Mitarbeiter Skills - Applikation zur Erfassung von Mitarbeiterfähigkeiten

Projektarbeit Nr.: 2 vorgelegt am: 01.10.2020

von: [redacted]

Matrikelnummer: [redacted]

Berufsakademie: [redacted]

Studienbereich: Technik

Studiengang: Praktische Informatik

Kurs: [redacted]

Ausbildungsstätte: dotSource GmbH
Goethestraße 1
07743 Jena

Betreuer Praxisbetrieb: [redacted]

Gutachter Berufsakademie: [redacted]

Sperrvermerk

Die vorliegende Arbeit beinhaltet interne und vertrauliche Informationen der Firma dotSource GmbH. Die Weitergabe des Inhaltes der Arbeit und eventuell beiliegender Zeichnungen und Daten im Gesamten oder in Teilen ist grundsätzlich untersagt. Es dürfen keinerlei Kopien oder Abschriften - auch in digitaler Form - gefertigt werden. Ausnahmen bedürfen der schriftlichen Genehmigung der Firma dotSource GmbH

I Inhaltsverzeichnis

II Tabellenverzeichnis

III Abbildungsverzeichnis

IV Abkürzungsverzeichnis

API	Application-Programming-Interface
BSON	Binary JSON
DBMS	Datenbankverwaltungssystem
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
SQL	Structured Query Language
URL	Uniform Resource Locator

1 Einleitung

Für Unternehmen stellt die Erfassung und Aktualisierung von Mitarbeiterinformationen eine Herausforderung dar. Gespeichert werden unter anderem Name, Emailadresse und das Geburtsdatum der Mitarbeiter. Auch deren Zertifikate oder an welchen Projekten sie arbeiten soll abrufbar sein. Mittels einer Lösung durch Übersichtsseiten, Word-Dokumenten oder Excel-Tabellen werden weitere Probleme erzeugt. Aufgelistete Daten veralten und müssen von Mitarbeitern selbst gepflegt werden. Auch ein effizientes Durchsuchen ist nicht möglich und die Maschinenlesbarkeit der Informationen ist nicht gewährleistet.

Einige Unternehmen entwickeln Anwendungen, um entsprechende Daten zu persistieren. Deren Nutzung ist in verschiedenen Fällen hilfreich. Beispielsweise können folgende Anwendungsfälle mithilfe einer Applikation in kurzer Zeit gelöst werden:

- „Welcher Mitarbeiter hat Erfahrung mit einer bestimmten Technologie?“
- „An wie vielen Projekten war der Mitarbeiter beteiligt?“
- „Welcher Entwickler hat ein SAP-Zertifikat?“
- „Welcher Entwickler beherrscht die Programmiersprachen Java und PHP?“

Zusätzlich zu diesen Daten könnten Projekte der Firma oder persönliche Informationen der Mitarbeiter abgefragt werden. Hierfür werden Daten durch Angestellte eingepflegt und können bei Notwendigkeit aktualisiert werden.

Nachfolgend werden die Konzeption und Umsetzung einer Applikation, welche Informationen zu Mitarbeitern eines Unternehmens erfasst, beschrieben. Es wird die Datenhaltung modelliert und die Bereitstellung der Daten über eine

Benutzerschnittstelle realisiert. Der Fokus der Arbeit liegt auf der Modellierung der Schnittstelle und Datenbank.

Die Erläuterung relevanter Grundlagen steht zu Beginn im Mittelpunkt. Dabei werden für das Verständnis der Arbeit notwendige Begriffe vorgestellt. Der Hauptteil beschreibt die Konzeption und Umsetzung der Applikation. Dafür wird die Entwicklung des Lösungsweges dargestellt und die Implementierung des Konzeptes erklärt. Das abschließende Fazit vergleicht die entstandene Lösung mit den gestellten Anforderungen und wirft einen kritischen Blick auf mögliche Erweiterungen sowie potenzielle Verbesserungen der Implementierung.

2 Grundlagen

In diesem Kapitel werden grundlegende Begriffe der Datenbankmodellierung, welche in der Arbeit Anwendung finden, thematisiert. Es werden Kardinalitäten und gängige Datenbankmodelle vorgestellt. Außerdem erfolgt eine Einführung in die Verwendung von HTTP-Methoden. Sie stehen bei der Erstellung und Verwendung einer Benutzerschnittstelle im Mittelpunkt.

2.1 Kardinalitäten

Mithilfe von Kardinalitäten werden Beziehungen zwischen zwei Entitäten beschrieben. Für eine Entität wird angegeben mit wie vielen Entitäten eines anderen Typs sie zusammenhängt[1]. Die wechselseitige Beziehung wird somit durch zwei Mengenangaben vollständig charakterisiert. Dabei spielt lediglich eine Unterscheidung zwischen höchstens einer Entität oder mehreren Entitäten eine Rolle. Es ergeben sich folgende drei Beziehungstypen:

- 1:1-Relation
- 1:N-Relation
- N:M-Relation

Abbildung 1 zeigt die eineindeutige Beziehung zwischen der Entität Mitarbeiter und der ihm zugeordneten Mitarbeiteridentifikation. Beiden Entitäten kann genau ein Objekt des jeweils anderen Typs zugeordnet sein.

Abbildung 1 - 1:1-Relation

Abbildung 2 zeigt ein Beispiel einer 1:N-Relation. Jeder Mitarbeiter arbeitet in genau einer Abteilung. Aus entgegengesetzter Leserichtung ergibt sich, dass jeder Abteilung mehrere Mitarbeiter zugeordnet sein können.

[1] vgl. [Tin08].

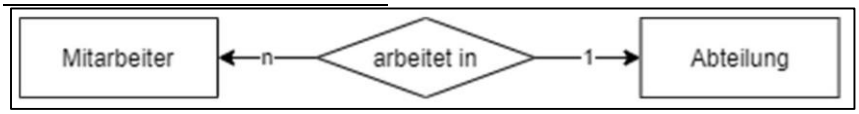


Abbildung 2 - 1:N-Relation

Eine N:M-Relation wird in der Abbildung 3 visualisiert. Dargestellt ist, dass an einem Projekt mehrere Mitarbeiter tätig sein können. Umgekehrt ist für jeden Mitarbeiter auch eine Beschäftigung an mehreren Projekten möglich.

