Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Marcus Jenz

Analyse und Neuentwicklung eines bestehenden webbasierten Kompetenzmanagementsystems

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Marcus Jenz

Analyse und Neuentwicklung eines bestehenden webbasierten Kompetenzmanagementsystems

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachterin: Prof. Dr. Ulrike Steffens

Eingereicht am: 11.11.2020

Marcus Jenz

Thema der Arbeit
Analyse und Neuentwicklung eines bestehenden webbasierten Kompetenzmanagementsystems

Stichworte
Softwareengineering, Webentwicklung, Rest, Symfony, Angular, Client-Server Architektur

Kurzzusammenfassung
Diese Arbeit befasst sich mit der Analyse eines bestehenden Projekts und dessen Neuentwicklung innerhalb eines Unternehmens. Ziel dieser Arbeit ist, einen möglichen Weg der Realisierung der Neuentwicklung der ursuprünglichen Funktionalitäten, sowie gegebenen Anforderungen, bis zu einem realisierbaren System aufzuzeigen. Es werden Konzepte sowie Praktiken der Analyse sowie die Entwicklung eines und dessen Umsetzung vorgestellt.
Teil dieser Arbeit ist die Umsetzung einer Auswahl der Anforderungen

Marcus Jenz

Title of the paper
Analysis and redevelopment of an existing web based competence management system

Keywords
software engineering, web development, rest, Symfony, Angular, client-server architecture

Abstract
This thesis deals with the analysis of an existing project and its new development within a company. The goal of this work is to show a possible way of realizing the new development of the original functionalities and given requirements to a realizable system. Concepts and practices of analysis as well as the development and implementation of a new system are presented.
Part of this work is the implementation of a selection of the requirements

Inhaltsverzeichnis

Abbildungsverzeichnis

Auflistungen

1 Einleitung

Diese Kapitel bietet einen Überblick über diese Arbeit. Sie Beschreibt die Motivation zu Beginn und das erwünschte Ziel des Projekts.
Innerhalb der Motivation wird verdeutlicht, warum das aktuelle Tool durch ein Neues ersetzt werden soll. Hierbei wird die generelle notwendigkeit des Tools betrachtet.
In der Zielsetzung wird das erwünschte Resultat dieser Arbeit aufgezeigt. Dies beinhaltet sowohl der Weg den Entwicklung als auch die Realisierung der Anwendung.
Alle Personengruppen die im Rahmen der Ausarbeitung genannt werden, schließen alle Geschlechter, ohne explizite Nennung, mit ein.

1.1 Motivation

Um die Machbarkeit von Projekten einschätzen zu können, ist es notwendig zu wissen, ob die Arbeiterinnen die notwendigen Kompetenzen besitzen, oder ob essenzielle Fertigkeiten fehlen.
In Abbildung 1.1 ist das seit Jahren positive Wirtschaftswachstum zu sehen. Die Ausnahme in den Jahren 2008 und 2009 sind durch die globale Finanz- und Wirtschaftskrise [12] zu erklären, und daher in dem Trend zu ignorieren. Durch dieses Wirtschaftswachstum, wachsen oft auch kleinere Firmen auf ein Maß, bei dem es schwer wird einen Überblick über die Fähigkeiten der einzelnen Arbeiterinnen zu haben. Insbesondere wenn die jeweiligen Angestellten sich privat weiterbilden, ist ein zusätzlicher Aufwand notwendig um im Bilde zu sein, wie sich Fertigkeiten verteilen.
In Anbetracht dessen, ist es wichtig im mer ein aktuelles Bild der vorhandenen und nicht vorhandenen Kompetenzen zu haben, ohne dabei unnötig viel Aufwand zu betreiben.
Um dieses Problem sinnvoll und ohne viel Aufwand zu lösen, benötigt man Tools mit denen man die Kompetenzen und Skills der Angestellten verwalten, filtern, und in geeigneter Form anzeigen kann, also ein klassisches Informationssystem.
Ein solches Informationssystem gibt es bereits in der Firma Silpion IT Solutions GmbH, dieses

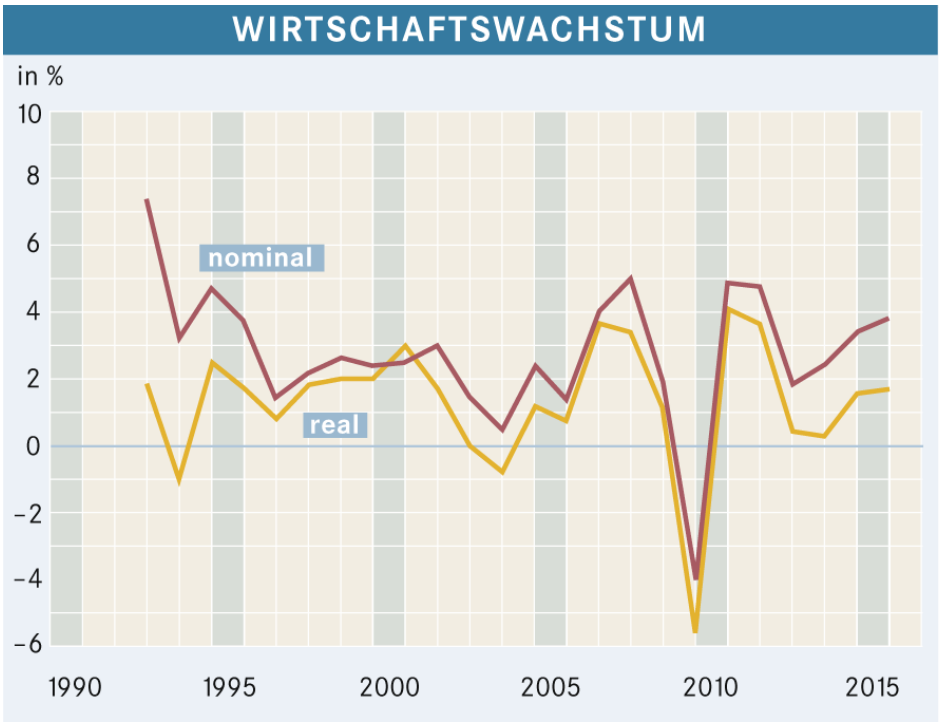

Abbildung 1.1: Wirtschaftswachstum 1990 bis 2015 in Deutschland [12]

erfüllt jedoch nur die grundlegenden Ansprüche, einer Firma die stetig wächst. Viele Stakeholder der Anwendung wünschen sich neue Funktionalitäten, und Verbesserungen von bestehenden Funktionalitäten. Das bestehende System, die SkillDB, wurde bisher nur weiterentwickelt, wenn der Entwickler zwischen Projekten Zeit hatte. Da für die Entwicklung eine Technologie benutzt wurde, für die es wenig Kompetenzen im Haus gibt, konnten diese Arbeiten nicht von anderen Entwicklern übernommen werden. Unter diesem Umständen ist die SkillDB und seine technologische Basis veraltet, während Funktionsanforderungen sich häuften.
Hierdurch besteht eine Notwendigkeit, entweder Kompetenzen aufzubauen, oder die SkillDB mit moderneren, im Hause weiterverbreiteten Technologien zu überarbeiten. Während es oft sinnvoll ist neue Kompetenzen aufzubauen und sich zu diversifizieren, wurde entschieden, dass weitere Kompetenzen im Grails-Bereich für Silpion nicht viel Nutzen haben.

1.2 Zielsetzung

Ziel dieser Bachelorarbeit ist es einen Weg zu zeigen, um ein bereits bestehendes System abzulösen, indem ein neues System entwickelt wird, das mindestens den Funktionsumfang des bereits bestehenden Systems beinhaltet. Dies wird anhand des, in der betreuenden Firma bestehenden, Systems 'Skill DB' beispielsweise aufgezeigt.
Es wird iterativ und inkrementell entwickelt, und dadurch stetig erweitert, verbessert und Fehler behoben. Dabei muss beachtet werden, dass die Anwendung einfach erweiterbar sein muss, da dies zum einen ein Indikator für gute Softwarequalität ist, und zum anderen bereits einige weitere Funktionswünsche vorhanden sind, die außerhalb des Rahmens dieser Bachelorarbeit umgesetzt werden sollen.
Am Ende dieser Arbeit soll ein Informationssystem vorhanden sein, dass zumindest die grundsätzlichen bisherigen Funktionen des bestehenden Systems besitzt. Dabei soll das neue System aus einer neuen allgemeinen Schnittstelle bestehen, die mit unterschiedlichen Clients konsumiert werden kann. Als Client wird in dieser Arbeit eine grafische Benutzerschnittstelle umgesetzt, die sich am "mobile first"Design orientiert. Also ein Design das sich verschiedenen Displaygrößen anpasst, damit man die Anwendung auf Desktop Geräten, Smartphones und Tablets gleichermaßen verwenden kann. Zusätzlich notwendige Anforderungen, die für einen Betrieb notwendig sind, sind nicht teil der Bachelorarbeit, und werden erst im Anschluss der Arbeit umgesetzt.

1.3 Struktur der Arbeit

Diese Arbeit ist in sechs Kapitel aufgeilt, deren Gliederung im Folgenden beschrieben wird.
Im zweiten Kapitel wird ein Überblick über das bestehende System gegeben. Das System wird mit Hilfe von User Stories beschrieben, die als Mindestvoraussetzung gelten.
In Kapitel drei wird ein Konzept dargestellt, dass die aus Kapitel zwei entstandenen Spezifikationen als Basis nimmt, um den Weg der Realisierung aufzuzeigen, der im Rahmen dieser Bachelorarbeit ausgewählt wurde. Es werden interne Abläufe gezeigt, und dabei eine Unterscheidung zwischen Client und Schnittstelle durchgeführt. Es werden außerdem Technologien und Frameworks festgelegt.
Das vierte Kapitel beschäftigt sich mit der Realisierung der Neuentwicklung. Hier werden sowohl das Vorgehen, als auch die Werkzeuge, die genutzt wurden um die Entwicklung durchzuführen, vorgestellt.

Kapitel fünf evaluiert das umgesetzte System und erläutert Probleme die im Laufe dieser Arbeit aufgetreten sind.
Im sechsten und letzten Kapitel, wird das Ergebnis analysiert. Außerdem wird ein Ausblick auf die weitere Verwendung, sowie die Weiterentwicklung gegeben.

2 Analyse

In diesem Kapitel werden die Funktionalitäten des bestehenden Systems, mithilfe der Zielsetzung aus dem ersten Kapitel, ausgearbeitet.
Innerhalb der Problemstellung, wird dargestellt warum ein Kompetenzmanagementsystem notwendig ist, und der bisherige Funktionsumfang nicht ausreichend ist. Dabei wird der Ist-Zustand der aktuellen Software anhand von User Stories festgehalten.

2.1 Problemstellung

In der betreuenden Firma, gibt es mehr als 150 Mitarbeiterinnen [58], von denen viele breit gefächerte Kompetenzen besitzen, sich selbst weiterbilden oder durch Konferenzen und Schulungen weitere Kompetenzen erlangen. Deswegen fällt es oft sehr schwer die Kompetenzen überblicken zu können und immer ein aktuelles Bild der Kenntnisse zu haben. Falls ohne ein solches Tool das Wissen über Kompetenzen vorhanden ist, ist es schwer dieses Wissen ganzheitlich weitergeben zu können.

2.2 Ist-Zustand

Das aktuell genutzt Informationssystem bildet mehrere User Stories und Rollen ab. In diesem Abschnitt werden vorhandenen Rollen und User Stories vorgestellt. Anschließend wird die technische Umsetzung der Anwendung beschrieben.

2.2.1 Rollen

Der aktuelle Stand der SkillDB bildet unterschiedliche Rollen mit unterschiedlichen Aufgaben ab. Allerdings existieren Rollen die niemandem zugewiesen sind, und es existieren ebenfalls

Rollen die keinen Effekt haben. Im folgenden werden alle Rollen beschrieben die existieren, unabhängig der Funktion und ob sie aktiv benutzt werden.

Sucher Diese Rolle wird Mitarbeitern zugewiesen, die nach Profilen suchen dürfen.

Mitarbeiter Diese Rolle wird Mitarbeitern zugewiesen, die fest bei Silpion angestellt sind. Diese Rolle hat keine Auswirkungen

Administrator Diese Rolle wird Mitarbeitern zugewiesen, die Abteilungsleiter sind.

Freier Mitarbeiter Diese Rolle wird Mitarbeitern zugewiesen, die nicht bei der Firma angestellt sind, sondern nach bedarf angeheuert werden. Diese Rolle hat jedoch keine Auswirkungen.

Content Manager Diese Rolle wird Abteilungsleitern zugewiesen.

Hangaround Diese Rolle hat keinerlei Auswirkung. Sie existiert ist aber niemandem zugeordnet.

Benutzer Diese Rolle ist jedem authentifizierten Benutzer zugeordnet.

2.2.2 User Stories

Um einzelne Funktionalitäten und deren Zusammenhang im Gesamtsystem zu beschreiben kann man User Stories benutzen. User Stories sind, wie der Name schon sagt, um den Benutzer zentriert. Eine User Story ist die Kurzbeschreibung eines konkreten Szenarios, das ein Benutzer erfährt. User Stories können formlos geschrieben sein, sind aber oft in der Form *Äls <Rolle> möchte ich <Ziel/Wunsch>, um <Nutzen>"* [3] gehalten. Hierzu kann man Akzeptanzkriterien definieren, die erfüllt werden müssen, um die User Story abschließen zu können. Diese Akzeptanzkriterien können funktional oder nicht-funktional sein. Diese kann man benutzen um sogenannte Akzeptanztests zu erstellen.

Login Als Benutzer möchte ich mich mit meinen Zugangsdaten anmelden, damit ich das Programm benutzen kann
Akzeptanzkriterien:

- Nach einem erfolgreichen Login wird man auf eine Übersichtsseite weitergeleitet.
- Bei einem fehlerhaften Login bekommt man eine entsprechende Rückmeldung.
- Die Zugangsdaten sind die, die im Firmenumfeld benutzt werden.
- Auch wenn man noch nie das System benutzt hat, kann man sich direkt einloggen, ohne dass ein Administrator einen Benutzer anlegen muss.

Kontaktdaten pflegen Als Benutzer möchte ich meine Kontaktdaten pflegen, damit Sucher Kontakt mit mir aufnehmen können
Akzeptanzkriterien:

- Es muss ein Name angegeben werden.
- Es muss eine E-Mailadresse angegeben werden.
- Es kann eine Telefonnummer angegeben werden.
- Es kann ein Bild gepflegt werden.

Allgemeinen Informationen pflegen Als Benutzer möchte ich meine Allgemeinen Informationen pflegen, damit Sucher besser wissen wer ich bin.
Akzeptanzkriterien:

- Es muss eine Berufsbezeichnung angegeben werden.
- Es muss eine Abteilung angegeben werden.
- Es kann ein Geburtsdatum angegeben werden.
- Es können Hobbies angegeben werden.
- Es kann ein Firmeneintrittsdatum angegeben werden.
- Es können berufliche Interessen angegeben werden.
- Es können Tätigkeitsschwerpunkte und Stärken angegeben werden.
- Es kann Reisebereitschaft als Ja/Nein Auswahl angegeben werden.
- Es kann Reisedistanz angegeben werden.
- Es kann Reisedauer angegeben werden.

Fähigkeiten pflegen Als Benutzer möchte ich meine Fähigkeiten pflegen, damit Sucher diese sehen können.
Akzeptanzkriterien:

- Fähigkeiten bestehen aus Fähigkeitsname und Fähigkeitsniveau.
- Fähigkeiten müssen aus bereits bestehenden Fähigkeiten ausgewählt werden.

Projekte pflegen Als Benutzer möchte ich meine Projekte pflegen, damit Sucher diese sehen können.
Akzeptanzkriterien:

- Es gibt eine Liste von Projekten.

- Ein Projekt besteht aus: Projektname, Kundenname, Projektbeschreibung, Projektbeginn, abgeschlossenes Projekt (Ja/Nein), Projektende, Meine Rolle im Projekt, Beschreibung meiner Tätigkeiten im Projekt, Größe des Team, eingesetze Fertigkeiten
- Eine eingesetzte Fähigkeit muss eine Fähigkeit sein, die ich in meinem Profil eingepflegt habe.

Abschlüsse und Sprachen pflegen Als Benutzer möchte ich meine Bildungsweg und Sprachen pflegen, damit Sucher diese sehen können.
Akzeptanzkriterien:

- Es gibt eine Liste von Abschlüssen.
- Ein Abschluss besteht aus Institution und Abschlussname
- Es gibt eine Liste von Sprachen.
- Eine Sprache besteht aus Sprachname und Niveau
- Niveau ist eins von: Grundkenntnisse, Gute Kenntnisse, fließend, verhandlungssicher, Muttersprache

Suche Als Sucher möchte ich nach Benutzern suchen, damit ich deren Profil aufrufen kann.
Akzeptanzkriterien:

- Alle Suchbegriffe müssen enthalten sein.
- Bei der Suche wird Groß- und Kleinschreibung ignoriert.
- Falls kein Benutzer gefunden wird, möchte ich eine entsprechende Rückmeldung erhalten.
- Es werden Namen und Fertigkeiten durchsucht.
- Suchergebnisse enthalten einen Link um das Profil der jeweiligen Benutzer anzuzeigen

Eigenes Profil anzeigen Als Benutzer möchte ich mein eigenes Profil anzeigen können, damit ich es überprüfen kann.
Akzeptanzkriterien:

- Es werden die Kontaktdaten angezeigt.
- Es werden die Allgemeinen Informationen angezeigt.
- Es werden die Fähigkeiten angezeigt.

- Es werden die Projekte angezeigt.
- Es werden die Abschlüsse und Sprachen angezeigt.

Fremdes Profil anzeigen Als Sucher, Administrator, oder Content Manager möchte ich fremde Profile anzeigen können, damit ich Informationen über den Besitzer bekomme.
Akzeptanzkriterien:

- Es gibt eine Datei mit einem Design dass der Corporate Identity entspricht.
- Es gibt eine Datei mit einem neutralen Design.
- Es ist möglich ein PDF herunterzuladen.
- Es ist möglich ein Word Dokument herunterzuladen.
- Es werden die fremden Kontaktdaten angezeigt.
- Es werden die fremden Allgemeinen Informationen angezeigt.
- Es werden die fremden Fähigkeiten angezeigt.
- Es werden die fremden Projekte angezeigt.
- Es werden die Abschlüsse und Sprachen angezeigt.

