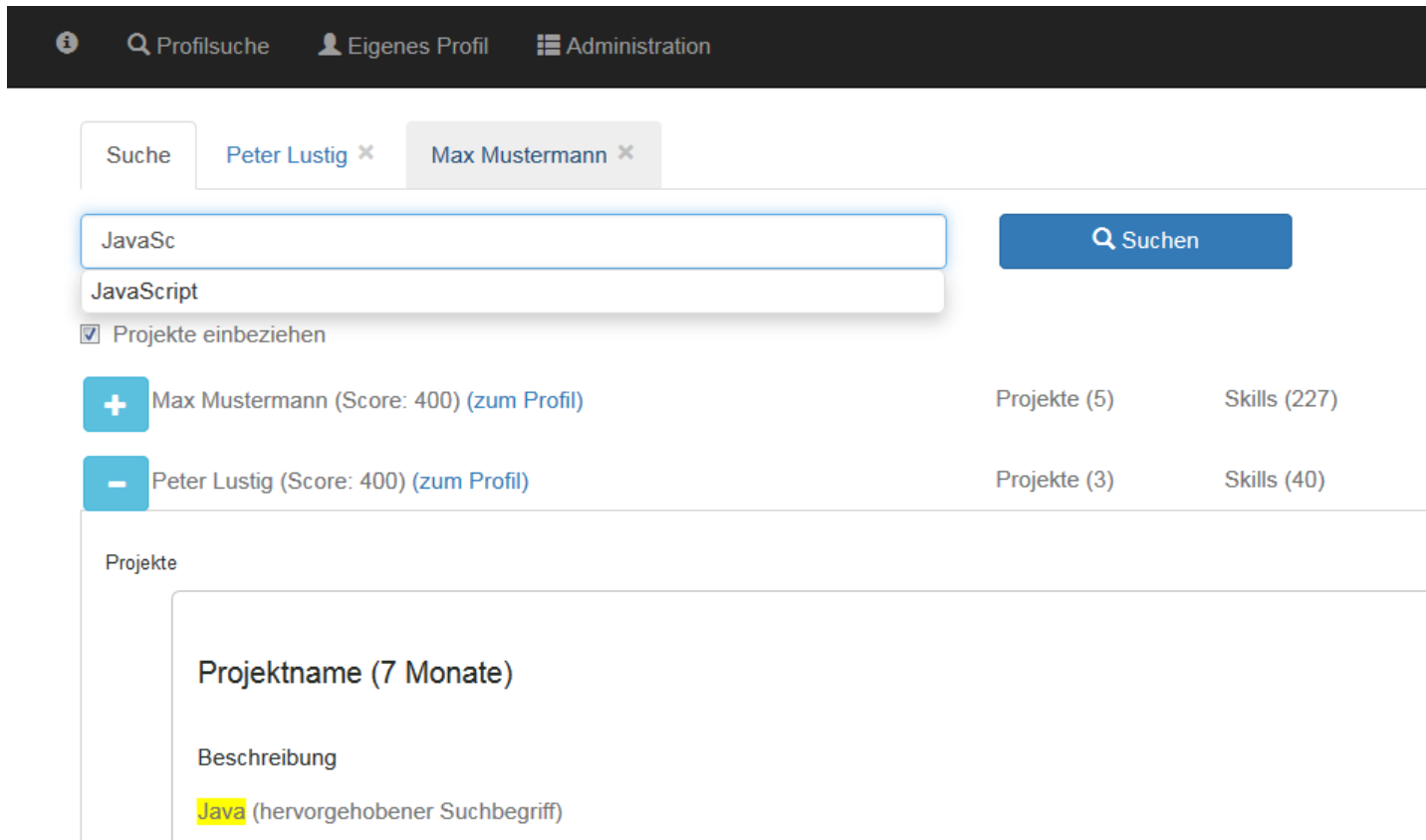





Abbildung 6: Datenbankmodell

## A.7 Oberflächenentwurf



Navigation:  Profilsuche  Eigenes Profil  Administration

Suche: Peter Lustig ✕ Max Mustermann ✕

JavaScript

☒ Projekte einbeziehen

**+** Max Mustermann (Score: 400) (zum Profil) Projekte (5) Skills (227)

**-** Peter Lustig (Score: 400) (zum Profil) Projekte (3) Skills (40)

Projekte

Projektname (7 Monate)