Abbildung 3 - N:M-Relation

2.2 Datenbankmodelle

Eine Datenbank ist ein System, das der Verwaltung von elektronischen Informationen dient. Von zentraler Bedeutung für die Speicherung ist das
Datenbankverwaltungssystem (DBMS), dessen Aufgabe die Steuerung ist[2]. Grundlage einer effizienten Datenhaltung ist die Wahl eines geeigneten Datenmodells, das durch den Hersteller des DBMS selektiert wird. Es gibt die beiden gängigen Modelle relationale und dokumentorientierte Datenbank.

Das relationale Modell basiert darauf, Daten in Tabellen darzustellen. Dabei entspricht jede Zeile einem Datensatz. Diesem ist eine Identifikation eindeutig zugeordnet. Die Spalten der Tabelle legen fest welche Attribute gespeichert werden. Somit können in den Zellen die Werte entsprechend der Attribute eines Datensatzes gespeichert werden[3]. Viele relationale Datenbanken nutzten Structured Query Language (SQL) zur Kommunikation. Diese ermöglicht schnelles Einfügen, Verändern oder Löschen von Daten.

Um semistrukturierte Daten zu persistieren kann das dokumentorientierte Modell angewandt werden. Im Gegensatz zur relationalen Datenbank werden Dokumente mit eindeutiger Identifikation genutzt. Somit ist kein festes Schema in zu speichernden

[2] vgl. [Ora20b].
[3] vgl. [Ora20a].

Entitäten erforderlich. Daten werden beispielsweise im JSON[4]-Format gespeichert[5]. Durch die Verwendung von Schlüssel/Werte-Paaren sind gewünschte Information abrufbar.

2.3 HTTP-Methoden

Das Hypertext Transfer Protocol (HTTP) dient der Kommunikation im World Wide Web. Oft tauschen Webbrowser und Webserver über dieses Protokoll Informationen aus. Entscheidend für die Übertragung sind die sogenannten HTTP-Methoden, welche die Art der Anfrage festlegen[5]. Einige von ihnen werden für die Implementierung der Mitarbeiterapplikation zum Einsatz kommen. Zum Beispiel die sogenannten GETMethoden. Sie werden verwendet, um eine Ressource anzufordern. Mithilfe von optionalen Parametern kann die Anfrage spezifiziert werden. Sie werden im Pfad des Uniform Resource Locator (URL) angegeben.

Um Daten an einen Server zu senden wird die Methode POST genutzt. Der Inhalt wird der Anfrage im Request Body hinzugefügt. Mit gesendeten Informationen kann durch den Server zum Beispiel eine Datenbank erweitert werden[6].

Bei einer PUT-Methode werden ebenfalls Daten an den Server gesendet. Wie bei einem POST wird eine Ressource an die Anfrage angehängt. An benannter Stelle, die der URL definiert, wird der Inhalt hochgeladen. Ist bereits eine Datei vorhanden, wird diese überschrieben[8]. Dem Entfernen einer Entität dient die DELETE-Methode. Durch den URL wird identifiziert, welche Ressource gelöscht wird.

---

[4] JavaScript Object Notation [5]
vgl. [ION20].
[5] vgl. [Tut20].
[6] vgl. [SEL20].
[8] vgl. [Hol20].

3 Konzeption

3.1 Entity-Relationship-Modell

Ein Entity-Relationship-Modell dient der Visualisierung eines gegebenen Sachverhaltes in der konzeptionellen Phase. Häufig wird es verwendet, um einen Datenbankentwurf zu erstellen. So wird dargestellt welcher Zusammenhang zwischen den verwendeten Entitäten besteht[7]. Das Modell zeigt Entitäten und deren Attribute auf. Zudem werden Beziehungen mit entsprechenden Kardinalitäten gekennzeichnet. Einige Attribute sind abhängig von zwei Objekten. Somit können sie keiner Entität eindeutig zugeordnet werden. Dieser Sachverhalt wird mit sogenannten RelationsAttributen abgebildet. Das bedeutet, dass die Beziehung zwischen zwei Entitäten weitere Informationen enthält.

Einzelne Objekte werden nach einem vorgegebenen Schema visualisiert. Die Folgende Tabelle 1 zeigt durch welche geometrischen Figuren welche Art von Objekten ausgedrückt wird. Diese werden mit Hilfe von Linien verbunden.

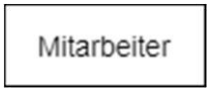
Art des Objektes	Geometrische Figur	Beispiel
Entität	Rechteck	
Attribut	Oval	
Relation	Raute	

Tabelle 1 - Visualisierungsschema Entity-Relationship-Modell

Für den gegebenen Kontext der Mitarbeiter Applikation wurde ein Entity-RelationshipModell entwickelt, welches in Abbildung 4 zu sehen ist.

[7] vgl. [Beg20].