Eigenes Profil druckbar speichern Als Benutzer möchte ich mein eigenes Profil als ausdruckbare Datei herunterladen können, damit ich Informationen über mich an Firmenfremde Personen weitergeben kann.
Akzeptanzkriterien:

- Es gibt eine Datei mit einem Design dass der Corporate Identity entspricht.
- Es gibt eine Datei mit einem neutralen Design.
- Es ist möglich ein PDF herunterzuladen.
- Es ist möglich ein Word Dokument herunterzuladen.
- Es werden die Kontaktdaten angezeigt.
- Es werden die Allgemeinen Informationen angezeigt.
- Es werden die Fähigkeiten angezeigt.
- Es werden die Projekte angezeigt.
- Es werden die Abschlüsse und Sprachen angezeigt.

Verfügbare Fertigkeiten verwalten Als Administrator oder Content Manager möchte ich Fertigkeiten verwalten können, damit nur geprüfte Fertigkeiten auf einem Profil gepflegt werden.
Akzeptanzkriterien:

- Fertigkeiten werden Kategorien zugeordnet.
- Fertigkeiten können deaktiviert werden.
- Wenn Fertigkeiten gelöscht werden, werden sie auch aus allen Profilen gelöscht.
- Fertigkeitsnamen sind eindeutig.

Verfügbare Fertigkeitskategorien verwalten Als Administrator oder Content Manager möchte ich Kategorien von Fertigkeiten verwalten können, damit Fertigkeiten einfacher zu verwalten und pflegen sind.
Akzeptanzkriterien:

- Kategorien können nur gelöscht werden, wenn sie leer sind.
- Namen der Kategorien sind eindeutig

Benutzer verwalten Als Administrator möchte ich Benutzer verwalten können, damit Rollen vergeben und alte Daten löschen kann.
Akzeptanzkriterien:

- Wenn ein Benutzer gelöscht wird, wird auch sein Profil gelöscht, und alle dazugehörigen daten.
- Benutzerrollen können aus den bestehenden Rollen ausgewählt werden.

2.2.3 Datenmodell

Dieser Abschnitt behandelt des zugrundeliegende Datenmodell für die Datenbank der Applikation. Abbildung 2.1 zeigt das Datenmodell als Entity-Relationship-Modell anhand eines UML Diagrams an.
Jeder Systembenutzer (SystemUser) kann beliebig viele Rollen (Role) besitzen. Es ist möglich Benutzer auszusperren indem man dem SystemUser alle Rollen entzieht. Jeder Benutzer darf genau ein Profil (Profile) besitzen. Ein Benutzer hat lediglich die logischen Felder Benutzername (username), Vorname (givenName), Nachname (lastName) und Abteilung (department). Diese Informationen sind zwar alle redundant, aber sie kommen von dem LDAP-Server, sodass hier eine offizielle Version der Informationen vorhanden ist. Diese Informationen werden genutzt

um das JWT zu genieren und mit Informationen zu befüllen. Die 1:1 Beziehung zwischen Profil und Benutzer regt an, Profil und Benutzer in eine Entität zusammenzuführen. Jedoch ist es in diesem Falle Sinnvoll den Benutzer nicht mit unnötigen Informationen zu beladen, da dieser Benutzer für Autorisierungen benutzt wird. Dies ermöglicht es, einfacher die Implementierung Authentifizierung auszutauschen. Hierdurch ist es einfach möglich die LDAP Authentifizierung durch eine Lokale auszutauschen.
Das Profil hat genau ein mal allgemeine Informationen (CommonInformation). Diese 1:1 Beziehung ist eine die historisch mitgenommen wurde, die Sinnhaftigkeit dieser Beziehung ist fragwürdig, und wird in zukünftigen Iterationen entfernt. Der Unterschied dieser Informationen ist lediglich ein Semantischer. Das Profil enthält Kontaktinformationen und die allgemeinen Informationen sind solche die in einem Portfolio für einen Kunden interessant sein könnten. Ein Profil hat beliebig viele Ausbildungen (Education), Sprachen (Language), Projekte (Project) und eigene Fertigkeiten (ProfileSkills).
Eine Ausbildung kann, hat einen Abschluss (graduation) und eine Institution (institution). Sie kann ein Schulabschluss sein, ein Zertifikat, ein Studium oder auch eine Berufsausbildung sein. Eine Sprache ist eine ähnlich einfache Entität wie eine Ausbildung, sie besitzt den Namen (name) und das Sprachniveau das man spricht (level).
Eigene Fertigkeiten (ProfileSkills) ist vereinfacht gesagt, eine M:N Beziehung zwischen Fertigkeiten und Profilen, die jedoch noch eine zusätzliche Information enthält. Daher ist die Struktur einer eigenen Fertigkeit denkbar einfach, die einzige angereicherte Information ist hierbei die Bewertung der Fertigkeit. Eine globale Fertigkeit (Skill) ist ebenfalls ein sehr einfaches Objekt, das nur einen Namen und die information ob es aktiv ist enthält. Jede Fertigkeit ist einer Fertigkeitskategorie (SkillCategory) zugeordnet, wobei es auch Kategorien geben kann, die keine Fertigkeiten besitzen.
Ein Projekt ist eine Sammlung primitiven Daten über ein Projekt, wie Start- und Endzeitpunkt, hat aber auch eine N:M verknüpfung zu den eigenen Fertigkeiten. Dies ermöglicht es einem Benutzer anzugeben, welche Fertigkeiten er in seinen Projekten anwenden konnte.

2.2.4 Technische Umsetzung

Die Applikation ist eine monolithische MVC Anwendung die mit dem Framework Grails der Version 2.4.3 entwickelt wurde [19]. Grails ist ein open-source Framework von Object Computing dass sich im Java Umfeld befindet, und auf Spring Boot basiert. Grails ermöglicht es Applikationen mit der Sprache Groovy der Apache Foundation zu entwickeln [20]. Groovy ist eine optional Typisierte Sprache die zu Java Bytecode compiliert wird. Daher ist es nicht

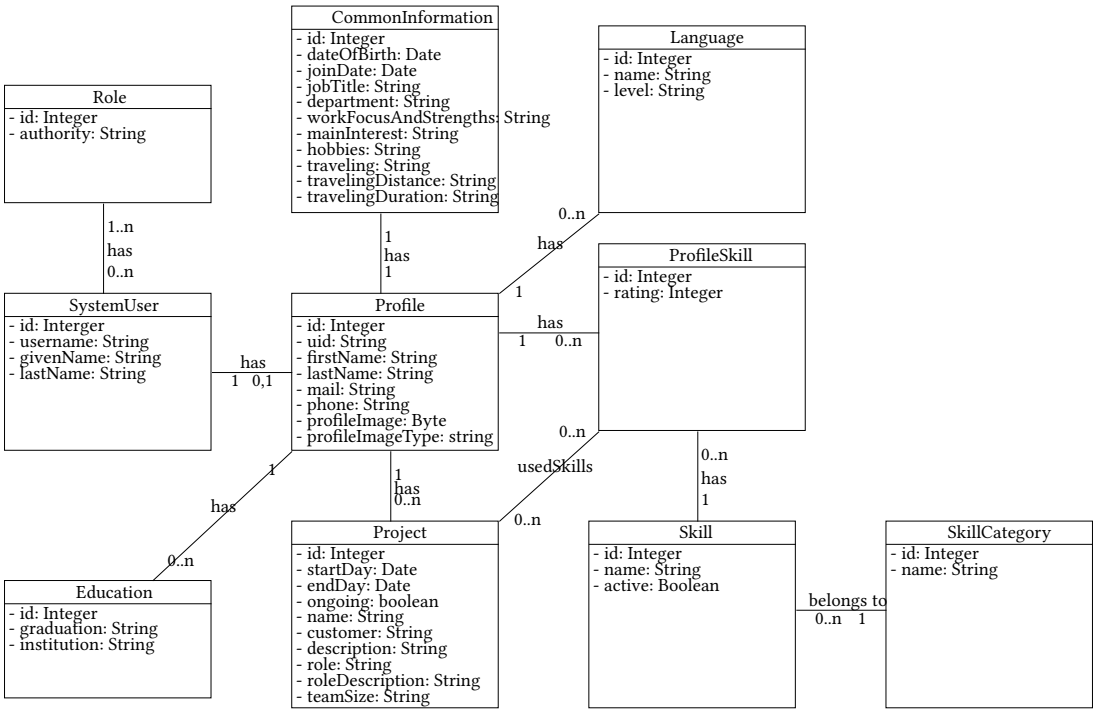

Abbildung 2.1: Entity Relationship Diagramm

zwangsläufig notwendig Groovy zu benutzen. Die anfallenden Daten werden in einer postgresql Datenbank persistiert. Während die Authentifizierung mithilfe des Firmeninternen LDAP Services erledigt wird, wird die Autorisierung durch die Applikation durchgeführt.

2.2.5 Probleme

Auch wenn die aktuelle Version funktioniert gibt es einige Probleme, die sich durch die aktuelle Applikation bilden. Die Applikation läuft unter Java 7, welches seit Juli 2015 keine öffentlichen Updates mehr bekommt. Stand Januar 2020, sind es nun etwa 4,5 Jahre ohne Sicherheitsupdates, was mehr als bedenklich ist, und Handlungsbedarf aufzeigt.
Das Grails Framework ist zumindest bei Silpion IT Solutions GmbH nicht weit verbreitet, sodass das Expertenwissen so sehr konzentriert ist, dass Ansprechpartner praktisch kaum verfügbar sind. Infolge dessen ist die Wartung des Tools sehr lange auf der Strecke geblieben. Dies drückt sich auch durch die verwendeten Versionen der Werkzeuge aus, die genutzt wurden.

Das Grails Framework in der Version 2.4 wird nicht mehr weiterentwickelt, erst die Version 2.5 bekommt nur noch Updates, die kritische Sicherheitslücken schließen.

Um Grails auf eine aktuelle Version zu heben, also um Updates zu bekommen die nicht nur kritische Sicherheitslücken schließen, müsste man das Framework auf die Version 3.3 anheben. Dies würde also ein Major update beinhalten, das viele Breaking Changes beinhaltet.
Die Dokumentation für Grails 3.0 [18] merkt im Bereich des Upgrades auf Grails 3.0 an: *Grails 3.0 is a complete ground up rewrite of Grails and introduces new concepts and components for many parts of the framework.* Und dass dies weitreichende Folgen hat, die bei einem Update zu bedenken sind. Namentlich nennt die Dokumentation unter anderem die beiden Punkte des Buildsystems und Änderungen bei Plugins. In vorherigen Versionen hat Grails auf weniger verbreitete Buildsysteme gesetzt, in der aktuellen Version des Tools kommt dadurch Aether der Eclipse Foundation zum Einsatz. Zu Aether existiert keine offizielle dokumentation mehr. Es gibt ein veraltetes Repository [47], welches auf eine nicht mehr existierende Seite der Eclipse foundation verweist. Dadurch ist einiges an Aufwand nötig um die Änderung des Buildsystems umzusetzen. Die weitere große Herausforderung, die bei einem Upgrade bestünde, sind die genutzten Plugins, also Bibliotheken von Drittanbietern. Durch die Änderungen in der Projektstruktur sind viele Bibliotheken nicht mehr unterstützt, sodass man hierbei neue alternativen suchen müsste. Hierfür wären neue Adapter notwendig.
Ein weiteres Problem der Anwendung ist, dass Sie nicht in Containern läuft, sondern die kompilierte Anwendung direkt von einem Apache Tomcat Server ausgeführt wird. Dies führt oft zu dem Problem, dass Updates an der Infrastruktur langwierig sind, weil man hier mit den Betreibern der Infrastruktur Rücksprache halten muss. Ein weiteres Problem hierbei ist, man schnell unterschiedliche Versionen auf unterschiedlichen Maschinen hat. So kann z.B. die Minor-Version der Datenbanksoftware auf einem Entwicklungsrechner, eine andere sein als die der Live-Umgebung, die wieder eine andere Version haben könnte als das Test-System.

Dadurch, dass die Anwendung eine monolithische ist, wird keine relevante Schnittstelle nach außen bereitgestellt, die von anderen Werkzeugen benutzt werden könnte. So ist es aktuell nicht möglich eine Mobile App, oder andere Applikationen, zu entwickeln, die Zugriff auf die persistierten Daten der Applikation hat.

2.3 Fazit

Die bestehende Applikation auf Grails Basis ist eine Funktionierende Anwendung, die ihre Aufgaben erledigt. Allerdings gibt es einige Probleme, das größte ist vermutlich die veraltete Version von Java und Grails. Durch die Obsoleszenz der verwendeten Software, wird das Problem der nicht benutzten Containern, wie z.B. Docker deutlicher. Während es durch Konfiguration in einem gut strukturierten Softwareprojekt, das Container benutzt, möglich ist, die Versionen der grundlegenden Software zu ändern, müssen im aktuellen Fall Betriebssystem Upgrades durchgeführt werden. Aufgrund dessen, ist es offensichtlich, dass die Software nicht in dem aktuellen Stand bleiben kann, in dem sie aktuell ist.
Es muss also eine Entscheidung getroffen werden wie man das Projekt weiterbehandelt. Während Grails [19] damit wirbt, dass es von großen Playern im Internet benutzt wird, wie z.B.: Google, Cisco, Sky und LinkedIn, gibt es bei Silpion sehr wenig Kompetenzen und Bedarf an Grails Kompetenzen, sodass eine Weiterentwicklung nur von sehr wenigen Leuten durchgeführt werden kann. Nach Abwägung der Probleme, unter Beachtung der Wünsche für eine Erweiterung der Software mit neuen Funktionen, wurde in zusammen mit der Geschäftsführung vereinbart eine Neuentwicklung mit Technologien zu starten, die eine bessere Verbreitung im Hause haben. Da diese Entwicklung in Zusammenhang mit dieser Bachelorarbeit zumindest teilweise durchgeführt wird, sind die Entwicklungskosten relativ gering.

3 Design

In diesem Kapitel wird ein Konzept für die Entwicklung der Silpion SkillDB aufgezeigt. Hierbei werden sowohl die funktionalen als auch die nicht funktionalen Anforderungen berücksichtigt. Im Gegensatz zu der alten monolithischen Applikation wird die neue Applikation die Benutzerschnittstelle (Frontend) und die Server (Backend) trennen. Die Trennung von Frontend und Backend wird im Design explizit weiter beibehalten.

Auch wenn es sich aktuell grundsätzlich bei der SkillDB um eine Firmeninterne Software handelt, deren Anzahl gleichzeitiger Benutzer nicht den 3-Stelligen Bereich in absehbarer Zeit überschreiten wird, sollte man bei der Konzeption immer so planen, dass die Applikation auch mit hohen Lasten auskommen kann. Denn in Zukunft ist es durchaus denkbar, dass die hier beschriebene Applikation sich in ein kommerziell relevantes Produkt entwickeln könnte, welches deutlich mehr gleichzeitige Nutzer
Da die Unabhängigkeit zwischen dem Client, mit dem User-Interface (Frontend), und dem Server (Backend), mit der eigentlichen Logik, durch die Anforderungen sehr wichtig ist, wird beim Design vollständig zwischen ihnen unterschieden.
Um die Kommunikation zwischen dem Frontend und dem Backend zu gewährleisten wird eine REST-API genutzt. Diese muss dabei gut ausgearbeitet werden, damit bei späteren änderungen, vorhandene Clients nicht unbrauchbar werden. Die REST-Schnittstelle wird dabei wie die Software inkrementell entwickelt.
Zu Beginn dieses Kapitels werden Entscheidungen, die das Design des Frontends und des Backends relevant sind beschrieben. Anschließend werden in den weiteren Sektionen relevanten Kriterien für Frontend und Backend vorgestellt.

3.1 Allgemein

Durch Verteilungsdiagramme, auf englisch auch deployment diagram, kann man eine Gesamtübersicht des Systems, inklusive Clients, Server und anderer Drittsysteme, aufzeigen. Hierbei

werden die jeweiligen Anwendungen konkreten physischen Maschinen zugeordnet. Ebenfalls werden im Verteilungsdiagramm die Kommunikationswege und abhängigkeiten dargestellt.

3.1.1 Verteilungssicht

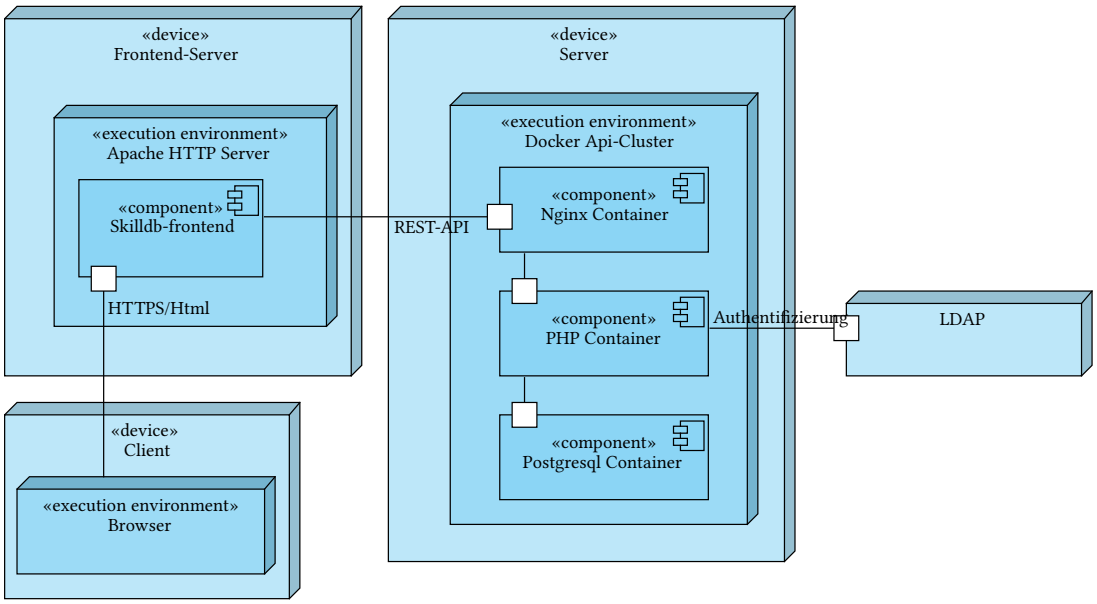

Abbildung 3.1: Verteilungssicht

Die Abbildung 3.1 zeigt eine Mögliche Verteilung der Software Applikation auf die verschiedenen physischen Maschinen. Es sind durchaus andere Konfigurationen denkbar, aber für den Entwicklungsprozess ist die Entscheidung auf diese Verteilung gefallen. Auf dem Backend-Server läuft Docker[15] welches die einzelnen Container der Backend Anwendung ausführt. Die Einzelnen Container sind hierbei einfach, ohne viele Änderungen, durch andere Austauschbar.