Beschreibung

Java (hervorgehobener Suchbegriff)

Abbildung 7: Entwurf der Suchmaske mit statischem HTML und Bootstrap

## A.8 Interessante Funktionen

```
1 private void initSkillLinks () {
2     log.info("Starting computation of DbEntity links...");
3     Date d = new Date();
4     Set<DbEntity> dbEntities = skillDao.getSkillsFromDB();
5     List<Profile> profiles = profileDao.getAllActiveProfiles();
6     for(DbEntity cur_from : dbEntities){
7         for(DbEntity cur_to : dbEntities){
8             if(cur_from.equals(cur_to)){ continue; }
9             List<Integer> s1List = new ArrayList<>();
10            List<Integer> s2List = new ArrayList<>();
11            for( Profile p : profiles ){
12                int firstSkillGrade = -2;
13                int secondSkillGrade = -2;
14                if(p.getSkills() == null){ continue; }
15
16                for( ProfileSkill ps : p.getSkills() ){
17                    if( firstSkillGrade == -2 && ps.getValue().equals(cur_from.getName()) ){
18                        firstSkillGrade = getGradeAsIntFromSkill(ps);
19                    }
20                    if( secondSkillGrade == -2 && ps.getValue().equals(cur_to.getName()) ){
21                        secondSkillGrade = getGradeAsIntFromSkill(ps);
22                    }
23
24                    if( firstSkillGrade >= -1 && secondSkillGrade >= -1 ){
25                        break;
26                    }
27                }
28
29                if( firstSkillGrade >= -1 && secondSkillGrade >= -1 ){
30                    s1List.add(firstSkillGrade);
31                    s2List.add(secondSkillGrade);
32                }
33            }
34        }
35        if(s1List.isEmpty()){ continue; }
36
37        Correlation correlation = new Correlation(s1List, s2List);
38        createLinkBetweenTwoSkills(cur_from, cur_to, correlation.getR(), correlation.getB(), correlation.getA());
39    }
40 }
41 }
42 Date b = new Date();
43 log.info("Successfully computed skill links in : " + ((b.getTime() - d.getTime())/1000) + " seconds.");
44 }
```

A Anhang

```
1 private List<Profile> getProjectScore(List<Profile> profiles, String[] searchStrings){
2     for(String searchString : searchStrings){
3         DbEntity dbEntity = skillDao.getSkillByName(searchString);
4         if(dbEntity == null){continue;}
5         for(int i = 0; i < profiles.size(); i++){
6             Profile profile = profiles.get(i);
7             int profileSkillValue = getSkillValueFromProfile(profile, dbEntity);
8             for(Project project : profile.getProjects()){
9                 double predictedSkillValue = 0.0d;
10                for(String tool : project.getToolsEnvironment().get(0).split(", ")){
11                    tool = tool.trim();
12                    DbEntity matchedDbEntity = skillDao.getMappedSkillFromToolName(tool);
13                    if(matchedDbEntity == null){continue;}
14
15                    if(dbEntity.equals(matchedDbEntity)){ // if the very same skill is found in a project add
16                        // the value from the profileSkill
17                        predictedSkillValue += profileSkillValue;
18                        continue;
19                    }
20                    Correlation correlation = skillDao.getCorrelationBetweenTwoSkills(matchedDbEntity,
21                        dbEntity);
22                    if(correlation == null){continue;}
23                    predictedSkillValue += correlation.getA() + getSkillValueFromProfile(profile,
24                        matchedDbEntity) * correlation.getB();
25                }
26                if(predictedSkillValue != 0) {
27                    predictedSkillValue = predictedSkillValue * getWeightFromProjectDuration(project); //
28                    // compute the weight of the project duration
29                    profile.setSearchValue((float) (profile.getSearchValue() + predictedSkillValue));
30                }
31            }
32        }
33    }
34    return profiles.parallelStream().sorted((p1, p2) -> Float.compare(p2.getSearchValue(), p1.getSearchValue()))
35        .collect(Collectors.toList());
36}
```