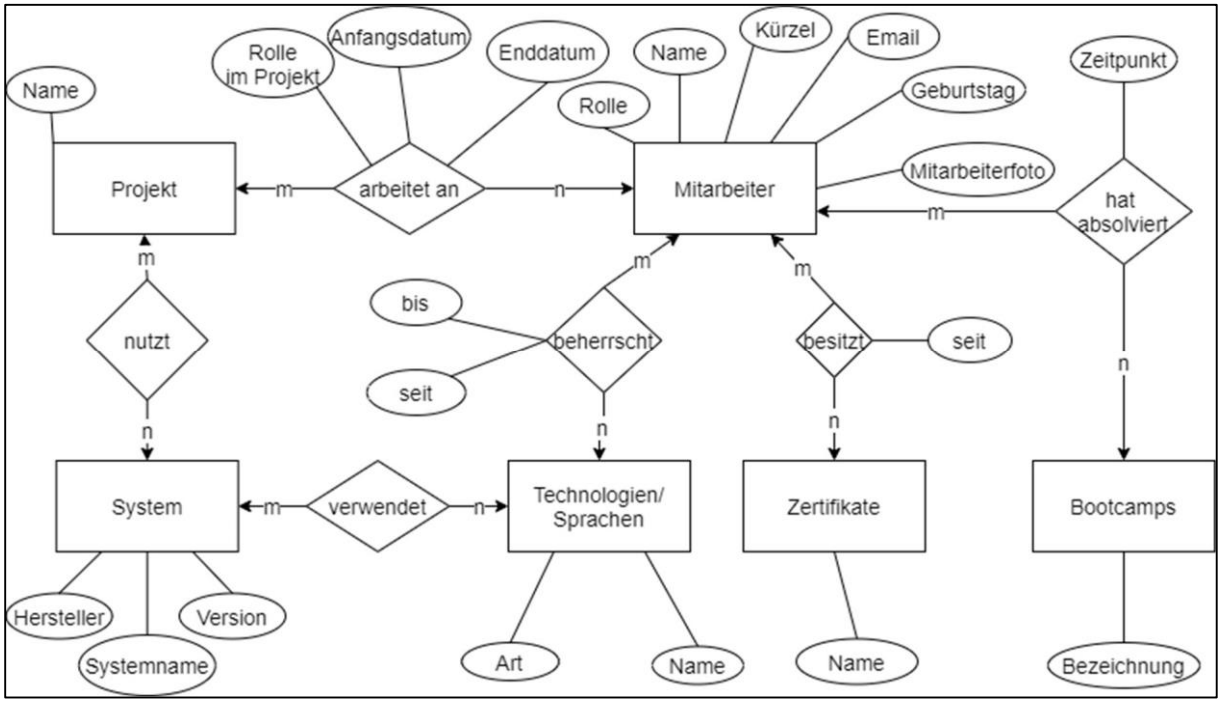


Abbildung 4 - Entity-Relationship-Modell

Von zentraler Bedeutung ist das Objekt Mitarbeiter. Dessen Attribute sind unter anderem Name, Kürzel und E-Mail-Adresse. Es sind vier Relationen zu anderen Entitäten dargestellt. Diese besitzen die Kardinalität N zu M. Beispielsweise besteht eine solche Verbindung zu der Entität Projekt. Das bedeutet, dass ein Mitarbeiter in mehreren Projekten tätig sein kann. Umgekehrt kann ein Projekt durch mehrere Personen bearbeitet werden. Die Relation besitzt drei weitere Attribute. Sie beinhalten Anfangs- und Enddatum der Mitarbeit, sowie die Rolle, die der Mitarbeiter im Projekt hat.

3.2 Systemüberblick

Abbildung 5 visualisiert den grundlegenden Aufbau der Applikation zur Erfassung von Mitarbeiterinformationen. Um Daten zu erhalten, wird ein sogenannter Controller angesprochen. Er ist der steuernde Teil einer Geschäftslogik und nimmt HTTP-Anfragen entgegen[8]. Der Client hat so die Möglichkeit, einen gewünschten

[8] vgl. [Mar20].

Application-Programming-Interface(API)-Endpunkt anzufragen. Die Implementierung verwendet Spring [9] . Das Framework dient der Unterstützung von Java-basierten Unternehmensanwendungen. Mittels des Moduls Spring Data[10] kommuniziert der Controller mit der Datenbank. So hat dieser die Möglichkeit, Informationen aus der Datenbank abzufragen und dem Client zu übergeben. Auch das Speichern von Daten ist möglich, wenn deren Übermittlung durch die HTTP-Anfrage erfolgt.

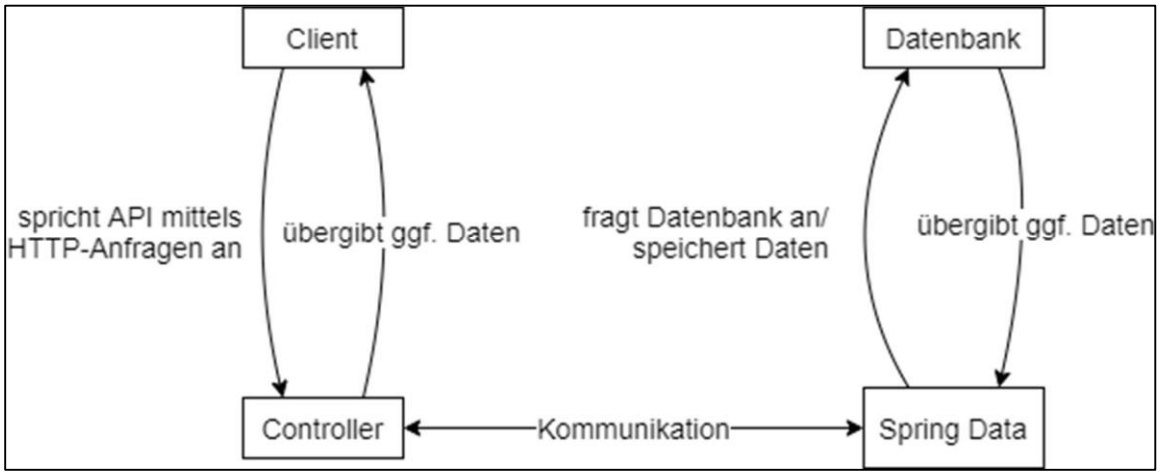


Abbildung 5 - Systemüberblick

3.3 Nutzung der Datenbank MongoDB

Für die Persistierung wurde die dokumentorientierte Datenbank MongoDB [11] gewählt. Informationen werden im BSON[12]-Format gespeichert. Dabei wird auf eine feste Struktur, die beim relationalen Modell besteht, verzichtet[15]. Im Weiteren stehen entscheidende Vorteile des Systems im Mittelpunkt.

Mit MongoDB ist eine Änderung an der Struktur eines Objektes ohne großen Aufwand möglich. Beispielsweise kann der Entität Mitarbeiter ein weiteres Attribut hinzugefügt werden. Bereits erfasste Mitarbeiter müssen nicht verändert werden. Diese Flexibilität ist ein Vorteil für die Anwendung, da Menge und exakter Aufbau der Informationen nicht vorhergesagt werden

[9] https://spring.io/
[10] https://spring.io/projects/spring-data
[11] https://www.mongodb.com/
[12] Binary JSON, Orientierung am Format JSON
[15] vgl. [Sas19].

können[13]. Ein weiterer Vorteil ist die Verwendbarkeit des Datentyps JSON. Das Erstellen und Auslesen von Paaren, bestehend aus Schlüssel und Wert, erfordert keine komplexen Implementierungen. Zudem ist das Format unabhängig von der Programmiersprache und von verschiedenen Systemen verwendbar.