Es ist zwar durchaus möglich die einzelnen Container auf unterschiedlichen physischen Maschinen zu betreiben, jedoch ist dies ein Mehraufwand in der Konfiguration der Server. Denkbar wäre hierbei dass in Zukunft ein Kubernetes[28] eingesetzt wird. Dies würde ermöglichen abgestürzte Docker Container automatisch neu zu starten oder mehrere Container bei hoher Last automatisch zu starten. Allerdings handelt es sich im aktuellen Stand um eine Firmeninterne Software mit weniger als 300 Benutzern, sodass eine automatische Skalierung nicht notwendig

sein wird.

Der Backend Cluster besteht im einfachsten Falle aus einem PostgreSQL[40] Container, einem PHP[36] Container, sowie einem NGINX[32] Container. PostgreSQL ist eine kostenfreie objektrelationales Datenbankmanagementsystem die in der alten Version der SkillDB ebenfalls benutzt wurde. Da es zum einen einfacher ist die Daten innerhalb der selben Datenbanksoftware zu migrieren, ist dies ein Vorteil gegenüber anderen Datenbankmanagementsystem, die in der Firma Anwendung finden. Da die Authentifizierung über den Firmeneigenen LDAP-Server läuft, muss der PHP-Container in dem die Anwendung läuft, mit diesem kommunizieren. NGINX ist ein kostenfreier Webserver, der neben dem Apache Webserver zu den zwei weit verbreitetsten Servern gehört. Zudem wird NGINX von 42% der 1000 meistbesuchten Webseiten genutzt, und somit der meistbenutzte Webserver bei den Top-Playern im internet [51]. Der Webserver bietet die Möglichkeit die durch die Anwendung bereitgestellte REST-API nach Außen verfügbar.
Da das Frontend über die Bereitgestellte REST-Schnittstelle über HTTPS anhand von JSON-LD mit dem Backend kommuniziert, ist es egal auf welcher physischen Maschine es liegt. Das Frontend wird von einem Apache-Webserver[9] gehostet und ist somit getrennt von dem Backend. Dies ermöglicht es auch einfach, das Frontend zu Entwicklungszwecken, auf einem Entwickler-Rechner auszuführen, während man die Schnittstelle im Firmennetz konsumiert.
Der Client greift über einen aktuellen Browser, wie bei Internetseiten überlich über HTTPS auf das Frontend zu, und bekommt HTML, CSS und Javascript geliefert.

3.1.2 Client-Server Architektur

Wie auch bei der veralteten SkillDB ist dieses Projekt ebenfalls als Client-Server Architektur konzipiert. Dabei gibt es unterschiedliche unterschiedliche Aufteilungen der Verantwortlichkeiten. Diese unterschiedlichen Aufteilungen kann man im Bild 3.2 erkennen. Dort wird ein gradueller Unterschied zwischen Thin Clients und Fat Clients gemacht. Im Extremfall eines Thin Clients besteht der Client lediglich aus einen Teil der Benutzerschnittstelle, während der Server Teile der Benutzerschnittstelle beinhaltet sowie die gesamte Applikation und Datenbank. Der extremfall eines Fat Clients vereint die Benutzerschnittstelle, Applikationslogik und teileweise Datenpersistenz in sich, der Server ist hin diesem Falle lediglich für Datenpersistenz zuständig.
Bei der gewählten Architektur handelt es sich eine Mehrschichten Architektur (multitier architecture). Die Folgenden Schichten sind vorhanden:

- Präsentationsschicht
- Logikschicht
- Persistenzschicht

Die Präsentationsschicht spiegelt das Frontend wider, die Logikschicht wird durch die Backend-Software repräsentiert, und die Persistenzschicht wird durch die Datenbank erfüllt.

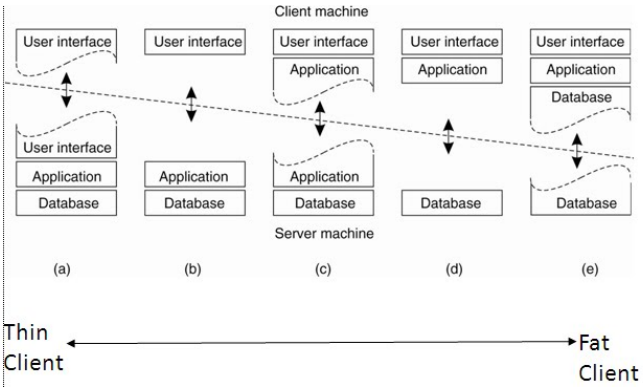

Abbildung 3.2: Thin-Fat Client Server Architektur

Die beiden Extremfälle a und e der Abbildung 3.2 sind in der gewählten Architektur offensichtlich nicht der Fall. Es handelt sich in der gewählten Architektur um eine Mischung aus Fall b und c. Während das Backend unabhängig des Frontends in sich schlüssig und funktional ist, und alle Use Case in sich umsetzt, befindet sich trotzdem auch ein redundanter Teil der Anwendung im Frontend. Dies hat damit zu tun, dass das Frontend, um die User Experience zu verbessern selbst bereits einige aufgaben vorher ausführt. Diese beinhalten Funktionalitäten bestehen hauptsächlich aus Validierungen und Authorisation, sodass Benutzer gar nicht erst ungültige Daten an das Backend schicken, oder versuchen Aktionen auszuführen, zu denen sie nicht berechtigt sind. Dadurch ist es möglich die Anzahl der Anfragen an den Server zu reduzieren, was transferierte Datenmengen, und Rechenzeit im Backend verringert, und die Applikation dadurch besser Skalierbar macht. Zwar lässt das Backend durch die Natur seiner Schnittstelle es zu, dass unautorisierte, oder fehlerhafte Anfragen absendet, allerdings müssen diese mit Entsprechenden Fehlermeldungen beantwortet werden. Für eine zukünftige, bisher nicht geplante, Native Mobile App wäre eine Verteilung wie im Abschnitt d genauso gut denkbar, wie die der in dieser Arbeit dargestellten Anwendung. Allerdings müsste für eine Anwendung des Typs d sichergestellt werden, das alle Operationen, korrekt autorisiert sind, und die Datenbank immer in einem validen zustand ist. Diese Methode bedeutet jedoch, dass Logik, wie z.B die Autorisierungslogik mehrfach gleich implementiert werden muss.

3.1.3 Kommunikationsschnittstelle

Das Backend ist so konzipiert, dass es durch eine REST-Schnittstelle kommuniziert. Representational state transfer (REST) ist ein Architekturstil für verteilte Applikationen. REST hat Resources im Fokus und folgt daher dem Prinzip, dass jede Resource durch eine eindeutige ID identifizierbar ist. Resources in diesem Zusammenhang können unterschiedlicher Form sein. Resources können unter anderem Binärdaten sein, wie z.B. Bilder, oder in einer der verschiedenen Textformen. Das Backend soll in der Lage sein mehrere Textrepräsentationen zu unterstützen, um es für zukünftige Applikationen einfacher zu machen, diese Schnittstelle zu konsumieren. Die Repräsentationen der Textdaten sind: JSON [26], JSON-LD [25], JSON-HAL [24] und XML [57]

Für das Backend wird ein Teil der HTTP/1.1 Methoden [21] benutzt um unterschiedliche Operationen auf den einzelnen Resourcen durchzuführen. In Tabelle 3.1 ist zu sehen welche HTTP Methode für welche Operationen genutzt werden. Die Kommunikation mit einer REST-

Methode	Anwendungsfall
GET	Wird benutzt um Resourcen zu lesen. Es handelt sich um eine reine Lese-Operation, daher werden keine Änderungen vorgenommen.
POST	Wird benutzt um neue Resourcen zu erstellen oder Formularinhalte zu übermitteln
DELETE	Wird benutzt um eine bestehende Resource zu löschen
PUT	Wird benutzt um eine bestehende Resource zu ersetzen

Tabelle 3.1: Erläuterung der HTTP Methoden

Schnittstelle soll zustandslos [44] sein, das beutet, dass jede einzelne Anfrage an die Schnittstelle alle notwendigen Informationen für die Anfrage enthält. Die Schnittstelle darf sich somit nicht auf Informationen von vorherigen Anfragen verlassen. Die Schnittstelle dürfte somit z.B. nicht eine PHP-Session erstellen, die es ihm erlaubt den Benutzer erneut zu identifizieren. Der Vorteil der Zustandslosigkeit ist die Skalierbarkeit. Dadurch das jeder einzelne Anfrage isoliert zu betrachten ist, ist es bei mehreren Instanzen egal auf Instanz die Anfrage ankommt. Jede Anfrage an den Server wird mit dem HTTPS Protokoll ausgeführt, um die Vertraulichkeit von Daten sicherzustellen. Jede Anfrage an die Schnittstelle, die verarbeitet werden kann, wird eine Antwort bekommen die einen HTTP/1.1 Statuscode [22] enthält, und je nach operation einen Inhalt. Statuscodes geben eine einfache Rückmeldung ob eine Operation erfolgreich war, oder

nicht. Im Folgenden wird eine - nicht vollständige - Übersicht über die einzelnen Methoden, und deren Antworten gegeben.

GET Anfragen

Wie bereits in Tabelle 3.1 beschrieben, werden GET Methoden benutzt um Informationen von der Schnittstelle zu erhalten. Es werden unter anderem GET-Anfragen von diesen Operationen geschickt:

- Abfrage aller bestehenden Skills
- Abfrage aller vorhandenen Skills für ein Profil
- Abfrage aller benutzten Skills für ein Profil in einem Projekt

Die Antworten sind unter anderem:

200 OK Ist eine GET Anfrage erfolgreich wird eine Antwort mit dem Status 200 zurückgegeben. Diese Antwort enthält die angefragten Resourcen

404 Not Found Angefragte Resource konnte nicht gefunden werden. Dies passiert entweder wenn eine ID verwendet wird die nicht existiert.

POST Anfragen

POST Anfragen werden im Allgemeinen benutzt um Resourcen zu erstellen. Beispielhaft folgen ein paar der POST-Anfragen

- Login Anfrage mit Benutzerdaten. Hier wird keine fachliche Resource erstellt, jedoch wird hier bei jedem Aufruf ein Autorisierungstoken erstellt.
- Neue Fertigkeit erstellen.
- Neues Fertigkeit einem Profil hinzufügen.
- Einen Abschluss einem Profil hinzufügen.

Die Antworten sind unter anderem:

201 Ceated Ist eine POST Anfrage erfolgreich wird eine Antwort mit dem Status 200 zurückgegeben. Diese Antwort enthält die neu erstellte Resource mit allen sichtbaren Feldern inklusive der ID

400 Bad Request Versucht ein User eine Resource mit invaliden Daten anzulegen, wird mit Status 400 geantwortet. Der Inhalt dieser Antwort ist ein Hinweis darauf welche daten nicht valide sind.

PUT Anfragen

PUT Anfragen werden benutzt um Resourcen zu ersetzen, im Gegensatz zum aktuell nicht benutzten PATCH [35]. Während bei einer PATCH Anfrage der lediglich der zu ändernde Teil eine Resource übermittelt wird, wird bei PUT die gesamte Resource mitgeschickt. Beispielhaft folgen ein paar der PUT-Anfragen

- Kontaktinformationen bearbeiten.
- Eine Fertigkeit bearbeiten.
- Eine Sprache Bearbeiten.
- Einen Abschluss eines Profils bearbeiten.

Die Antworten sind unter anderem:

200 OK Ist eine PUT Anfrage erfolgreich wird eine Antwort mit dem Status 200 zurückgegeben. Diese Antwort enthält die neue Resource mit allen sichtbaren Feldern inklusive der ID

400 Bad Request Versucht ein User eine Resource mit invaliden Daten zu bearbeiten, wird mit Status 400 geantwortet. Der Inhalt dieser Antwort ist ein Hinweis darauf welche daten nicht valide sind.

404 Not Found Angefragte Resource konnte nicht gefunden werden. Dies passiert entweder wenn eine ID verwendet wird die nicht existiert.

DELETE Anfragen

DELETE Anfragen werden benutzt um Resourcen zu löschen. Beispielhaft folgen ein paar der DELETE-Anfragen

- Ein Profil löschen.
- Eine Fertigkeit löschen.
- Eine Sprache aus einem Profil entfernen.
- Einen Abschluss eines Profils entfernen.

Bei Erfolg, enthalten die Antworten die angefragten Resourcen als Inhalt, und einen entsprechenden Statuscode. Bei Nichterfolg wird je nach Fehler ein anderer Inhalt übermittelt. Die Antworten sind unter anderem:

204 No Content Ist eine DELETE Anfrage erfolgreich wird eine Antwort mit dem Status 204 zurückgegeben. Diese Antwort enthält wie der Name andeutet, keinen Inhalt

404 Not Found Angefragte Resource konnte nicht gefunden werden. Dies passiert entweder wenn eine ID verwendet wird die nicht existiert.

Allgemeine Antworten

Jede Anfrage kann mit einem der folgenden Fehler beantwortet werden, deren Inhalt leer ist. Diese Antworten weisen darauf hin, dass der Client etwas falsch gemacht hat.

401 Unauthorized Versucht ein User eine Resource anzufragen ohne angemeldet zu sein, wird der Status 401 zurückgeben. Der Inhalt diese Antwort ist leer.

403 Forbidden Versucht ein angemeldeter User auf eine Resource zuzugreifen, auf die er keinen Zugriff hat, wird ein Status 403 zurückgegeben. Die Antwort ist leer.

404 Not Found Wenn eine Anfrage an das Backend an eine Route geschickt wird die nicht existiert, wird diese Antwort zurückgegeben.

Während alle bisher beschriebenen Antworten von Fehlerfreiem Verhalten ausgehen, kann es durchaus möglich sein, dass das Backend in einen Fehler läuft. Hier gibt es 2 häufige Fehler die durch unterschiedliche Statuscodes zu unterscheiden sind. Einer dieser Fehler ist, dass nicht behandelte Ausnahmen auftreten. Diese werden in der Regel durch den Statuscode 500 symbolisiert. Ursachen dafür könnten sein, dass der Datenbankserver nicht zu erreichen ist, der LDAP Server nicht zu erreichen ist, oder ein Programmfehler auftritt. Der andere Fehlerfall ist ein Timeout, Dieser wird durch den Fehlercode 504 symbolisiert. Dieser Fehler tritt auf wenn die Beantwortung einer Anfrage zu lange dauert, dies tritt auf wenn die Verarbeitung einer Anfrage sehr aufwändig ist, oder der Aufruf eines Drittsystems zu lange dauert.

3.1.4 Sicherheit

Da Benutzer im in dieser Applikation sensible Daten pflegen sollen, ist es essenziell die Schnittstelle abzusichern. Da nicht alle Benutzer die gleichen Privilegien haben, ist es ebenfalls essenziell zwischen diesen Benutzern zu unterscheiden. Um die weiter oben genannte Zu-

standslosigkeit der REST-Schnittstelle zu gewährleisten, darf ebenfalls keine Session für die Unterscheidung einzelner Nutzer benutzt werden. Eine typischer zustandslose Absicherung einer REST-Schnittstelle ist ein sogenannter API-Key, also ein Passwort, das nur berechtigen Personen bekannt ist, und bei jeder Anfrage mitgeschickt wird. Ein einzelner API-Key ermöglicht es zwar die Schnittstelle Zustandslos zu halten, jedoch nicht die Unterscheidung unterschiedlicher Rollen. Um die unterscheidung von Nutzern sinnvoll gewährleisten zu können ist es notwendig, dass mindestens jede Benutzergruppe andere Zugangsdaten hat. Allerdings ist das Teilen von Zugangsdaten für mehrere Benutzer der gleichen Rolle nicht nur organisatorisch problematisch, sondern auch sicherheitstechnisch bedenklich, falls z.B. einmal ein Mitarbeiter aus der Firma ausscheidet. Die Sicherheit und die Praktikabilität kann man erhöhen indem man jeden einzelnen Benutzerin einzelne Zugangsdaten zuordnet, die die Benutzerin selber verwaltet. Eine Unterscheidung der einzelnen Benutzer im gegensatz zu einzelnen Rollen hat den Vorteil, dass man einfach ausgeschiedene Mitarbeiterinnen sperren kann ohne ein bestehendes Passwort für andere Benutzerinnen zu ändern. Außerdem ist es einfach einer Mitarbeiterin andere Privilegien zuzuweisen, ohne neue Zugangsdaten zu verteilen. Während mehrere API-Keys für unterschiedliche Rollen machbar wären, da die Anzahl der Rollen und somit der API-Keys überschaubar sind, ist es sehr aufwändig API-Keys für jeden Benutzer zu verwalten. Daher ist es notwendig dass benutzer ihre eigenen Anmeldedaten selbst verwalten können.
Hier gibt es mehrere Möglichkeiten dies zu bewerkstelligen. Eine Möglichkeit ist, eine Benutzerdatenbank zu Benutzernamen und Passwörtern zu implementieren, gegen die sich authentifiziert wird. Während dies in den meisten Fällen bedenkenlos geschehen kann, würde es im Silpion IT Solutions GmbH Umfeld bedeuten, dass alle Mitarbeiter sich weitere Benutzerdaten merken müssten, die ebenfalls regelmäßig aktualisiert werden sollten. Daher bietet es sich an, die im Firmenumfeld benutzten Benutzerdaten zu nutzen, die jeder Mitarbeiter pflegen muss, um z.B. E-Mails zu empfangen. Eine denkbare Methode hierfür wäre OAuth2.0 [49], welches es ermöglicht, dass keine Benutzerdaten an die Schnittstelle gesandt werden müssen, und man sich direkt bei dem OAuth Server autorisiert. OAuth2.0 ist jedoch leider noch nicht bei Silpion umgesetzt, sodass diese Lösung ausfällt jedoch für die Zukunft denkbar wäre. Als sinnvolle Möglichkeit die bestehenden Benutzerdaten zu benutzen besteht die Möglichkeit sich gegen den Firmeninternen LDAP [29]-Service zu authentifizieren.
Bei diesem Vorgehen wird wie in Abbildung 3.3 zu sehen, eine Login-Anfrage an das Backend mit den Zugangsdaten des Benutzers geschickt. Die Schnittstelle wird als erstes versuchen vom LDAP Service den Benutzer abzuholen, falls dieser existiert. Wenn dieser existiert und die Schnittstelle ein Benutzerobjekt erhalten hat, wird sie den Service prüfen lassen, ob die

Zugangsdaten korrekt sind. Diese Zugangsdaten leitet die Schnittstelle an den LDAP Service weiter, um deren Korrektheit zu prüfen. Falls die Benutzerdaten korrekt sind wird ein JSON Web Token [27] an das Frontend zurück geschickt, welches ab dann bei jeder Anfrage mitgeschickt wird.