[13] vgl. [Ion20].

4 Umsetzung

Gegenstand dieses Kapitels ist die Umsetzung der entwickelten Applikation zur Erfassung von Mitarbeiterinformationen. Die Implementierung kann in drei Schritte eingeteilt werden. Zuerst wird die Realisierung der Benutzerschnittstelle und deren zentrale Funktionen vorgestellt. Danach steht der Aufbau der Datenbank im Mittelpunkt. Zuletzt wird erläutert, auf welche Weise eine Clientimplementierung die Benutzerschnittstelle nutzen kann.

4.1 Realisierung der API

Um die Erstellung der API zu vereinfachen, wird die Software Swagger[14] verwendet.

Damit wird eine Dokumentation angefertigt, die notwendige Entitäten und HTTPAnfragen definiert. Daraus wird das Gerüst der Schnittstelle, bestehend aus APIEndpunkten und Entitäten, automatisch generiert.

4.1.1 Umsetzung der N:M-Relationen

Für die Umsetzung der N:M-Relationen ist es wichtig, dass sowohl Informationen über die beteiligten Entitäten als auch Daten über die Beziehung abrufbar sind. Für die Implementierung wurden zwei Möglichkeiten betrachtet.

Mit der ersten Variante, die Abbildung 6 visualisiert, werden drei Collections[15] umgesetzt. Jeweils eine nimmt Daten der Entitäten auf. In der dritten Collection werden Informationen bezüglich der Relation gespeichert. Dazu gehören Relationsattribute und die IDs der Entitäten, welche zueinander in Beziehung stehen.

Der Vorteil bei dieser Variante ist, dass Informationen lediglich einmal gespeichert werden. Ein Nachteil besteht dagegen in der vergleichsweise aufwendigen Abfrage von Daten. Der Grund dafür ist, dass der Client bei der Abfrage einer Entität auch

Informationen von dessen Beziehungen erhalten soll. Somit müssen bei einer GETAnfrage Daten von verschiedenen Quellen abgefragt werden.

[14] https://swagger.io/

[15] Sammlung von Dokumenten, eine oder mehrere bei einer MongoDB-Datenbank

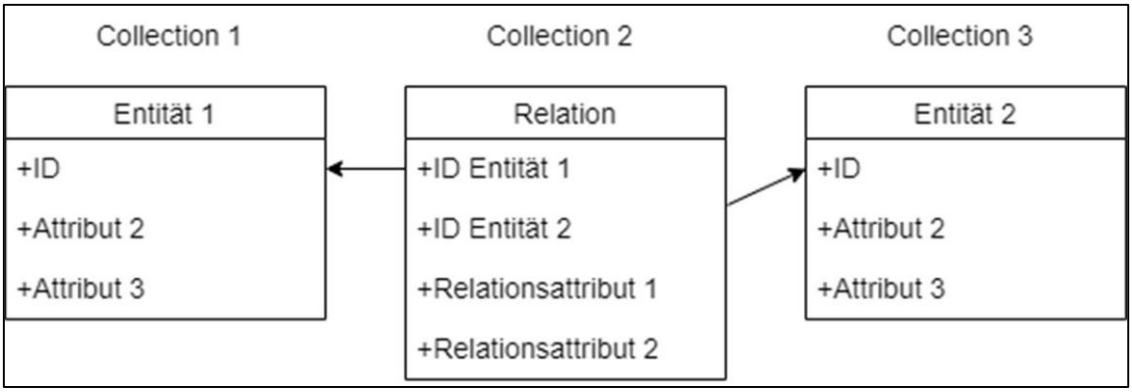


Abbildung 6 - Umsetzung N:M-Relation mit drei Collections

Bei Anwendung der zweiten Variante werden zwei Collections implementiert. Diese sind jeweils einer Entität zugeordnet. Zusätzlich beinhalten gespeicherte Objekte Informationen über deren Beziehungen. Es werden Links zur Abfrage und IDs der anderen Instanz sowie Relationsattribute persistiert. Daraus ergibt sich der Vorteil, dass bei einer Abfrage lediglich eine Collection angefragt wird und eine GET-Anfrage ohne großen Aufwand bearbeitet werden kann. Im Gegensatz dazu ist das Aufnehmen von Informationen in die Datenbank aufwendig. Hierfür müssen alle Daten konsistent gehalten werden. Beispielsweise müssen nach Anlegen eines Mitarbeiters dessen Beziehungen zu anderen Objekten überprüft werden. Instanzen, zu denen eine Relation besteht, müssen ebenfalls entsprechende Daten speichern. Infolgedessen entsteht ein weiterer Nachteil, da diese Daten doppelt gespeichert werden.
Es wird davon ausgegangen, dass ein Großteil der Daten einmalig angelegt wird. Im Gegensatz dazu soll die Abfrage von Informationen, welche mehrfach geschehen kann, performant implementiert werden. Aufgrund dessen, dass bei der Applikation die Abfrage der Informationen im Mittelpunkt steht, wird die zweite Variante gewählt. Mit dieser wird die Anwendung in Bezug auf GET-Anfragen optimiert.

4.1.2 Suchfunktion über alle Entitäten

Die Anwendung soll eine Suche über alle Entitäten ermöglichen. Hierfür soll ein Suchterm mit signifikanten Feldern aller Objekte abgeglichen werden. Gemäß Tabelle 2 ist festgelegt welche Attribute der Entitäten beachtet werden sollen.