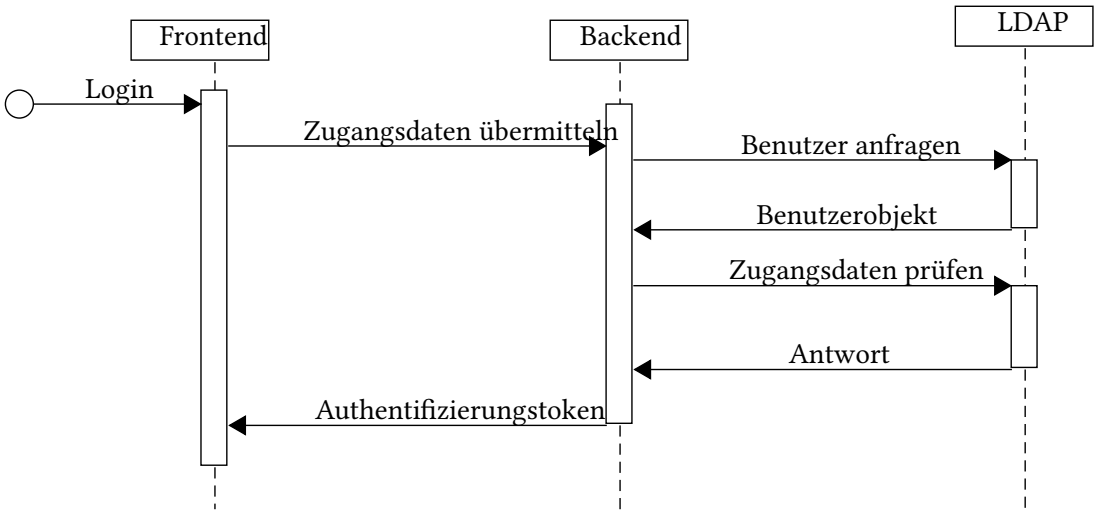

Abbildung 3.3: LDAP Authentifizierung Sequenzdiagramm

Ein JSON Web Token (JWT) ist ein standardisiertes Accesstoken das Informationen, sogenannte claims, über den Nutzer enthält. Ein JWT besteht immer aus drei Teilen: Dem Header, der Payload und der Signatur. Der Header besteht üblicherweise aus 2 Teilen dem Algorithmus, und dem Typ. Der Typ ist bei JWT immer JWT und der Algorithmus gibt an womit das Token Signiert ist, in der SkillDB ist es RS256, also eine RSA Signatur mit der SHA-256 Hashfunktion. Die Payload ist der Teil der die claims, also die Benutzerinformationen, enthält. Diese Informationen sind in der SkillDB die folgenden:

iat Das Erstellungsdatum des Tokens

exp Das Ablaufdatum des Tokens

roles Die Rollen des Nutzers

username Den Benutzernamen

firstname Den Vornamen des Nutzers

lastname Den Nachnamen des Nutzers

Der Inhalt der Signatur ist die Kombination des Base64 kodierten Headers, und der Base64 kodierten Payload. Das Token ist die Konkatenation des Base64 kodierten Headers, eines Punktes, der Base64 kodierten Payload, eines Punktes und der Base64 kodieren Signatur.

```
1    const token = base64(header) + '.' + base64(payload)
2                + '.' + base64(signature)
```

Auflistung 3.1: Aufbau eines JWT

So wird das folgende JWT in 3.2 zu den Daten in 3.3 dekodiert. An 3.3 lässt sich leicht erkennen, dass die Daten einen Benutzer widerspiegeln. Diese Informationen können von jedem Client und von dem Backend benutzt werden. Da die Informationen des JWT nur Base64 kodiert sind, ist es einfach möglich die mitgelieferten Informationen zu nutzen. Obwohl es theoretisch einfach möglich ist, die Informationen des JWT durch einen böswilligen Angreifer zu verändern, wird das Backend ein verändertes Token verweigern, da die Signatur des Tokens durch die Änderung nicht mehr gültig ist.

```
1  eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJpYXQiOjE1ODQ5NjE4NjksIm
2  V4cCI6MTk0NDk2MTg2OSwicm9sZXMiOlsiUk9MRV9DT05URU5UX01BTkFHRU1FT
3  lQiLCJST0xFX1VTRVIiLCJST0xFX1NFQVJDSCIsIlJPTEVfU0xQTiJdLCJ1c2Vy
4  bmFtZSI6ImFkbWluIiwicGFzc3dvcmQiOm51bGwsImZpcnN0bmFtZSI6IkhhbnM
5  iLCJsYXN0bmFtZSI6Ild1cnN0aWdlciJ9.Ss40wy054xaU7--GgOrTG1sr7I_qp
6  aOqgZdLO81VfnA8W2xgjDCpU2_IUck3LNavcPMXC8wOnaF82PRAboAkQmMQ-ut9
7  4ohXEen_-FpLVhYzZCergwqvVeCUwm_PvBq967sj0I1Gh_kyIvgUEeaZGuHaVvz
8  kZbRuDkiepVa8vLsfJZUqlp6vPvQKPfHuN1NCB3CSU-J-ljh7hY_uatlymB_11f
9  Y6KRAJeG0h5zHrgqxqXG27NwfMJ9DbEgj-UbinQSgotjxAFzbrgVPFoQNAvIhkT
10 -srLIapwv-OmuLzB2vFVgXOFiND39eipUzyazF1gBz89YKmV7c7I4asEugOpK9H
11 U4ZXV8HNZsTn4tTxQMgfRxtuYE79X2KQc2K-HnCrqTSoPDTEMVWG-jjuZCdgDXe
12 ovX-tMp3FvS36X1gQOZmJQOr5In3zsKxWmCP7y6mBNmqyPhgrrG9Cm9Ow3kXqqM
13 3wvaQKMiPpkZ_P1xg1Fzgvgj1hRbtlCwFkjaxyni5VZJqLgBGY_eU1al6IasLjZ
14 2Ud1ZasTElYHi7kTsnTKFoPSLu3rewfezfk1s0NkoifRtLgWLBqdU6G-a54f4Dm
15 XF-mA2lTlVp-QY2cIV9uGIOz7w9rf0NJj9hEEqOR_dDGbGgDZgqV4rnNIm8U6QE
16 bx4Z30VEhuLVFstEHv7wewXY
```

Auflistung 3.2: Beispiel JWT

So folgendermaßen interpretiert

```
1  {
2      "iat": 1584961869,
3      "exp": 1944961869,
4      "roles": [
5          "ROLE_CONTENT_MANAGEMENT",
```

```
6             "ROLE_USER",
7             "ROLE_SEARCH",
8             "ROLE_SLPN"
9         ],
10        "username": "admin",
11        "firstname": "Hans",
12        "lastname": "Wurstiger"
13    }
```

Auflistung 3.3: Dekodiertes beispiel JWT

Durch den Fakt, dass Zugangsdaten an die Schnittstelle geschickt wird, ist es notwendig dass die Kommunikation zwischen allen Services verschlüsselt wird. Daher ist es notwendig dass das Frontend nur per HTTPS erreichbar ist. Das Frontend darf ebenfalls nur über HTTPS mit der Schnittstelle kommunizieren. Die Schnittstelle kommuniziert durch LDAPS [56]. LDAPS ist das nicht standardisierte, aber weit verbreitete LDAP über SSL.

3.2 Backend

Das Backend stellt alle notwendigen Funktionen über eine gesicherte REST-Schnittstelle bereit. Dabei ist es für die gesamte Datenhaltung zuständig und stellt außerdem sicher, dass alle Daten in einem validen Zustand sind. Da das Backend die gesamte Logik zur Validierung und Autorisierung besitzt, ist er unabhängig von jedem Client, und braucht kein Wissen über Konsumenten der Schnittstelle. Die REST-Schnittstelle wird in der Sprache PHP entwickelt. PHP ist ein rekursives Akronym und steht für "PHP Hypertext Preprocessor-[39]. PHP ist eine Open-Source [38] Skriptsprache, die einen Marktanteil von etwa 41% in den Alexa Top 1M hat [37]. PHP befindet sich aktuell in Version 7.4 und erlaubt es objektorientiert und statisch typisiert zu programmieren. Auch wenn PHP eine Skriptsprache ist, die nicht compiliert wird, ist es mithilfe moderner IDEs einfach möglich Syntaxfehler, und Typfehler festzustellen, sodass diese nicht während der Laufzeit auftreten.

3.2.1 Symfony 5

Als Grundlage für das REST-Schnittstelle wird, das PHP Framework Symfony in der Major Version 5 Benutzt. Symfony hat sich als Framework in dem Silpion PHP Umfeld durchgesetzt, wodurch hier einige Kompetenzen vorhanden sind. Bei Symfony handelt es sich um ein Open Source Framework, welches in mehrere Komponenten unterteilt ist. Dies ermöglicht es, ein

Projekt schlank zu halten, da man nur die Komponenten nutzt, die man wirklich benötigt. Das Basispaket des Frameworks wurde bisher über 48 Millionen mal heruntergeladen [48] und hat bei mehr Github 23.000 Sterne. Das einzige PHP-Framework welches mehr Sterne auf Github hat, ist lediglich Laravel.

Symfony zeichnet sich dadurch aus, dass es als Framework nutzbar ist, jedoch auch als einzelne unabhängige Komponenten. Zusätzlich ist es möglich sogenannte Bundles einzubinden, die zusätzliche Funktionalitäten bereitstellen. Wichtige Komponenten die in diesem Projekt als Teil des Frameworks benutzt werden, sind unter anderem: Dependency Injection, Routing, Serializer und ORM. Dependency Injection ist ein Design Pattern, welches Abhängigkeiten an jeweilige Objekte übergibt, und nicht im Objekt selber erzeugt. Dies ermöglicht es, einfach die Implementierung einer Abhängigkeit einfach auszutauschen, z.B. gegen Mock-Objekte um Unit-Tests zu vereinfachen. Der Router ermöglicht es, eine oder mehrere URLs einer sogenannten Methode zuzuordnen, die die Anfrage erhält, und daraus eine Antwort generiert. Der Serializer ist eine Komponente, die Objekte in ein menschen lesbares Format bringen kann, und umgekehrt, das Format ist dabei leicht austauschbar. Diese komponente ist dafür Zuständig, dass Profile als JSON, XML, oder JSON-LD über die Schnittstelle ausgegeben werden und empfangen werden. ORM ist die Kurzform von Object-Relational Mapping, also ein Objekt Relationales Abbildungssystem, und ist somit eine Abstraktionsschicht zur Datenbank. Für diesen Zweck wird bei Symfony üblicherweise Doctrine benutzt. 3.2.1

Symfony empfiehlt eine Struktur, die es ermöglicht Code möglichst oft wieder zu verwerten, und zu kapseln. Symfony ist ein Framework welches es ermöglicht einem Mehr-Schichten Design Pattern zu folgen. Im Falle der Schnittstelle handelt es sich um ein Drei-Schichten Model, das Action Domain Responder [2] Design Pattern. Im MVC Pattern ist eine Applikation in die drei Schichten Action, Domain und Responder unterteilt. Die Action ist der Teil der applikation, der die Domain und den Responder aufruft. Die Domain beinhaltet die Domänenlogik, also alle Geschäftslogik inklusive der Persistenz. Der Responder nutzt die Informationen die er von der Action bekommt, und erstellt daraus eine HTTP Antwort.

Eine HTTP Anfrage wird von einem Web-Handler, in unserem Falle dem Router, an die Action weitergeleitet.Mit dem Workflow in 3.4 Die Action extrahiert die Informationen aus der Anfrage, die notwendig sind um diese zu beantworten. Diese Informationen werden nun an die Domäne weitergeleitet, sodass die Domäne daraus die Informationen für die Antwort

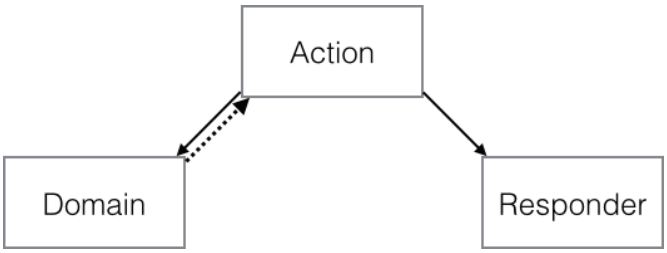

Abbildung 3.4: Action Domain Responder Workflow nach Paul M. Jones [2].

generieren kann. Die Informationen der Antwort werden von der Domäne an die Action zurückgegeben, und dann von der Action an den Responder weiter geleitet, welcher die Anfrage dann mit den übermittelten Informationen eine HTTP Antwort erstellt und diese übermittelt.

Das wichtigste Bundle, welches für dieses Projekt benutzt wird, ist API-Platform [11], welches es ermöglicht eine API weites gehend durch Konfiguration zu erstellen. So ist es in den meisten fällen nicht nötig eine Action oder den Responder zu implementieren, lediglich Teile der Domäne werden hiermit benötigt.

API Platform

API Platform wurde 2015 in der Version 1.0.0 veröffentlicht, und ist seitdem bestandteil des Symfony Ökosystems. Die Aktuellste Version ist die Version v2.5.7 und hat mittlerweile über 200 Mitwirkende. Api Platform wird von einigen großen Firmen benutzt, darunter General Electric, Renault und BNP Paribas.
Das Bundle übernimmt einen Großteil der Arbeit, um eine funktionale API zu erstellen. Es ermöglicht nicht nur REST-APIs zu erstellen, sondern ermöglicht es auch GraphQL anbindungen möglich zu machen. In diesem Projekt wird jedoch aktuell keine GraphQL Anbindung umgesetzt. Außerdem ermöglicht API Platform durch unterschiedliche Serializer und Deserializer unterschiedliche Formate in einer REST-API anzubieten. 3.1.3

Doctrine

Doctrine [16] ist eine Sammlung von OpenSource Projekten, welche seit 2006 besteht. Sie enthält unter anderem ein ORM, also ein objektrelationales Abbildungssystem, und ein DBAL ein Database Abstraction Layer, also eine Datenbankabstraktionsschicht.

Das objektrelationale Abbildungssystem wird genutzt und relationale Daten aus einer relationalen Datenbank in Objekte zu transformieren und objekte in Daten eines relationalen Datenbanksystems.
Die Datenbankabstraktionsschicht standardisiert die Kommunikation mit der Datenbank. Dadurch wird es ermöglicht, die Datenbank auszutauschen, ohne Änderungen an dem Programm vorzunehmen. Mithilfe des DBAL schreibt man keine SQL befehle mehr, sondern man nutzt die abstraktere Syntax der DQL [17]. Die Doctrine Query Language ist im gegensatz zu SQL eine Abfragesprache, die auf Objekte ausgelegt ist und keine Tabellen oder Spalten kennt. DQL benutzt lediglich Objekte und deren Felder, um eine Abstraktion von der tatsächlichen Datenbank und der Datenbankstruktur zu bieten.
Das DBAL ermöglicht es außerdem ein Datenbankschema zu erzeugen oder zu aktualisieren, die Datenbankstruktur wird hierfür über eine konfiguration definiert. Die Konfiguration kann in verschiedenen Formaten vorliegen, darunter Annotationen, Yaml, XML oder PHP-Code. Für dieses Projekt wurden Annotationen genutzt, die es ermöglichen mit wenig Aufwand die Persistenz von Objekten zu definieren.

3.2.2 Komponenten

Die Applikation lässt sich im Backend unter anderem in die Folgenden Bereiche einteilen: Controller, Services Repositories, Entitäten und Voter. Im Folgenden sollen die einzelnen Komponenten Typen erläutern werden. Dabei werden jeweils ein paar Beispiele aufgeführt, mit den jeweiligen Aufgaben die diese erfüllen, um ein plastisches Bild zu liefern.

Controller

Controller sind wie der Name erahnen lässt die Steuerelemente der Applikation. Sie sind die Schnittstelle, bzw. Fassade nach außen. Sie haben eine oder mehrere Actions 3.4. Eine Action ist im Falle der Applikation eine Operation auf der API. z.B. das Abfragen eines Profils. Controller kommunizieren mit der Domäne um anschließend an den Responder die notwendigen Daten weitergeben zu können.
In der Applikation ist es, dank API Platform für die meisten Fälle nicht notwendig Controller zu erstellen. Für einfache CRUD Operationen, also Create, Read, Update, Delete, stellt API Plattform die Controller bereit. Es müssen lediglich Controller erstellt werden, die komplexere Logik enthalten.

ProfileImageController Dieser Controller liest Bild-Binärdaten aus der Datenbank und liefert sie Base65 kodiert aus. Außerdem ist dieser Controller dafür zuständig, Bilder hinzuzufügen sowie zu löschen.

ExportController Der ExportController erstellt über die Domäne, eine Word oder PDF Datei die man als Portfolio an Kunden verschicken kann und liefert diese aus.

SearchController Der SearchController hat die Aufgabe suchen zu realisieren. Hier werden Profile anhand Ihrer Fähigkeiten und Namen durchsucht.

SecurityController Dieser Controller enthält den Endpunkt der für die Authentifizierung benutzt wird. Hier wird also der LDAP Server angesprochen

SkillDeleteController Das globale Löschen von Fähigkeiten wird hier realisiert, da diese Auswirkungen auf Profile haben können.

UserDeleteController Hier ist die Schnittstelle enthalten, die das Löschen eines Benutzers auslöst. Dabei wird auch das zugehörige Profil gelöscht.

Services

Services bilden die Geschäftslogik einer Applikation ab und werden in der Regel von Controllern oder anderen Services aufgerufen. Services können mit anderen Anwendungen kommunizieren, das Persistieren von Daten veranlassen. Sie beinhalten alle notwendigen Berechnungen und sind nur innerhalb der Applikation erreichbar und somit vor direkt äußerem Zugriff geschützt.

ExportService Um einen Profilexport zu erstellen wird dieser Service genutzt.

LdapService Dieser Service ist verantwortlich für die Kommunikation mit einem LDAP Server. Die Konfiguration, mit welchem Server gesprochen wird, wird über Dependency Injection an den Service übergeben.

LdapAuthenticator Dieser Service nutzt den LdapService um Authentifizierungsmechanismen bereitzustellen.

ProfileService Der ProfileService stellt Operationen bereit, die zum Löschen eines Profils notwendig sind

Repositories

Repositories sind Komponente die lesende Datenbankoperationen abstrahieren. Sie bieten Methoden an, die das Suchen nach Objekten vereinfachen. Die Komplexität eines Repositories

kann sehr stark variieren, während einfache Repositories praktisch keine Funktionen selber implementieren, ist es auch üblich Abfragen in DQL [17] zu formulieren.

Entities

Entitäten sind Objekte die Objekte widerspiegeln, die in der Datenbank gespeichert werden. Sie repräsentieren die unter andrem Profile, Fähigkeiten, Nutzer, Rollen. Viele dieser Entitäten sind ebenfalls die Basis für die Endpunkte und Operationen der REST-API. Durch die Benutzung von API Platform wird hier durch Annotationen ein großer Teil der API Definiert. Sie enthalten ebenfalls über Annotationen die Konfiguration Doctrine Konfiguration, sodass sie persistiert werden können (siehe Kapitel 4.1.3).

Voter

Voter sind Klassen, die durch Eventlistener Aufgerufen werden. In der applikation kann an verschiedenen Stellen abgefragt werden, ob ein Benutzer ein Objekt ansehen oder manipulieren darf. Diese Voter sind einfache Klassen die für ein Gegebenes Subjekt entscheiden ob eine Erlaubnis vorliegt.
Dabei wird als Erstes entschieden ob ein Voter für ein gegebenes Objekt abstimmen darf indem geprüft wird, ob der Voter das Subjekt der Abstimmung, sowie die Operation der Abstimmung unterstützt. Falls der Voter feststellt, dass er an der Abstimmung Teilnehmen kann, nutzt er seine implementierte Logik um zu entscheiden, ob ein gegebener Nutzer auf einem gegebenem Subjekt eine bestimmte Operation ausführen darf. Sollte der Voter entscheiden, dass er nicht an der Abstimmung Teilnehmen kann, enthält er sich einfach.
Bei Votern ist es durchaus möglich, dass mehrere Voter gleichzeitig abstimmen. Daher muss man festlegen welche Strategie benutzt wird um zu einem abschließendem Ergebnis zu kommen. Hierbei gibt es Vier strategien die das System von vorneherein unterstützt:

- **affirmative** Bei dieser Strategie reicht es aus, wenn ein einzelner Voter positiv abstimmt.
- **consensus** Bei der Konsens Strategie wird eine positive mehrheit benötigt
- **unanimous** Diese Strategy erfordert, dass alle Voter die abstimmen ein positives Ergebnis liefern
- **priority** Diese Strategie ist ähnlich der affirmative strategie, jedoch werden hier Voter priorisiert und stimmen in einer determinierten Reihenfolge ab. Hier reicht es aus, wenn der erste Voter der sich nicht enthält positiv abstimmt.

Zusätzlich ist es jedoch möglich eigene Strategien zu implementieren, falls die bisherigen nicht reichen. Üblicherweise werden diese Voter am Anfang eine Action, also einer Operation, ausgeführt. Wenn, basierend auf der ausgewählten Strategie, zu einem negativen Ergebnis gekommen wird, erhält ein Client eine Antwort mit einer 401 oder 403 Antwort 3.1.3.

ProfileVoter Der ProfileVoter entscheidet ob ein Benutzer ein Profil ansehen, bearbeiten, löschen oder erstellen kann.

ProjectVoter Dieser Voter stimmt bei Projekten darüber ab, ob ein gegebener Benutzer ein Projekt bearbeiten, ansehen, bearbeiten, löschen oder erstellen kann.

SkillVoter Wenn Operationen auf Fähigkeiten passieren, wird der SkillVoter abstimmen und entscheiden ob ein User diese Ausführen darf.

3.2.3 Sequenzdiagramme

Sequenzdiagramme modellieren einen konkreten Ablauf eines gegebenen Szenarios. Hierfür werden real vorhandene Operationen und die Nachrichten die sie erhalten oder versenden genutzt. Dies wird in der zeitlich korrekten Reihenfolge visualisiert. Dadurch ist es einfach möglich das Zusammenspiel einzelner Objekte zu demonstrieren.
Im Folgenden werden daher ein paar Operationen dargestellt, um einen Überblick in die grobe Struktur der Applikation zu bekommen. Der interne Ablauf des Clients wird hierbei jedoch nicht aufgezeigt, da der konkrete Ablauf von Client zu Client unterschiedlich sein kann.

Suche nach Profilen

Entwicklerin Alice sucht nach Profilen die die Fähigkeit *git* haben oder deren Name *git* enthält. In Abbildung 3.5 ist der Ablauf der Suche nach Profilen dargestellt.

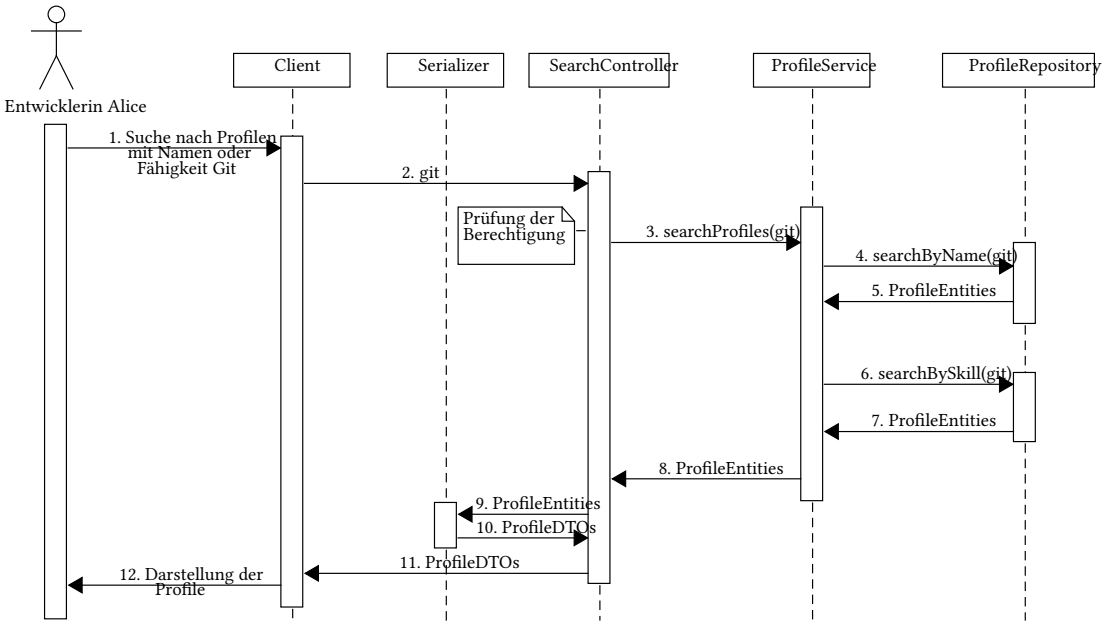

Abbildung 3.5: Profilsuche Sequenzdiagramm

1. Die Nutzerin gibt den Suchbegriff in den Client ein.
2. Der Client schickt eine Nachricht mit den Daten an den API Server.
3. Der Controller überprüft ob die eingabe Valide ist, und ob der Benutzer eine Suche ausführen darf. Wenn der Suchbegriff eine festgelegte Mindestlänge besitzt, und der Benutzer eine Suche ausführen darf, wird der ProfileService mit dem Suchbegriff aufgerufen.
4. Der ProfileService wird das ProfileRepository nach Profilen die den Suchbegriff im Namen enthalten befragen.
5. Das Repository sucht in der Datenbank nach entsprechenden Profilen und schickt alle gefunden Ergebnisse an den Service zurück.
6. Der Profilservice wird nun das Repository nach Profilen befragen, die eine Fähigkeit besitzen, die den Suchbegriff im Namen enthält.
7. Das ProfileRepository sucht auf der Datenbank und gibt gefundene Profile zurück.
8. Der ProfileService Aggregiert die beiden Ergebnisse und leitet diese an den Controller weiter
9. Der SearchController leitet die Ergebnisse an den Serializer weiter.

10. Der Serializer wandelt die Objekte in DTOs um und erstellt daraus eine Repräsentation die dem angefragten Format entsprechen. Im zusammenhang dieser Bachelorarbeit und dem Client der in diesem Zuge entwickelt wird, handelt es sich dabei um JSON-LD
11. Der Controller schickt nun eine antwort mit dem Inhalt aus dem Serializer an den Client.
12. Der Client stellt die Ergebnisse für die Benutzerin dar.

Erstellen einer Fertigkeitskategorie

Abteilungsleiterin Alice möchte die neue Fertigkeitskategorie CVS anlegen, damit sie später dort Fertigkeiten zuordnen kann. In Abbildung 3.6 ist der Ablauf zu sehen.

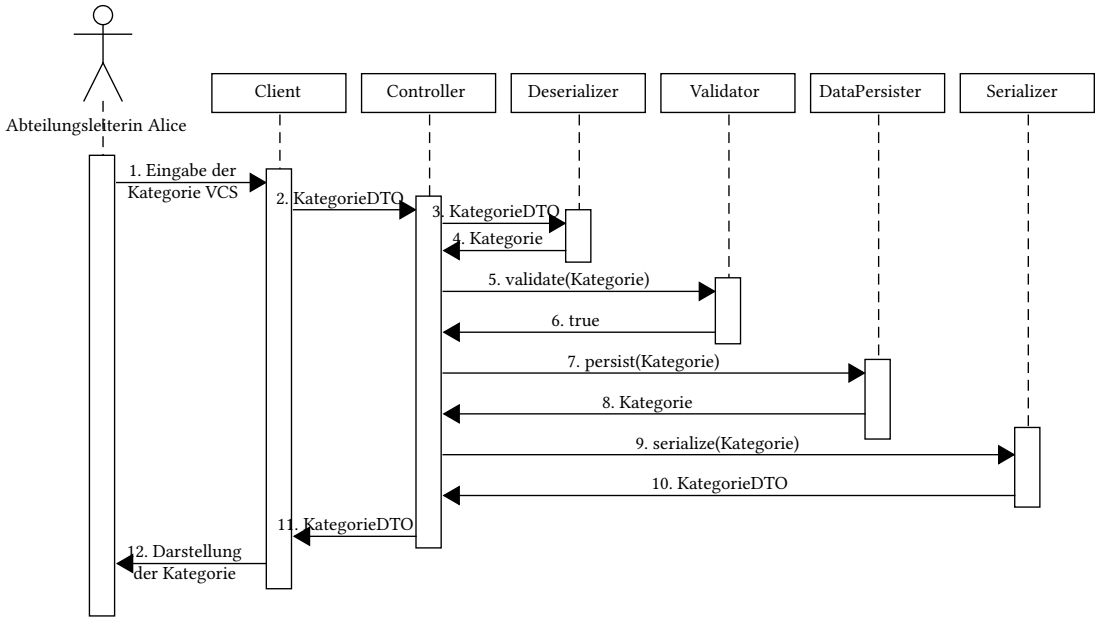

Abbildung 3.6: Erstellen eine Fertigkeitskategorie Sequenzdiagramm

1. Abteilungsleiterin Alice gibt die Daten für eine neue Fertigkeitskategorie an.
2. Der Client schickt ein KategorieDTO om Body eines POST-Requests an den Server.
3. Der Controller sendet das KategorieDTO an den Deserializer, nachdem geprüft wurde ob der Benutzer diese Operation ausführen darf.
4. Der Deserializer erstellt aus dem DTO ein Objekt der Konkreten Klasse und gibt es an den Controller zurück.

5. Der Kontroller übergibt das Objekt an den Validator
6. Der Validator prüft ob die Daten in dem Objekt korrekt sind und gibt bei Erfolg true zurück. Was benötigt wird, um ein Objekt als Valide zu erkennen wird über Constraints definiert. Im Falle dieser Applikation, erfolgt dies über Annotationen an der Entität.
7. Der Controller gibt das valide Objekt an der DataPersister weiter um es zu persistieren.
8. Der DataPersister persistiert die Daten in der Datenbank und gibt anschließend das Objekt zurück. Dies kann neue Werte enthalten, z.B. eine ID, falls diese automatisch generiert wird.
9. Der Controller gibt das komplette Objekt nun an den Serializer
10. Der Serializer wandelt das Objekt in ein DTO um und dieses in eine Repräsentation die dem angefragten Format entspricht.
11. Der Controller erstellt eine Antwort die das DTO enthält und leitet diese an den Client weiter
12. Der Client stellt die Daten für den Benutzer dar.

Erster Login eines Mitarbeiters

Der neue Mitarbeiter Bob loggt sich zum ersten mal ein, dabei wird ein neuer Benutzer und ein neues Profil in der Datenbank angelegt. In Abbildung 3.7 wird der Ablauf dafür dargestellt. Die detaillierte Kommunikation, zwischen LDAP-Server und Applikation wird dabei außen vor gelassen.

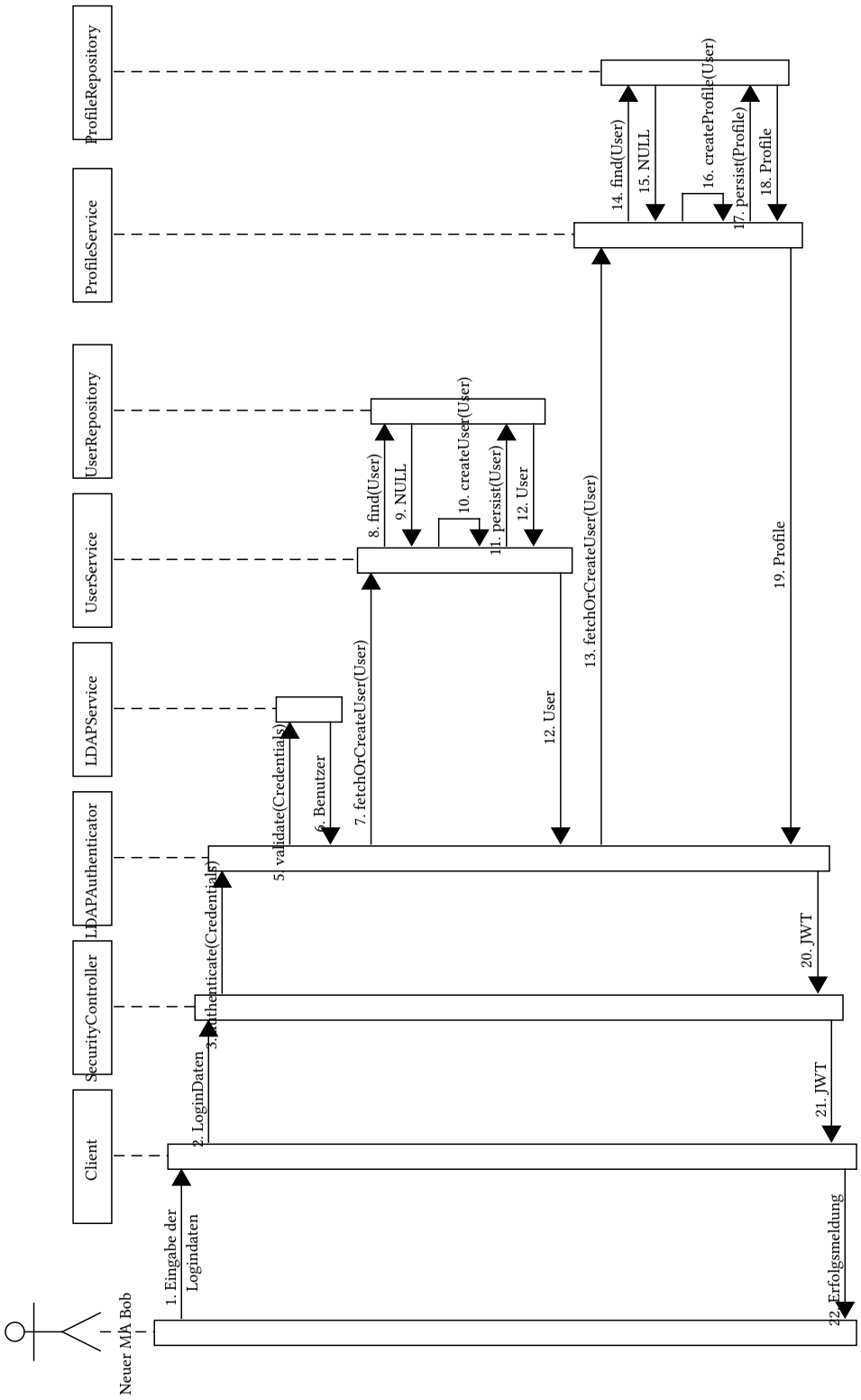

Abbildung 3.7: Erster Login eines Mitarbeiters

1. Der Benutzer gibt seine Logindaten in die Eingabemaske ein.
2. Der Client sendet die Logindaten an den Security Controller.
3. Der SecurityController gibt diese Daten weiter an den LDAPAuthenticator.
4. Dieser Authenticator gibt diese Daten an den LDAPService weiter, damit er diese validiert.
5. Der Service gibt ein Benutzerobjekt mit den Daten des Nutzers zurück, da die Logindaten korrekt sind.
6. Der Authenticator fragt nun den UserService nach einem Benutzerobjekt, das in der Datenbank gespeichert ist.
7. Der Service befragt das Repository dem Benutzer
8. Das Repository kann den Benutzer nicht finden, da er vorher noch nie mit dem Tool interagiert hat, und gibt somit ein leeres Ergebnis zurück.
9. Der UserService wird nun einen neuen Benutzer erstellen, der die Daten aus dem LDAP enthält.
10. Der Service veranlässt das UserRepository nun, den Benutzer zu persistieren.
11. Das Repository beantwortet dies mit dem einem Objekt das den Persistierten Benutzer entspricht.
12. Der UserService gibt diesen Benutzer nun an den LDAPAuthenticator zurück.
13. Der Authenticator fragt nun den ProfileService nach einem Profilobjekt, das in der Datenbank gespeichert ist.
14. Der Profilservice lässt nun das ProfilRepository nach einem Profil suchen, das zu diesem User gehört.
15. Das Repository gibt eine leere Antwort zurück, da der Benutzer noch nie mit dem Tool interagiert hat.
16. Der Service erstellt ein neues Profil mit den Daten des Benutzers
17. Der Service reicht das neu erstellte Profil dem Repository zum Persistieren weiter.
18. Nach dem Persistieren wird das Repository ein Profilobjekt zurückgeben, dass dem persistierten Objekt entspricht.
19. Der ProfileService gibt dieses Objekt an den Authenticator zurück.
20. Der Authenticator gibt ein JWT an den SecurityController weiter

21. Das Token wird nun vom Security Controller an den Client weitergegeben.
22. Der Client zeigt dem Benutzer eine Erfolgsmeldung an.