Mitarbeiter	Projekt	System	Technologie	Bootcamp	Zertifikat
Vorname Nachname Kürzel	Name	Name Hersteller	Name	Name	Name

Tabelle 2 - Relevante Attribute bei globaler Suche

Die globale Suchfunktion wird von dem in der Abbildung 7 dargestellten API-Endpunkt realisiert. Mithilfe der Annotation „@RequestMapping“ wird die Anfrage als GETMethode und der Pfad mit „/search/{searchTerm}“ definiert. Weiterhin sind zwei Parameter angegeben. „searchTerm“ wird als Variable im Pfad gesetzt. Sie entspricht dem Suchbegriff, der mit den relevanten Attributen verglichen wird. Mittels des Parameters „ignored“ wird eine Liste von Entitäten angegeben, die nicht durchsucht werden sollen. Die Variable kann optional in dem sogenannten Query-String[16] übergeben werden.

```
@RequestMapping(value = "/search/{searchTerm}",
                method = RequestMethod.GET)
public ResponseEntity<Object> search(
                        @ApiParam(value = "Searchterm which is compared with important
                                Attributes of all objects", required = true)
                @PathVariable("searchTerm")             String
searchTerm,
                        @ApiParam(value = "Enities that can be ignored during search",
                                allowableValues = "Employee, System, Project, Bootcamp,
                                    Certificate, Technology")
                                @Valid @RequestParam(value = "ignored", required = false)
                            List<String> ignored);
```

Abbildung 7 - API-Endpunkt "search"

Die Logik der Implementierung, welche Abbildung 8 zeigt, entspricht dem folgenden Schema, das für alle Entitäten wiederholt wird: Zunächst findet eine Überprüfung statt, welche feststellt ob die zu betrachtende Klasse in der Variablen „ignored“ vorhanden ist. Ist das der Fall wird mit der nächsten Entität fortgefahren. Ansonsten werden nacheinander alle relevanten Attribute des Objekttyps betrachtet. Hierfür werden Anfragen an die Datenbank gestellt. Es werden alle Instanzen zurückgegeben, deren

Wert des durchsuchten Attributs dem Suchterm entspricht. Die Ergebnisse der

[16] Teil des URL; enthält benannte Parameter als Daten

Datenbankanfrage werden einer Liste „result“ hinzugefügt. Nach der Betrachtung aller Entitäten liefert der API-Endpunkt eine HTTP-Response, bestehend aus den Suchergebnissen und dem HTTP-Status „200“, an den Client. Abbildung 9 zeigt die Response einer Anfrage mit dem Suchterm „ProjektTest“.

```
List<Object> result = new ArrayList<>();
ignored = Optional.ofNullable(ignored).orElse(new ArrayList<>()); if
(!ignored.contains("Employee")) {              result.addAll(employeeRepository.
                    findByAbbreviationOrFirstNameOrLastName(searchTerm,                    searchTerm,
searchTerm));
}
if (!ignored.contains("Project")) {  result.add(projectRepository.findByName(searchTerm));
} if (!ignored.contains("System")) {
          result.addAll(systemRepository.
                                findByNameOrVendor(searchTerm, searchTerm));
}
if (!ignored.contains("Technology")) {       result.add(technologyRepository.findByName(searchTerm));
}
if (!ignored.contains("Bootcamp")) {         result.add(bootcampRepository.findByName(searchTerm));
}
if (!ignored.contains("Certificate")) {      result.add(certificateRepository.findByName(searchTerm));
}
result.removeAll(Collections.singletonList(null)); return new
ResponseEntity<Object>(result, HttpStatus.OK);
```

Abbildung 8 - Implementierung globale Suche

```
HTTP/1.1 200
X-Application-Context: application:8080
Content-Type: application/json;charset=UTF-8
Transfer-Encoding: chunked Date: Wed, 23 Sep
2020 12:27:42 GMT

[
  {
    "name": "ProjektTest",
    "employeeLinks": [
      {
        "role": "Developer",
        "from": "01.08.2019",
        "to": "23.09.2020",
        "link": {
          "datatype": "Employee",
          "link": "http://localhost:8080/employee/findByAbbreviation/test",
          "name": "test"
        }
      }
    ],
    "systemLinks": []
  }
]
```

Abbildung 9 - Response einer Anfrage an den Suchendpunkt

4.1.3 Realisierung einer GET-Anfrage

Abbildung 10 zeigt eine Implementierung der Abfrage von Mitarbeitern. Der APIEndpunkt nimmt einen Parameter entgegen. Anhand von diesem werden Objekte in der Datenbank gesucht. Im Beispiel entspricht dieser Parameter dem Kürzel. Ist ein Mitarbeiter mit dem Wert des Attributes in der Datenbank gespeichert, werden dessen Informationen an den Client zurückgegeben.

```
@RequestMapping(value = "/employee/findByAbbreviation/{abbreviation}",   method = RequestMethod.GET)
public ResponseEntity<Employee> findEmployeeByAbbreviation(
                                @ApiParam(value = "Abbreviation to filter by", required = true)
                                        @PathVariable("abbreviation") String abbreviation)
{
        Employee result = employeeRepository.                findByAbbreviation(abbreviation);
        return new ResponseEntity<Employee>(result, HttpStatus.OK); }
```

Abbildung 10 - Beispiel Abfrage Mitarbeiter

4.1.4 Realisierung einer POST-Anfrage

Beim Anlegen eines Objektes in der Datenbank muss darauf geachtet werden, dass alle Beziehungen konsistent gespeichert werden. Hierfür muss jede angegebene Relation der Instanz überprüft werden. Das Schema, nach dem API-Endpunkte mit der Methode POST implementiert werden, visualisiert Abbildung 11. Nachfolgend wird die Logik am Beispiel der Entität Mitarbeiter erläutert.

Dem Objekttyp Mitarbeiter sind vier N:M-Relationen zugeordnet worden. Nacheinander werden die Beziehungen überprüft. Für den Fall, dass das Objekt zu einer Entität keine Relation besitzt, wird mit den Relationen der nächsten Entität fortgefahren. Ansonsten wird festgestellt, ob die Instanzen, auf die der Mitarbeiter verweist, in der entsprechenden Collection existieren. Ein Mitarbeiter kann beispielsweise einem Projekt, welches noch nicht in die Datenbank aufgenommen wurde, zugeordnet sein. In diesem Fall wird das Projekt neu erstellt und gespeichert. Hierfür werden ID und Beziehungsinformationen des Mitarbeiters verwendet. So wird eine konsistente Relation hergestellt.