3.3 Client

Im Umfang dieser Bachelorarbeit werden teile eines Clients erstellt. Dieser Client ist die Benutzerschnittstelle für das Backend, 3.2 daher nutzt er die Schnittstelle die die REST-API definiert hat. Der Client ist verantwortlich für die Darstellung des Daten und Dienste, dabei wird der Client vorsorglich die Eingabedaten validieren. Dies bedeutet, dass der Client nicht nur ein einfache Darstellung ist, sondern ebenfalls einen Teil der Geschäftslogik enthalten muss.
Der Client der Teil dieser Arbeit ist, ist lediglich eine Beispielhafte Umsetzung, andere Clients können gleiche Funktionen grundlegend anders umsetzen.

3.3.1 Komponentenbasierte Architektur

Bei der Umsetzung einer modernen dynamischen Webseite hat sich die Komponentenbasierte Architektur durchgesetzt. Webkomponenten ermöglichen es, eine Web-Applikation in wiederverwendbare Komponenten zu gliedern [55]. Es zwar ist möglich die Komponenten ohne irgendein Framework zu benutzen und wiederzuverwerten, jedoch gibt es mehrere Frameworks die dies vereinfachen. Bekannte Frameworks die die Komponentenbasierte Architektur nutzen sind react [43], VueJs [53] und angular [4].
In Abbildung 3.8 kann man eine Applikation als Komposition von Komponenten verstehen. Hier liegt der Fokus auf der Wiederverwertbarkeit der einzelnen Komponenten.
Komponenten können einzelne Sichtbare Elemente auf einer Seite widerspiegeln, das könnte ein Button sein, ein Menü, oder ein Akkordeon. Komponenten können jedoch auch Services sein, die z.B. eine Schnittstelle konsumieren, diese Komponenten, werden von anderen Komponenten benutzt um Informationen zu erhalten, die sie anzeigen können. Eine Komponente ist also eine modulare, austauschbare und wiederverwertbare Sammlung von Funktionalitäten, die diese Funktionalitäten kapseln und auf einer einfacheren Schnittstelle nach außen anbieten.

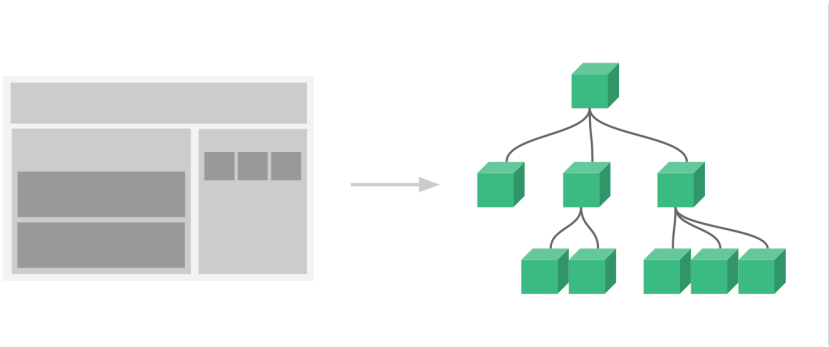

Abbildung 3.8: Komponentenbasierte Webseite laut VueJs Einführung [52].

3.3.2 Komponenten

In diesem Kapitel soll eine grobe Übersicht über die struktur der genutzten Komponenten aufgezeigt werden. Da es bei einem komponentenbasierten Architekturstil zu vielen kleinen Komponenten kommt, ist es müßig jede einzelne Komponente aufzuzeigen. Daher werden hier die Komponenten erläutert, die den groben Ablauf des Clients demonstrieren können. In Abbildung 3.9 ist der Aufbau des Clients Anhand zwei aufrufbaren Seiten aufgezeigt. Wie auch in Abbildung 3.8 zu sehen, gibt es eine Wurzel-Komponente, von denen andere Komponenten aus genutzt werden. Diese AppComponent Komponente, bindet eine Nachrichten Komponente ein (MessageComponent), die Rückmeldungen an den Benutzer anzeigen kann, außerdem enthält sie ein Komponente die die Navigation innerhalb der Applikation ermöglicht (NavigationComponent). Die Router Komponente, (RouterComponent), wird ebenfalls von der Wurzelkomponente aufgerufen. Der Router sorgt dafür, dass abhängig von der URL andere Komponenten angezeigt werden. Er bietet unter anderem eine Übersichtsseite (Dashboard-Component) und eine ProfilDetailseite (ProfileDetailComponent) an.

Die Übersichtsseite enthält eine Komponente die die Suche nach Profilen anbietet (Search-Component) und den AuthGuard. Der AuthGuard ermöglicht es der jeder Komponente zu überprüfen ob ein Benutzer autorisiert ist den aktuellen Inhalt zu sehen. Die Suchkomponente nutzt die Komponenten Profileservice und SearchResultComponent, sowie den Authguard. Der ProfileService und der AuthGuard haben zwar nicht *Component* im Namen, sind jedoch trotzdem eine modulare, gekapselte komponente, die einfach durch andere Komponenten auszutauschen sind, solange die gleiche Schnittstelle implementiert ist. Diese Komponente ist für die Kommunikation mit der REST-API zuständig, sofern es Profile betrifft, so wird sie

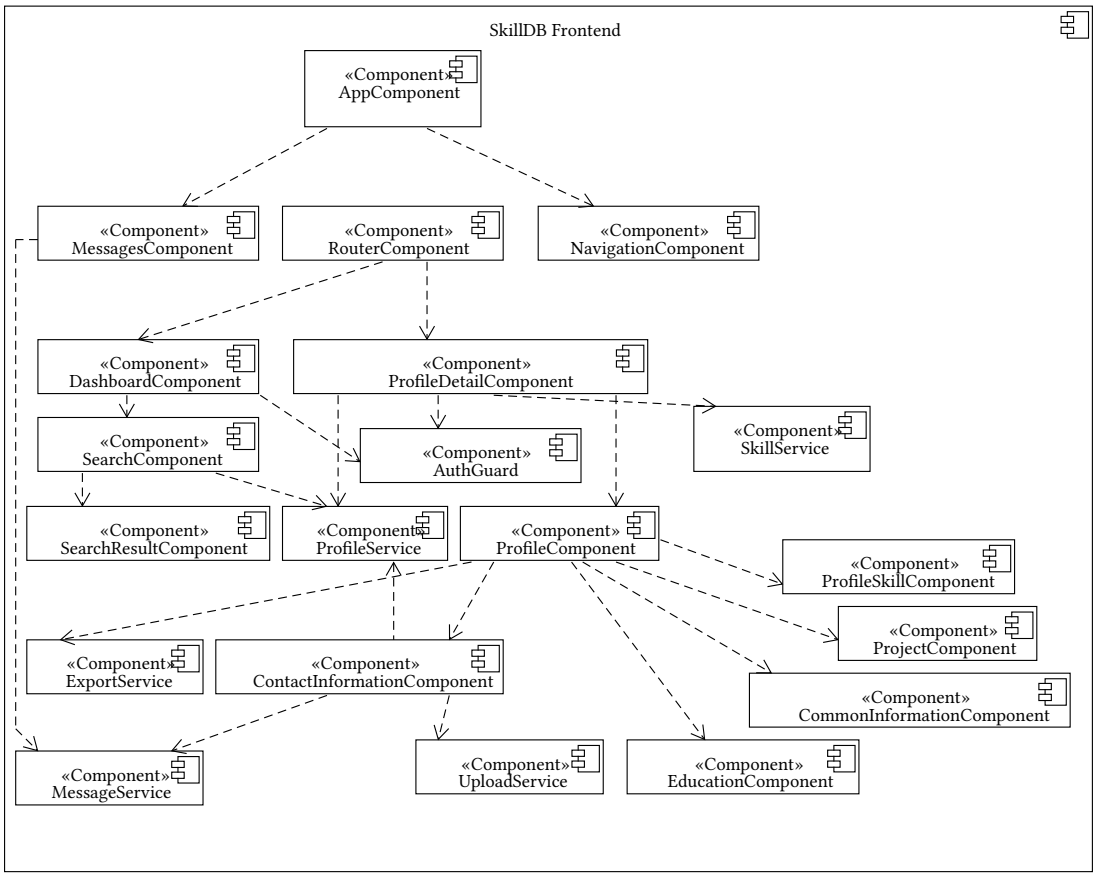

Abbildung 3.9: Teilweises Komponentendiagramm des Clients

hier benutzt um die Suche nach Profilen 3.2.3 auszuführen. Die SearchResultCompontent wird genutzt um die Suchergebnisse für den Benutzer darzustellen.
An diesem Beispiel kann man deutlich sehen, dass Komponenten direkt mit dem Benutzer interagieren können, wie z.B. die Suchkomponente, oder tiefergehende Funktionen mit einer einfacheren Schnittstelle bereitstellen, wie z.B. der ProfileService. In diesem Client gibt es nun also eine Wurzelkomponente, die der Startpunkt der Anwendung ist. Diese nutzt sichtbare Komponenten die immer angezeigt werden, wie die Navigationskomponente, und die Nachrichtenkomponente. Außerdem nutzt sie die interne Routerkomponente, welche entscheidet was weiterhin angezeigt werden soll. Der Router ruft sogenannte Pages, also Seiten, auf, Komponente die vom Router eine oder mehrere URLs zugewiesen bekommen. Falls eine solche Seite Daten braucht die sie Anzeigen soll, wie z.B. die Profildetailseite, benutzt sie eine ServiceKomponente, die sie mit Daten versorgt.

Eine Seite die delegiert nun die Darstellung dieser Informationen an weitere sichtbare Komponenten, die von der Seite angeordnet und mit Daten versorgt werden. Diese Komponenten können weitere Funktionalitäten anbieten, um z.B. Daten zu bearbeiten und zu persistieren. Als Beispiel hierfür soll die ContactInformationComponent dienen. Die Aufgabe dieser Komponente ist die Kontaktinformationen anzuzeigen, zu bearbeiten, und dem MessageService Rückmeldung zu geben ob die Operationen erfolgreich waren oder nicht. Dazu nutzt sie ebenfalls den ProfileService um das Persistieren der bearbeiteten Daten zu veranlassen. Einen weiteren Service, den UploadService, um den Upload von Profilbildern zu ermöglichen. Bei der implementierung kann es dabei noch weitere Komponenten geben zu denen die oben genannten Aufgaben delegiert werden können.
Grundsätzlich lassen sich die Komponenten in die Folgenden Kategorien unterscheiden.

Page Diesen Komponenten werden vom Router URLs zugeordnet. Sie entsprechen weitesgehend dem was eine Internetseite entspricht, benutzen jedoch weitere Komponente

Service Services enthalten Geschäftslogik oder oder andere Funktionen. Hier werden REST-Anfragen durchgeführt, Informationen gespeichert, oder Berechtigungen geprüft.

Model Diese Komponenten sind reine Informationsträger, sie spiegeln unter anderem die Objekte aus der REST-API wider.

3.3.3 Sequenzdiagramme

Analog zu 3.2.3 werden in diesem Teil Sequenzdiagramme genutzt, um das Zusammenspiel der einzelnen Komponenten zu skizzieren. Bei allen Diagrammen gilt hier, dass das Backend nicht genauer betrachtet wird, und als eine geschlossene Komponente behandelt wird.

Suchen nach Profilen

In Abbildung 3.10 wird der Ablauf nach der Suche von Profilen dargestellt, dabei sucht Praktikant Bob nach Profilen die die Fertigkeit *git* besitzen, nachdem er eingeloggt ist.

1. Der Nutzer füllt die Suchmaske aus, und schickt eine Suchabfrage ab.
2. Die SearchComponent delegiert die suchanfrage an den ProfileService.
3. Der ProfileService befragt die REST-API nach Profilen die *git* enthalten.
4. Das Backend schickt eine Sammlung von Profil Data transfer objects.

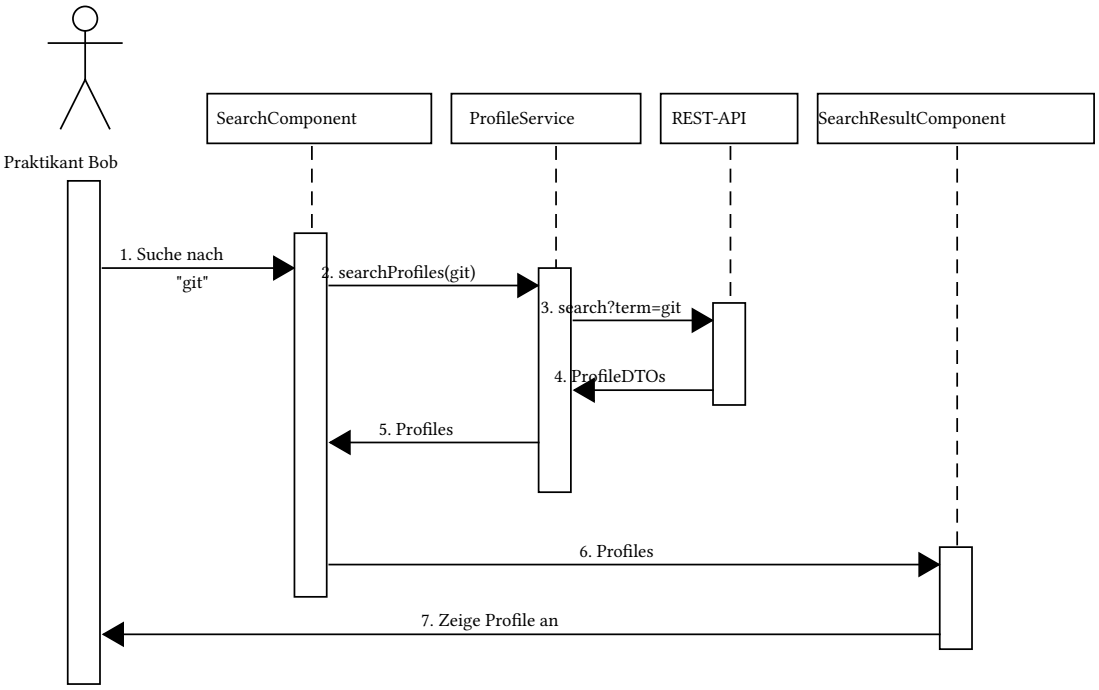

Abbildung 3.10: Sequenzdiagramm: Suchen nach Profilen im Client

5. Der ProfileService transformiert die DTOs in Profilobjekte, und schickt diese an die Suchkomponente.
6. Die Suchkomponente reicht die Profile an die SearchResultComponents weiter, damit die Suchergebnisse angezeigt werden können.
7. Die SearchResultComponent zeigt die Profile in ihrem Template für den Nutzer sichtbar dar.

Aufruf des Dashboards

Praktikant Bob ruft das Dashboard der Seite auf, nachdem er eingeloggt ist. In Abbildung 3.11 ist das Sequenzdiagramm für diesen Ablauf zu sehen.

1. Der Benutzer ruft die Dashboardseite auf.
2. Die AppComponent weist den Router an, den Inhalt für die Seite mit der endung /dashboard zu rendern

3. Der Router entscheidet, dass mit dieser Adresse die DashboardComponent ihr Template rendern soll.
4. Die Dashboard Komponente prüft mithilfe des AuthGuards ob der Benutzer den eigenen Inhalt sehen darf.
5. Der AuthGuard entscheidet, dass der Benutzer berechtigt ist, und teilt dieses der Dashboard Komponente mit.
6. Die Komponente rendert ihr Template und gibt es an den Router weiter
7. Der Router gibt diesen Inhalt weiter an die Wurzelkomponente
8. Die Wurzelkomponente weist die Navigationskomponente an, ihren Inhalt zu rendern
9. Die NavigationComponent gibt das gerenderte Template an die Komponente zurück
10. Die WurzelKomponente ruft die MessageComponent auf um ihr Template zu erhalten
11. Die MessageComponent ruft den MessageService auf, und befragt ihn nach Nachrichten die anzuzeigen sind
12. Der Service gibt nachrichten an die Komponente zurück
13. Die Komponente rendert ihr Template mit den erhaltenen Nachrichten und gibt dieses an die AppComponent zurück
14. Die AppComponent fügt zusammen um dem Benutzer die Dashboard Seite anzuzeigen.

Hierbei ist die DashboardComponent vereinfacht dargestellt, die DashboardComponent nutzt durch Komposition weitere Komponenten, die für eine bessere Übersicht hier nicht behandelt werden. Außerdem ist es möglich, dass die Schritte zwei bis sieben, parallel zu den Schritten acht bis neun und zehn bis 13 ausgeführt werden.