Für den Fall, dass das verwiesene Projekt bereits erstellt wurde, wird geprüft, ob es eine Beziehung zum Mitarbeiter besitzt. Fehlt diese Verlinkung, wird sie ergänzt. Anschließend wird mit dem nächsten Projekt, auf das der Mitarbeiter verweist, fortgefahren. Diese Logik wird für die Beziehungen zu den Entitäten Bootcamp, Zertifikat und Technologie wiederholt.

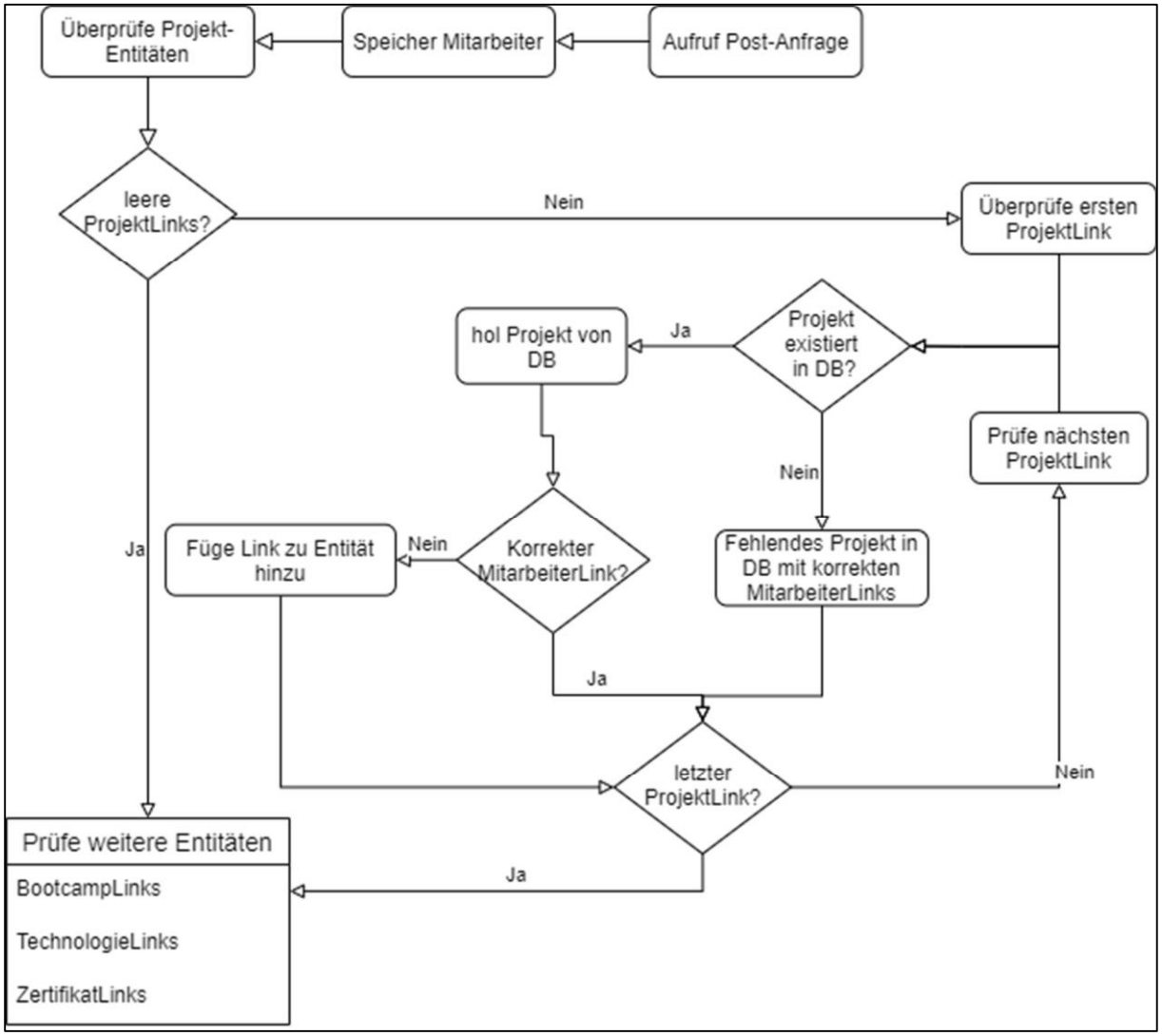


Abbildung 11 - Schema POST-Anfrage

4.2 Erstellung der Datenbank mittels MongoDB

Das Projekt ist als Maven[17] Projekt aufgesetzt. In der zugehörigen Datei „pom.xml“ ist die Abhängigkeit zu Spring Data definiert. Den entsprechenden Ausschnitt zeigt Abbildung 12. Das Modul des Frameworks Spring stellt Funktionalitäten für die Datenbankanbindung bereit. Voraussetzung dafür ist die Installation von MongoDB.

[17] https://maven.apache.org/

```
<dependency>
                    <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
```

Abbildung 12 - Abhängigkeit zu Spring Data in der Datei "pom.xml"

4.2.1 Definition einer Entität

Mit MongoDB als Datenbank werden Objekte in Dokumenten gespeichert. Hierfür müssen die zu erfassenden Entitäten definiert werden. Abbildung 13 zeigt die Implementierung eines Mitarbeiters mit dessen Attributen. Für alle Attribute werden sogenannte Annotationen verwendet. Sie sind spezielle Interfaces, die die Einbindung von Metadaten in den Quelltext ermöglichen[18]. „@JsonProperty“ wird genutzt, um den Namen des Schlüssels zu definieren. Er wird in einem JSON-Objekt verwendet, sodass eine Konvertierung zwischen der Variablen und dem Datentyp möglich ist[19]. Mit der Annotation „@Valid“ wird ein Objekt, das zum Beispiel aufgrund einer POSTAnfrage erstellt wurde, automatisch auf Korrektheit überprüft. Sind übergebene Daten unzulässig, wird automatisch ein entsprechender HTTP-Response generiert[23]. Beispielsweise ist ein Objekt fehlerhaft, wenn anstelle eines Integers ein anderer Datentyp wie ein String angegeben wird. „@ID“ legt die Variable fest, anhand der ein Objekt in der Datenbank identifiziert ist. Bei der Entität Mitarbeiter ist die ID ein Kürzel.
Dieses muss für die Datenbank einzigartig sein.

[18] vgl. [Hor18].
[19] vgl. [Eug15].
[23] vgl. [Gil19].

```
public class Employee   {
  @JsonProperty("abbreviation")
  @Id
  private String abbreviation = null;
  @JsonProperty("role")
  private RoleEnum role = null;
@JsonProperty("firstName")   private String
firstName = null;   @JsonProperty("lastName")
private String lastName = null;
  @JsonProperty("email")   private String email =
null;   @JsonProperty("birth")   private String
birth = null;   @JsonProperty("photoUrl")
private String photoUrl = null;
  @JsonProperty("projectLinks")
  @Valid
  private List<EmployeeProjectLink> projectLinks = null;
  @JsonProperty("technologyLinks")
  @Valid
  private List<EmployeeTechnologyLink> technologyLinks = null;
  @JsonProperty("bootcampLinks")
  @Valid
  private List<EmployeeBootcampCertificateLink> bootcampLinks = null;
  @JsonProperty("certificateLinks")
  @Valid
  private List<EmployeeBootcampCertificateLink> certificateLinks = null;   [...] //further functions and
constructors }
```

Abbildung 13 - Entität „Employee“

4.2.2 Erstellen der Datenbank

Die Datenbank wird mit der Implementierung eines Interfaces erstellt. Dieses ist von der Basisklasse „MongoRepository“ abgeleitet. Es muss angegeben werden, welche Entität gespeichert werden soll. Abbildung 14 zeigt die Implementierung für Mitarbeiter. Das Interface umfasst für die Anwendung notwendige Methoden. Sie dienen der Abfrage der Datenbank. Bei Beachtung der durch Spring vorgegebenen Syntax wird die Logik automatischen geschrieben. Bei Nutzung der Methode „findByAbbreviation“ wird der Mitarbeiter mit dem entsprechenden Kürzel returniert. Da jeder Wert für das Attribut einmalig ist kann maximal ein Objekt gefunden werden. Im Gegensatz dazu liefert „findByProjectLinksLinkName“ eine Liste von Mitarbeitern, die in dem angegebenen Projekt arbeiten. Weitere Funktionen generieren einen Wahrheitswert. Mit ihnen wird überprüft, ob ein Mitarbeiter mit angegebenem Kürzel und einer bestimmten Eigenschaft existiert. Beispielsweise liefert die Methode „existsByAbbreviationAndCertificateLinksLinkName“ den Wert „true“, wenn die Person ein gewünschtes Zertifikat besitzt.

```
public interface EmployeeRepository extends
                                                MongoRepository<Employee,String>{
   Employee findByAbbreviation(String abbreviation);
   List<Employee> findByProjectLinksLinkName(String name);
   List<Employee> findByCertificateLinksLinkName(String name);
   List<Employee> findByBootcampLinksLinkName(String name);
   List<Employee> findByTechnologyLinksLinkName(String name);
   List<Employee> findByFirstName(String name);     List<Employee>
findByLastName(String name);
   boolean existsByAbbreviationAndProjectLinksLinkName(String
                                                    abbreviation, String projectName);
   boolean existsByAbbreviationAndTechnologyLinksLinkName(String
                                                    abbreviation, String technologyName);
   boolean existsByAbbreviationAndBootcampLinksLinkName(String
                                                   abbreviation, String bootcampName);
   boolean existsByAbbreviationAndCertificateLinksLinkName(String
                                                     abbreviation, String certificateName);
}
```

Abbildung 14 - Datenbank für Mitarbeiter

4.3 Verwendung der Schnittstelle

4.3.1 Erstellen eines Projektes

Um ein Projekt in der Datenbank anzulegen muss die Benutzerschnittstelle über eine HTTP-Anfrage mit der Methode POST angesprochen werden. Dafür muss der entsprechende URL genutzt werden. Dieser entspricht „http://localhost:8080/project“ für eine Anwendung, die auf dem lokalen Rechner ausgeführt wird. Zusätzlich muss dem Request Body der Anfrage ein Projekt hinzugefügt werden. Es wird im Format JSON angegeben. Zur Demonstrierung soll ein Projekt mit dem Namen „ProjektTest“ angelegt werden. Unter dem Schlüssel „employeeLinks“ wird eine Relation zu einem Mitarbeiter angegeben. Die entsprechende Anfrage ist in der Abbildung 15 dargestellt.

```
POST http://localhost:8080/project
Content-Type: application/json

{
 "name": "ProjektTest",
 "systemLinks": [],
 "employeeLinks": [
   {
    "role": "Developer",
    "from": "01.08.2019",
    "to": "23.09.2020",
    "link": {
     "datatype": "Employee",
     "link": "http://localhost:8080/employee/findByAbbreviation/Test",        "name": "Test"
    }
   }
 ] }
```

Abbildung 15 - Post-Anfrage zum Anlegen eines Projektes

4.3.2 Abfrage eines Projektes

Über GET-Anfragen an den entsprechenden Controller kann das angelegte Projekt abgefragt werden. Für die HTTP-Anfragen ist ein String der im URL als Pfadparameter angegeben wird erforderlich. Dieser wird mit einem Attribut der Projekte abgeglichen. Welches Attribut betrachtet wird, hängt von der Wahl des API-Endpunktes ab, der angefragt wird. Beispielsweise kann einer der beiden URLs, die in der Abbildung 16 angegeben sind, verwendet werden, um das angelegte Projekt als Response zu erhalten.

```
GET http://localhost:8080/project/findByEmployee/Test

GET http://localhost:8080/project/findByName/TestProjekt
```

Abbildung 16 - Beispielabfragen für Projekte

5 Fazit

Ziel der vorliegenden Arbeit war es eine Applikation zu entwickeln, welche Mitarbeiterinformationen erfasst. Es sollte die Datenhaltung modelliert und die Bereitstellung der Daten über eine Benutzerschnittstelle realisiert werden.