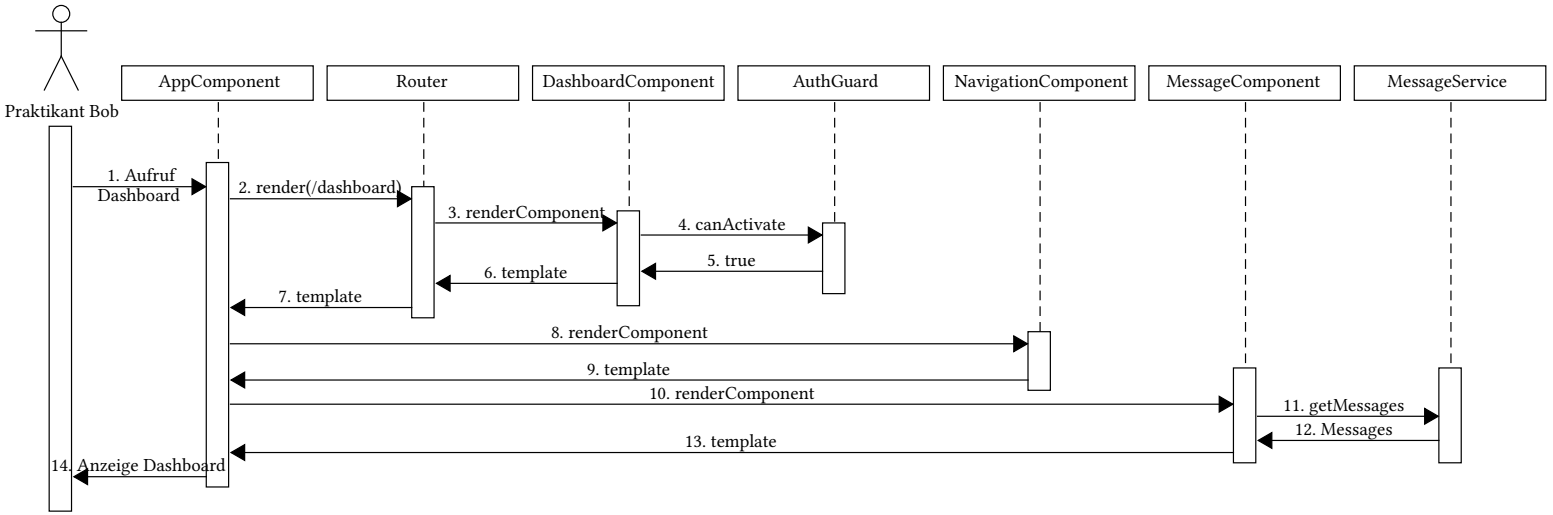

Abbildung 3.11: Sequenzdiagramm: Aufrufen des Dashboards

3.3.4 Angular

Angular [4] ist ein Open-Source Framework das viele Konzepte einer Komponentenbasierten Architektur 3.3.2 umsetzt. Angular ist leicht mit seinem Vorgänger AngularJs [7] zu verwechseln, unterscheidet sich aber deutlich von ihm. Angular existiert seit 2016 und wird von einer Community, sowie Google entwickelt. Es wird unter der MIT-Lizenz [5] veröffentlicht, was ermöglicht das Framework sowohl für open-source als auch closed-source Projekte zu verwenden.
Am 24. Juni 2020 wurde die Angular Version 10.0 veröffentlicht, welche zum Zeitpunkt der erstellung des Clients noch nicht verfügbar war. Dies bedeutet, dass Angular im Client mit Version 9.* genutzt wird, ein Upgrade auf Angular 10 wird daher in einem späteren Release erfolgen.
Angular steht in Konkurrenz zu React und VueJs. Während React als Open-Source Framework von Facebook und Instragram unterstützt wird, wird Vue.js nicht direkt von großen Partnern unterstützt. Bei beginn der Realisierung des Clients, gab es jedoch wenig Kompetenzen im Bereich Vue.js und React in der betreuenden Firma. Dies bedeutete, dass sich für Angular entschieden hatte, um eine weiterentwicklung des Tools möglichst einfach Fortzusetzen.
Angular empfiehlt die Verwendung von Typescript [50] welches es ermöglicht statisch typisierten objektorientierten Code zu schreiben. Es ist dafür gedacht Single-Page Applications zu entwickeln, also Webseiten bei denen kein neuladen der Seite durchgeführt wird, sondern lediglich der Inhalt der sich verändern einer Seite ausgetauscht wird. Dies wird durch dynamisches nachladen von neuen Inhalten ermöglicht, und erzeugt ein flüssigeres Nutzererlebnis im Vergleich zu herkömmlichen Internetseiten. Außerdem vertritt Angular den Mobile First-Ansatz, was bedeutet dass mobile Endgeräte bzw. deren Anforderungen priorisiert werden.
Angular benutzt ebenfalls RxJS [45], eine Bibliothek die benutzt wird um asynchrone und eventbasierte Anwendungen zu erstellen. Hierfür nutzt es das Observer Pattern mit sogenannten Observables, welche an vielen Stellen in einer Angular Applikation genutzt werden. Angular verfügt, ebenso wie Symfony über Dependency Injection, und ermöglicht es dadurch Code einfacher zu kapseln.

4 Realisierung

Dieses Kapitel beschreibt die Umsetzung der Anwendungen beschrieben. Wie auch im Design Kapitel,wird hier zwischen dem Backend und dem Client unterschieden.

4.1 Backend

In diesem Bereich wird die Entwicklung des Backends beschrieben. Das Backend beinhaltet alle Logik die notwendig ist, um einen konsistenten Datenstand zu gewährleisten, außerdem beinhaltet es die Schnittstelle zur Datenbank. Hier werden ebenfalls Systeme vorgestellt die zwingend notwendig sind, um ein Ausführen der Applikation zu gewährleisten, oder wichtig bei der Entwicklung waren.

4.1.1 Verwendete Komponenten

An dieser Stelle werden die externen Komponenten beschrieben, die notwendig sind um die Applikation auszuführen, aber nicht teil der Applikation selbst sind. Hierbei handelt es sich um einen Webserver, ein Datenbanksystem, und den LDAP Server, sowie die Programmiersprache PHP. Diese Komponenten sind alle durch andere Komponente ihrer Art austauschbar.

PHP

PHP ist eine Skriptsprache die hauptsächlich zur erstellung dynamischer Webseiten oder Webanwendungen benutzt wird. Der Name PHP steht für *PHP Hypertext Preprocessor* und ist somit ein rekursives Acronym. Die Sprache wurde ursprünglich als funktionale dynamisch typisierte Sprache entwickelt. Jedoch wurde die Sprache um Laufe von über 25 Jahren weiterentwickelt und erlaubt nun auch in Major Version 7 statisch typisierten Objektorientierten Code zu schreiben. Da PHP Code keine Compilierte Sprache ist, sondern auf einem interpreter direkt

ausgeführt wird, ist es hilfreiche eine intelligente IDE zu haben, die den Code inspiziert. In diesem Projekt wird PHP objektorientiert und in den meisten Fällen ebenfalls statisch typisiert. Außerdem wird PHP in zusammenhang mit Symfony 3.2.1 benutzt, um einen großteil erprobter Bibliotheken zu nutzen.

Webserver

Ein Webserver ist eine Anwendung die Anfragen von außen an interne Anwendungen weiterleitet, und deren Antworten an den Client zurückschickt. Der Webserver ist somit dafür verantwortlich dafür, die REST-Schnittstelle zu veröffentlichen. Im Zuge der Entwicklung wurden die beiden am meisten benutzten Webserver genutzt [54]. Es wurde der Apache HTTP Server benutzt, um die Anwendung direkt auf einem Server nach außen kommunizieren zu lassen, und später wurde Nginx im genutzt um die Applikation im Docker-Kontext nach außen verfügbar zu machen. Was aufzeigt, dass ohne Probleme die Webserver ausgetauscht werden können.
Der Apache HTTP Server wird von der Apache Software Foundation entwickelt und ist ein Open-Source Projekt und steht über die Apache 2.0 Lizenz [10] kostenlos zur verfügung. Üblicherweise wird ein Apache Webserver direkt auf einem Server ausgeführt, während in einem Docker-Kontext fast immer ein Nginx zu finden ist. Jedoch gibt es keinen wahrnehmbaren Unterschied bei der Ausführung mit beiden Webservern, sodass hier eine definitive Entscheidung welcher der beiden Webserver aussteht.

Datenbanksystem

Ein Datenbanksystem ist zuständig für die persistente Speicherung von Daten. In diesem Projekt wird eine PostgreSQL Datenbank benutzt. PostgreSQL ist, wie der Name bereits vermuten lässt, weitestgehend SQL-Konform [41]. Das bedeutet, dass PostgresSQL viele Features des SQL-Standards unterstützt, was aber nicht bedeutet, dass es nicht noch weitere eigene Funktionalitäten anbietet. Im Zuge der Planung der Neuentwicklung der SkillDB wurde deutlich gemacht, dass die bereits bestehenden Daten, nicht verloren gehen sollen. Dies hatte im ersten Schritt zur Folge, dass die Datenbanksoftware für eine einfache Migration die gleiche bleiben sollte, und somit wird im alten sowie im neuen System PostgreSQL benutzt. Grundsätzlich lässt sich die Datenbank jedoch einfach austauschen, wird bei den Tests (siehe Kapitel 5.1.1) eine SQLite Datenbank, anstelle eine PostgreSQL benutzt.

Datenbank Updater

Die bestehende Datenbank ist ein einem Format, welches nicht komplett mit einem PHP nutzbar ist. Daher war es nötig ein Werkzeug zu entwickeln, welches die Datenbank in ein passendes Format bringt. Hierfür wurde ein Java Tool entwickelt das Grails nutzt um die folgenden Aufgaben zu erledigen. Es muss die bestehende Datenbank ausgelesen werden, und ein SQL Script zu erstellt werden welches die notwendigen Änderungen enthält um die Datenbank mit PHP nutzen zu können. So liest dieses Programm beispielsweise Daten Datumsfelder aus der Datenbank die als Binärdaten gespeichert sind, und erstellt ein SQL Script das diese in SQL-datetime Felder umwandelt. Das so erstellte SQL Script kann nun auf eine Datenbank angewandt werden, die das PHP Tool anspricht.

4.1.2 Dependency Manager

Ein Dependency Manager oder Package Manager ist ein Programm, welches Fremdbibliotheken installieren kann. Bei Paketmanagern werden die benötigten Bibliotheken mit ihren Versionen als Konfiguration festgehalten. Dies erspart die notwendigkeit diese Bibliotheken in einem Versionskontrollsystem zu speichern, weil genau festgelegt ist welche Bibliotheken mit welcher Version benötigt werden. Der Paketmanager kann diese dann mit einfachen Befehlen herunterladen und installieren. Das hat den Vorteil, dass Repositories weniger speicherplatz benötigen. Paketmanager bieten außerdem Funktionalitäten um Bibliotheken einfach zu entfernen oder zu aktualisieren.
Paket Manager installieren diese Abhängigkeiten lokal für jedes einzelne Projekt, sodass unterschiedliche Projekte auf dem gleichen Computer, die gleiche Bibliothek in unterschiedlichen Versionen, konfliktfrei nutzen können.
Composer [13] ist der standard Paket Manager wenn es sich um PHP Programme handelt. Er ist ebenfalls ein Open-Source Projekt das unter der MIT-Lizenz veröffentlicht wurde und mit PHP entwickelt wurde.
Die Abhängigkeiten zu Drittbibliotheken werden bei composer in einer JSON Datei festgehalten, der composer.json. Diese Datei wird mit in ein Versionskontrollsystem geladen, damit alle Entwickler diese nutzen können. Zusätzlich zu dieser Datei gibt es noch eine composer.lock datei, die ebenfalls das JSON-Format benutzt und in das VCS geladen werden sollte. Beide Dateien unterscheiden sich in ihrem Zweck.

composer.json Diese Datei enthält alle Pakete die man benutzen möchte, mit Versionseinschränkungen. Diese Einschränkungen können spezifische Versionen sein, oder auch

eine Reihe von Versionen. So erlaubt es z.B. die Einschränkung 2.1.* alle Versionen zu nutzen die mit 2.1 anfangen.

composer.lock Diese Datei enthält alle Pakete die tatsächlich benutzt werden. Dies bedeutet, dass auch Abhängigkeiten der Abhängigkeiten angeben werden. Außerdem wird in dieser Datei explizit angegeben, welche Version genutzt wird.

Als beispiel soll hier der Unterschied zwischen den beiden Dateien anhand der Definition von API-Platform erläutert werden. In 4.1 ist ein Ausschnitt aus einer composer.json aufgezeigt. Dort kann man sehen, dass die Bibliothek *symfony/process* mit dem Versions Constraint *^5.0.4* eingebunden. Dies bedeutet, dass jede Version, die größer oder gleich 5.0.4 und kleiner als 6.0.0 ist, erlaubt ist.

```
1 {
2     "minimum-stability": "stable",
3     "require": {
4         "symfony/process": "^5.0.4",
5     }
6 }
```

Auflistung 4.1: Auszug aus einer composer.json Datei

In 4.2 kann man in Zeile 4 sehen, dass dadurch die version *5.1.8* genutzt wird. In Zeile 17 wird die Systemanforderung gestellt, dass eine PHP Version vorhanden sein muss die größer ist als 7.2.5, und in der darauffolgenden Zeile wird angegeben, dass eine weitere Bibliothek benötigt wird.

```
1  {
2      {
3              "name": "symfony/process",
4              "version": "v5.1.8",
5              "source": {
6                  "type": "git",
7                  "url": "https://github.com/symfony/process.git
                       ",
8                  "reference": "f00872c3f6804150d6a0f73b4151daab
                       96248101"
9              },
10             "dist": {
```

```
11                  "type": "zip",
12                  "url": "https://api.github.com/repos/symfony/p
                        ...",
13                  "reference": "f00872c3f6804150d6a0f73b4151daab
                        96248101",
14                  "shasum": ""
15              },
16              "require": {
17                  "php": ">=7.2.5",
18                  "symfony/polyfill-php80": "^1.15"
19              },
20              "type": "library",
21              "time": "2020-10-24T12:01:57+00:00"
22          },
23  }
```

Auflistung 4.2: Auszug aus einer composer.lock Datei

Da die lock-Datei ebenfalls im VCS ist, hat jede Entwicklerin diese Datei und kann sicher sein, dass alle die gleiche Version der Bibliotheken haben. In Zeile 1 von 4.3 kann man den Befehl sehen um alle Bibliotheken die in einer .lock Datei stehen zu installieren. Der Befehl in Zeile 2 aktualisiert alle Bibliotheken innerhalb ihrer erlaubten Versionsreichweiten. Dies sorgt dafür dass die .lock Datei aktualisiert wird, sodass alle anderen Entwicklerinnen ebenfalls die gleichen Versionen nutzen. Zeile 5 zeigt einen Befehl, der auflistet welche Pakete nicht mehr aktuell sind.

```
1   composer install
2   composer update
3   composer require
4   composer remove
5   composer outdated
```

Auflistung 4.3: Liste einiger Composer Befehle

Für den Composer gibt es das Standard Repository Packagist [34]. Hier sind mehr als 280.000 Pakete mit über 2,4 Millionen Versionen verfügbar. Dort sind die Pakete mit ihren Abhängigkeiten und Konflikten zu anderen Bibliotheken sichtbar. Allerdings ist es auch möglich andere Repositories einzubinden. Diese Repositories können verweise auf ein Dateisystem sein, oder

auch Verweise auf ein privates GIT Repository. Dies ermöglicht es proprietäre Bibliotheken ebenfalls mit Composer und seinen Vorteilen zu nutzen.

4.1.3 Umsetzung

Hier wird auszugsweise die Umsetzung des Projekts vorgestellt. In 4.4 ist ein Teil der Klasse SystemUser zu sehen. Dieser Benutzer spiegelt den aktiven Benutzer im Programm wieder. Durch die Nutzung von API-Platform und Doctrine ist wenig zusätzlicher Code notwendig um mit dem Benutzer zu interagieren.
Wie in den Zeilen Sechs bis Neun zu sehen, beginnen alle Annotationen, die Datenbank relevant sind mit *ORM*. Der Eintrag `@ORM\Entity` teilt Doctrine mit, dass es sich hier um eine Klasse handelt die Verwaltet werden soll. Der Eintrag `@ORM\Table` ist optional und lässt Einfluss auf die Tabelle nehmen. Hier wird er genutzt um einen UniqueConstraint auf das Spalte uid zu setzen.
Alle Felder die nach Zeile 31 kommen, sind Felder die in der Datenbank gespeichert werden. So ist die Definition ab Zeile 34ff die Definition für die ID der Tabelle. `GeneratedValue` weist Doctrine an eine Strategie zu nutzen um Automatisch den Wert zu generieren. In dieser Anwendung wird die von der Datenbank vorgegebene Strategie verwendet, sie kann jedoch auch explizit angegeben werden. In Zeile 41 kann man in der `Column` Definition sehen, dass der Spaltenname vom Namen der Variable abweicht. Falls der Name nicht angegeben ist, wird der Name des Feldes als Spaltenname benutzt.
Für das Feld `roleEntities` ist wie in 2.1 beschrieben eine N:M Verbindung notwendig, was in Zeile 46 durch die `ManyToMany` Annotation ausgedrückt wird. In den Zeilen 47ff wird die Join Tabelle, die Rollen und Nutzer verbindet beschrieben. In Zeile 58 ist die 1:1 Verbindung zwischen Profil und SystemBenutzer beschrieben.
Zwischen den Zeilen 10 und 29 befindet sich die Konfiguration für API-Platform, sie beginnt mit `@ApiResource`. In der detaillierten Konfiguration für die verschiedenen Operationen, jeweils eine Operation definiert, auf die Voter 3.2.2 abstimmen können. Zusätzlich sind alle Operationen hinter einer globalen Sicherheitsregel versteckt, die erwartet, dass ein Benutzer eingeloggt ist. In Zeile 27 ist zu sehen, dass für die Operation mit der DELETE-Methode ein Controller angegeben ist. Dies sorgt dafür, dass eigene Logik angewandt wird, um einen Benutzer zu löschen, damit alle Zugehörigen Objekte ebenfalls gelöscht werden. Außerdem wurde in Zeile 14ff eine besondere Operation definiert, die auf die `/me` Route hört, dessen Controller die Klasse `UserMe` ist und per GET-Anfrage aufgerufen wird.

```
1 <?php
```

```
2
3  namespace App\Entity;
4
5  /**
6   * @ORM\Entity(repositoryClass="App\Repository\SystemUserRepository")
7   * @ORM\Table(uniqueConstraints={
8   *      @ORM\UniqueConstraint(name="system_user_uid_key", columns={"uid"})
9   * })
10  * @ApiResource(
11  *      collectionOperations={
12  *          "get"={"method"="GET",
13  *                 "security"="is_granted('SYSTEM_USER.VIEW.ALL')"},
14  *      "special"={
15  *          "method"="GET",
16  *          "path"="/me.{_format}",
17  *          "controller"=UserMe::class
18  *      }
19  *      },
20  *      itemOperations={
21  *          "get"    ={"method"="GET",
22  *                     "security"="is_granted('SYSTEM_USER.VIEW', object)"},
23  *          "put"    ={"method"="PUT",
24  *                     "security"="is_granted('SYSTEM_USER.EDIT', object)"},
25  *          "delete"={"method"="DELETE",
26  *                     "security"="is_granted('SYSTEM_USER.DELETE', object)",
27  *                     "controller"=UserDeleteController::class},
28  *      }
29  *      )
30  */
31 class SystemUser implements UserInterface
32 {
33     /**
34      * @ORM\Id
35      * @ORM\GeneratedValue
36      * @ORM\Column(type="bigint")
37      */
38     private $id;
39
40     /**
41      * @ORM\Column(name="uid", type="string")
42      */
43     private $username;
44
```

```
45      /**
46       * @ORM\ManyToMany(targetEntity="App\Entity\Role")
47       * @ORM\JoinTable(name="system_user_role",
48       *         joinColumns={@ORM\JoinColumn(
49       *         name="system_user_authorities_id", referencedColumnName="id"
50       *         )},
51       *         inverseJoinColumns={@ORM\JoinColumn(
52       *                 name="role_id", referencedColumnName="id"
53       *         )}
54       *         )
55       */
56      private $roleEntities;
57      /**
58       * @ORM\OneToOne(targetEntity="App\Entity\Profile", mappedBy="user")
59       */
60      private $profile;
61  }
```

Auflistung 4.4: Teile der SystemUser Klasse

Die Benutzerklasse hat im wirklichen Projekt noch mehr Felder die Informationen enthalten, und ohne weitere Konfiguration ausgegeben werden. Es ist jedoch möglich über weitere Konfiguration Felder bei bestimmten Operationen auszulassen, oder zusätzliche Felder mit einzubinden.