Nach der Erläuterung des Datenmodells sowie des Systemüberblicks wurde die Wahl der Datenbank begründet. Mit MongoDB nutzt die Anwendung ein dokumentorientiertes Datenbankmodell. Anschließend erfolgte ein Überblick über die Realisierung der Applikation. Entsprechend der Anforderungen wurde eine Anwendung implementiert. Mit dieser können Mitarbeiterinformationen persistiert und abgefragt werden. Mittels HTTPAnfragen an die entwickelte Benutzerschnittstelle wird die Anwendung genutzt. Das entstandene Programm bedient sich dem Framework Spring. Über dieses kann ein Controller mit der Datenbank kommunizieren.

Mit der Applikation können die in der Einleitung aufgeführten Anwendungsfälle gelöst werden. Beispielsweise kann eine Anfrage eine Liste von Mitarbeitern liefern, welche ein definiertes Zertifikat besitzen. Durch die Abfrage der Schnittstelle kann ebenfalls bestimmt werden, welche Person angegebene Technologien beherrscht.

Die aktuelle Lösung beinhaltet keine Clientimplementierung. Das bedeutet, dass für den Anwender kein Frontend zur Verfügung steht. Da das verwendete Datenformat JSON bei großer Menge von Informationen für den Menschen nur schwer erfassbar ist, sollte ergänzend ein Frontend umgesetzt werden. Dieses kann Daten passend visualisieren und die Nutzung der Applikation vereinfachen.

Als Kritikpunkt ist anzumerken, dass die Implementierung an einigen Stellen nicht dem typischen Entwurfsmuster einer API folgt. Beim Anlegen eines Objekts mittels einer HTTP-Anfrage soll lediglich eine Ressource in die Datenbank gespeichert werden. Im Gegensatz dazu werden in der entstandenen Lösung oftmals mehrere Ressourcen beim Ansprechen eines API-Endpunktes aufgenommen. Grund dafür ist, dass darauf geachtet wird, Beziehungen konsistent zu persistieren. So wird eine Relation in beiden beteiligten Entitäten gespeichert.

Weiterhin kann die Benutzerschnittstelle von allen externen Personen genutzt werden. Da es sich um sensible Informationen des Unternehmens handelt, ist es sinnvoll, eine

Authentifizierung einzurichten. Mit dieser kann sich der Anwender identifizieren und der Zugriff auf Angestellte beschränkt werden.

6 Literaturverzeichnis

[Beg20] Begerow Beratungsgesellschaft mbH & Co. KG, „Entity-Relationship-Modell (ER-Modell / ERM) | Datenmodellierung Grundlagen“, 08.08.2020+00:00, https://www.datenbanken-verstehen.de/datenmodellierung/entity-relationship-modell/, Abgerufen am: 01.09.2020.197Z

[Eug15] Eugen Paraschiv, „Jackson Annotation Examples | Baeldung“, 2015, https://www.baeldung.com/jackson-annotations, Abgerufen am: 11.09.2020.161Z

[Gil19] Gilbert Lopez, „Spring Boot Bean Validation Example | Examples Java Code Geeks - 2020“, 2019, https://examples.javacodegeeks.com/spring-boot-bean-validationexample/, Abgerufen am: 11.09.2020.067Z

[Hol20] Holger Reibold, „HTTP-Methoden - HTTP-Grundlagen: Hypertext Transfer Protocol - TecChannel Workshop“, 01.09.2020.000Z, Abgerufen am: 01.09.2020.626Z

[Hor18] Horn, Torsten, „Annotations (ab Java 5)“, 30.07.2018.000Z, https://www.torsten-horn.de/techdocs/java-annotations.htm, Abgerufen am: 11.09.2020.106Z

[ION20] IONOS Digitalguide, „Dokumentenorientierte Datenbank“, 01.09.2020.000Z, https://www.ionos.de/digitalguide/hosting/hostingtechnik/dokumentenorientierte-datenbank/, Abgerufen am: 01.09.2020.884Z

[Ion20] Ionos (1&1), „MongoDB: Vorstellung und Vergleich mit MySQL“, 28.07.2020.000Z, https://www.ionos.de/digitalguide/websites/webentwicklung/mongodb-vorstellung-und-vergleich-mit-mysql/, Abgerufen am: 28.07.2020

[Mar20] Margaret Rouse, „Was ist Model View Controller (MVC)? - Definition von WhatIs.com“, 10.09.2020.000Z,

https://www.computerweekly.com/de/definition/Model-View-Controller-MVC, Abgerufen am: 10.09.2020.246Z

[Ora20a] Oracle, „Was ist eine relationale Datenbank?“, 24.08.2020.000Z, https://www.oracle.com/de/database/what-is-a-relational-database/, Abgerufen am: 31.08.2020.639Z

[Ora20b] Oracle, „Was ist eine Datenbank?“, 24.08.2020.000Z, https://www.oracle.com/de/database/what-is-database.html, Abgerufen am: 31.08.2020.570Z

[Sas19] Sascha Thattil, „MySQL oder MongoDB: Welches Datenbanksystem ist besser?“, 2019, https://www.yuhiro.de/mysql-oder-mongodb/, Abgerufen am: 30.07.2020

[SEL20] SELFHTML e.V., „HTTP/Anfragemethoden – SELFHTML-Wiki“, 28.06.2020.000Z, https://wiki.selfhtml.org/wiki/HTTP/Anfragemethoden, Abgerufen am: 01.09.2020.402Z

[Tin08] Tino Hempel, „Kardinalität - einfache Version nach Chen“, 23.08.2008.000Z, https://tinohempel.de/info/info/datenbank/kardinalitaet.htm, Abgerufen am: 31.08.2020.606Z

[Tut20] Tutorials Point, „HTTP - Überblick - Tutorialspoint“, 26.08.2020.000Z, https://www.tutorialspoint.com/de/http/http_overview.htm, Abgerufen am: 01.09.2020.536Z

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich,

1. dass ich meine Studienarbeit mit dem Thema:

   Mitarbeiter Skills - Applikation zur Erfassung von Mitarbeiterfähigkeiten

   ohne fremde Hilfe angefertigt habe,
2. dass ich die Übernahme wörtlicher Zitate aus der Literatur sowie die Verwendung der Gedanken anderer Autoren an den entsprechenden Stellen innerhalb der Arbeit gekennzeichnet habe und
3. dass ich meine Studienarbeit bei keiner anderen Prüfung vorgelegt habe.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Jena, 02.10.2020

Ort, Datum Unterschrift