Als Beispiel dient dafür der Code in 4.5, welcher für besser Übersichtlichkeit nur den relevanten Code enthält. Hier wird durch Konfiguration, die Entität `Skill` der Entität bei lesenden operationen angehängt.

```
1   <?php
2
3   namespace App\Entity;
4
5   /**
6    * @ApiResource(normalizationContext={"groups"={"profileSkill.get"}})
7    */
8   class ProfileSkill
9   {
10      /**
11       * @Groups({"profileSkill.get", "profile.search"})
12       * @Assert\NotBlank()
13       * @Assert\Positive()
14       */
```

```
15      private $rating;
16
17      /**
18       * @ORM\ManyToOne(targetEntity="App\Entity\Skill",
19       *      inversedBy="profileSkills")
20       * @ORM\JoinColumn(nullable=false)
21       * @Groups({"profileSkill.get", "profile.search"})
22       */
23      private $skill;
24  }
```

Auflistung 4.5: Teile der ProfileSKill Klasse

In Zeile 6 ist ein `normalizationContext` angegeben, da der Name der Gruppe einer Konvention folgt, gilt sie für alle GET-Anfragen. Om Zeile 11 und Zeile 21 sieht man das dort Groups angegeben wurden, durch angeben oder weglassen dieser Gruppen, kann man steuern welche Felder über die Api exponiert werden. Dadurch definiert man über Konfiguration Data Transfer Objects. In Zeile 21 kann man sehen, dass diese Gruppe an einer Entität hängt, die über eine 1:N Beziehung verbunden ist. In der Entität sind nun die Felder, die bei lesenden Anfragen auf ProfileSkills angezeigt werden ebenfalls mit der Gruppe `profileSKill.get` versehen. In Auflistung 4.6 kann man eine Beispielhafte Antwort auf eine GET-Anfrage nach einem ProfileSkill sehen.

```
1  {
2    "@context": "/api/contexts/ProfileSkill",
3    "@id": "/api/profile_skills/3989",
4    "@type": "ProfileSkill",
5    "id": "3989",
6    "rating": 2,
7    "profile": "/api/profiles/2642",
8    "skill": {
9      "@id": "/api/skills/137",
10     "@type": "Skill",
11     "id": "137",
12     "name": "CouchDB",
13     "active": true,
14     "category": "/api/skill_categories/132"
15   }
16 }
```

Auflistung 4.6: Beispielantwort bei einer GET-Anfrage auf ProfileSkills

Durch die Nutzung des normalizationContexts und der entsprechenden Gruppe sind nun die Informationen des Skills zu dem ProfilSkill inkludiert. Ohne diese extra Konfiguration, würde ein Skill analog zu dem Profil als IRI gerendert werden.
Im Folgenden wird die Struktur der Voter anhand von Beispielen erläutert. Alle Voter müssen die Methoden `supports` und `voteOnAttribute` implementieren. Die `supports` Methode bestimmt ob ein Voter abstimmen darf, und die `voteOnAttribute` Methode stimmt ab. In Auflistung 4.4 kann man sehen, dass die Attribute ein Präfix haben, dass der Klasse entspricht, dadurch ist es einfach zu bestimmen ob ein Voter abstimmen darf, oder nicht. Diese Logik wurde daher in eine Superklasse ausgelagert.

```
1   protected function supports($attribute, $subject)
2   {
3       if (in_array($attribute, $this->collectionOperations())) {
4           return true;
5       }
6       $class = $this->getSubjectClass();
7       if (in_array($attribute, $this->itemOperations())
8           && $subject instanceof $class) {
9           return true;
10      }
11      return false;
12  }
```

Auflistung 4.7: supports Methode der Superklasse

Die Superklasse definiert drei abstrakte Methoden, die die Attribute definieren auf denen abgestimmt wird, und für welche Klasse sie abstimmen. Anschließend implementiert jeder Voter seine eigene Logik zum Abstimmen.

```
1   protected function voteOnAttribute(
2       $attribute,
3       $subject,
4       TokenInterface $token
5   )
6   {
7       $user = $token->getUser();
8       // if the user is anonymous, do not grant access
9       if (!$user instanceof UserInterface) {
10          return false;
```

```
11          }
12          switch ($attribute) {
13              case self::VIEW_ALL:
14              case self::EDIT:
15              case self::DELETE:
16                  return $this->hasAdminRole($user);
17              case self::VIEW:
18                  return $this->hasAdminRole($user)
19                      || $user->getUsername() === $subject->getUsername();
20              case self::CREATE:
21              default:
22                  return false;
23          }
24      }
```

Auflistung 4.8: voteOnAttribute Methode des SystemUserVoters

In Auflistung 4.8 kann man die Methode zum Abstimmen, des SystemUserVoters sehen. Die Logik ist hier sehr simpel, da Benutzer nur den Eigenen Benutzer sehen dürfen, es sei denn sie sind Admins. Außerdem ist es für alle Nutzer unmöglich neue Benutzer über einen Endpunkt zu erstellen, da dieses automatisch beim ersten Login passiert 3.7.
Die Anforderung Exporte von Profilen 2.2.2 zu erstellen, legt nahe, dass es sich dabei um ein gleiches Layout handeln sollte. Daher ist es wichtig gewesen, eine Exportfunktion bereitzustellen, die das Template des alten Systems benutzt. Aus diesem Grund ist die Entscheidung getroffen wurden, die Funktionalität der alten Anwendung einfach weiter zu benutzen. Dafür wurde der zuständige Code identifiziert, und so gekapselt, und in ein neues Tool bewegt, dass man diese Funktion über die Kommandozeile aufrufen kann. So wurde der alte Code wiederverwendet um einen Kommandozeilenbasierten Exporter zu erstellen, der eine JSON-Datei konsumiert, und daraus z.B. ein Word-Dokument erstellt.

```
1       $jsonFile = $this->generateJsonFile($profile);
2       $process = new Process(
3           [
4               'java',
5               '-jar',
6               $this->exportJar,
7               '--stream',
8               $template,
9               $jsonFile,
10              $target
11          ]
12      );
```

```
13      $process->run();
14      $content = $process->getOutput();
15      $this->fileSystem->remove($jsonFile);
16      return $content;
```

Auflistung 4.9: Teile der exportProfile Methode in der PHP Anwendung

In der Anwendung wird diese Java anwendung dann über die Process-Bibliothek, wie in 4.9 zu sehen, ausgeführt. Dabei wird eine JSON Datei mit den Daten des Profils erstellt. Anschließend wird der Exporter aufgerufen, und jedwede Ausgabe wird gespeichert, da diese durch das Flag `-stream` den Inhalt des Dokuments enthält. Dies wurde erledigt, um weniger auf einer Festplatte zu schreiben und damit die Laufzeit des Exports zu verringern.

4.2 Client

Als Teil dieser Arbeit wurde ebenfalls ein Client als Webservice entwickelt. Dieser Client wird auch als Frontend bezeichnet. Dieser Abschnitt beschäftigt sich mit der Entwicklung des Clients und der genutzten Technologien und Techniken.
Dieser Client ist nur eine von vielen Möglichkeiten einer umsetzung. Andere Clients könnten zum Beispiel Windows oder Linux Anwendungen sein. Jedoch muss jeder Client die Schnittstelle des Backends implementieren.
In Abbildung 4.1 ist das entwickelte Frontend auf einem Mobilgerät bei der Anzeige von Fertigkeiten eines Profils zu sehen.

4.2.1 Verwendete Komponenten und Tools

Relevante Komponenten die für das Frontend benutzt wurden werden hier beschrieben. Dabei werden Komponenten, die ebenfalls für das Backend genutzt werden nicht mehr genauer beschrieben.
Hierbei ist anzumerken, dass der Client keinerlei persistenz besitzt. Es gibt also keine Datenbank oder Dateien die dauerhaft gespeichert werden.

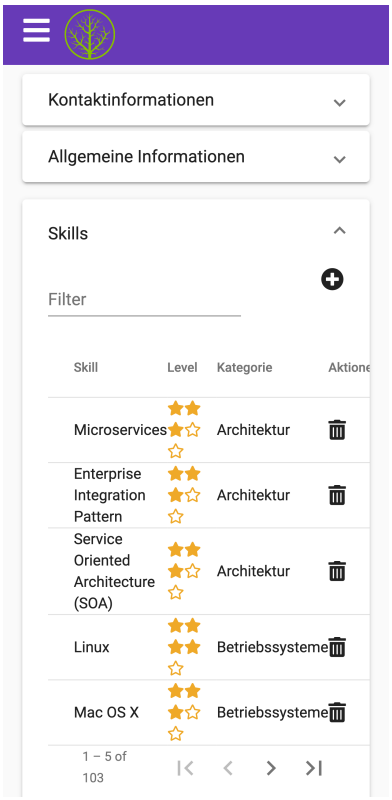

Abbildung 4.1: Fertigkeiten eines Profils auf einem Mobilgerät

Webserver

Der Client benötigt einen Webserver, der es ermöglicht ihn nach Außen kommunizieren zu lassen (siehe Kapitel 4.1.1). Dazu wird aktuell ein Apache Webserver benutzt. Es ist möglich den gleichen Webserver zu nutzen, jedoch nicht notwendig.

Typescript

Da das Frontend mithilfe des Frameworks Angular (siehe Kapitel 3.3.4) entwickelt wird, wird Typescript als Sprache genutzt. Typescript ist eine von Microsoft entwickelte Sprache, die eine Obermenge von Javascript ist. Typescript Code wird von einem Compiler in Javascript code übersetzt, und kann damit auch Javascript Code verwenden. Typescript ermöglicht es objektorientierten und statisch typisierten Code zu schreiben.
Der compilierte Javascript Code wird an den Client übertragen, sodass er in einem Browser aus-

geführt werden kann. Daher ist es üblich den Code zu minimieren(minify) und alle Funktions- und Variablennamen durch Buchstaben zu ersetzen(uglify).

NodeJS und NPM

NodeJS [33] ist eine Laufzeitumgebung für Javascript Code, außerhalb von Browsern. NodeJS (kurz Node) ist ein Open-Source Projekt, das unter der MIT-Lizenz veröffentlicht wird. Node wird üblicherweise genutzt um Javascript-Code auf Servern auszuführen. Der Node Package Manager (NPM) wird mithilfe von Node ausgeführt und verhält sich ähnlich der Dependency Manager aus Kapitel 4.1.2.

4.2.2 Mobile First Webdesign

In den Letzten Jahren haben die Anzahl der mobilen Endgeräte auf dem Markt wie in Abbildung 4.2 zu sehen ist deutlich zugenommen. Dem Diagramm kann man entnehmen, dass der Anteil der Mobilgeräte dem der Desktop Geräte etwa entspricht. Dies wirft die Frage auf wie man eine Webanwendung gestaltet, sodass sie von Mobilen sowie von Desktop Benutzern gleichermaßen benutzt werden kann.
Eine Schwierigkeit dabei ist die geringe Displaygröße von Smartphones. Zusätzlich kommt

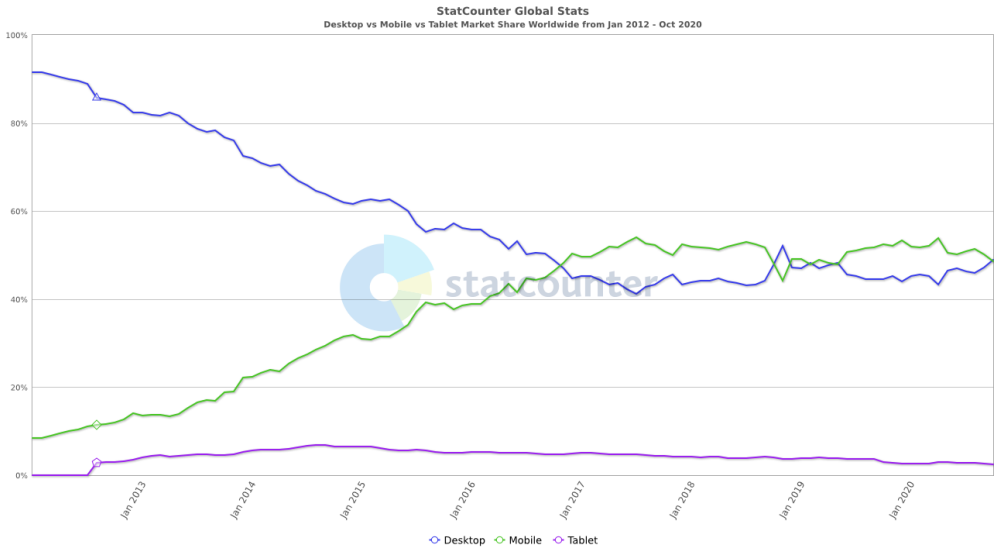

Abbildung 4.2: Anteil der Nutzung von Tablets Desktop- und Mobilgeräten [14]

hinzu, dass Smartphones in der Regel vertikal gehalten werden, ganz im Gegensatz zu Desktop

Geräten. Das bedeutet, man muss ein Erlebnis für Benutzer schaffen, dass sowohl im Querformat als auch im Hochformat funktioniert. Dafür hat sich der sogenannte Mobile First [31] Ansatz durchgesetzt. Demnach unterscheidet sich die Entwicklung einer Mobile First anwendung dadurch, dass sie zuerst für mobile Endgeräte konzipiert ist. Dabei wird ein Entwurf für ein Smartphone entwickelt, und anschließend, wenn dieser Erledigt ist, wird dieser Entwurf auf große Displays skaliert. Dies kann bedeuten, dass man zusätzliche Funktionalitäten einbindet, den Seitenaufbau verändert oder Leerräume einbindet.
Bei dem Design für mobile Ansichten muss man also die folgenden Einschränkungen bei Smartphones achten.

Steuerungsmechanismen Smartphones haben ein kleines Display und werden fast immer mit den Fingern benutzt. Dies bedeutet, dass kleine Steuerelemente schwer zu erreichen sind. Falls ein Benutzer Texteingaben treffen muss, öffnet sich die Tastatur auf dem Handy und ein Teil der Website verschwindet dahinter. Daher sollten Benutzereingaben auf das Nötigste reduziert werden.

Nutzungskontext Die Benutzung des Internets auf Smartphones passiert oft nebenbei, und daher sind nutzer weniger konzentriert. Smartphones werden oft unterwegs und draußen benutzt, also an Orten mit schlechten Licht oder vielen Ablenkungen. Daher soll man ein möglichst einfaches, lesbares und ablenkungsfreies Design entwerfen.

Navigation Typische Navigationsleisten in Webanwendungen mit einer Navigationsbar über die ganze Breite der Seite, funktionieren auf Geräten im Hochformat nicht. Außerdem sollte für Mobilgeräte beachtet werden, dass Benutzer nicht als erstes eine große Navigation zu sehen bekommen, anstelle von echtem Inhalt. Daher ist es notwendig andere Navigationsmechanismen umzusetzen, oder die Navigation an das Ende der Seite zu stellen.

In diesem Projekt wird dies erreicht indem das Material Design [30] von Google benutzt wird. Material ist ein von Google entworfenes minimalistisches Design. Die Umsetzung des Designs ist ein Open-Source Projekt das unter der MIT-Lizenz veröffentlicht wurde. Die große Beliebtheit des Designs hat dafür gesorgt, dass es viele Pakete für unterschiedliche Frameworks gibt, die das Design umsetzen. In diesem Projekt wurde dafür Angular Material [6] benutzt,

4.2.3 Umsetzung

In diesem Abschnitt wird auszugsweise die Umsetzung des Projekts vorgestellt. Im Gegensatz zum Backend, bei dem viel Programmierarbeit durch Bibliotheken abgenommen wurde, wurde

dies hier nicht erledigt. Im Client sind die meisten Bibliotheken, neben dem Framework, Bibliotheken, die Designelemente bereitstellen.
In Abbildung 3.9 kann man sehen, dass die AppComponent, drei Komponenten einbindet. Diese drei Komponenten sind in dem Code von 4.10 ebenfalls zu sehen. In Zeile 2 und 12 wird die Navigationskomponente eingebunden. In Zeile 4 ist zu sehen, dass die Nachrichten Komponente eingebunden wird. Und der Router wird in Zeile 10 eingebunden. Die Logik die zu dieser Komponente gehört macht nichts weiter außer festzustellen ob Inhalte nachgeladen werden oder nicht. In Zeile 6 und Zeile 10 wird bestimmt ob der Inhalt des Routers oder ein Spinner angezeigt werden soll.

```
1  <div class="mat-app-background basic-container" style="height: 100%">
2    <app-nav>
3      <div class="content">
4        <app-messages></app-messages>
5      </div>
6      <div *ngIf="loading" class="spinner">
7        <mat-progress-spinner mode="indeterminate"></mat-progress-spinner>
8      </div>
9      <div class="content">
10       <router-outlet *ngIf="!loading"></router-outlet>
11     </div>
12   </app-nav>
13 </div>
```

Auflistung 4.10: Der Template Code der app.compontent.html

Der Router selbst in eine Komponente die durch Angular mitgeliefert wird, und durch eine externe Datei konfiguriert wird. In Auflistung 4.11 ist ein Ausschnitt aus den Routen Definitionen zu sehen. Dort sind drei Definitionen von Routen zu sehen, die sich jeweils Unterschiedlich verhalten. Die Erste in Zeile 1 leitet einfach nur auf die `\dashboard` route weiter, wenn man die Seite ohne weitere Route aufruft. Das bedeutet, wenn man die applikation zum Beispiel über http://localhost:4200 aufruft, wird man zu http://localhost:4200/dashboard weitergeleitet

```
1  {path: '', redirectTo: '/dashboard'},
2  {
3    path: 'dashboard',
4    component: DashboardComponent,
5    canActivate: [AuthGuard]
6  },
7  {
8    path: 'profile/:id',
9    component: ProfileDetailComponent,
```

```
10      canActivate: [AuthGuard],
11      resolve: {
12        profile: ProfileDetailResolverService,
13        user: UserMeResolverService,
14        skills: SkillListResolverService,
15
16      }
17    },
```

Auflistung 4.11: Auszug aus der Routendefinition des Clients

In Zeile 2 beginnt die Definition der Route des Dashboards, also der Komponente auf der die Suche zu sehen ist. Hier wird ebenfalls ein `path` angegeben auf den der Router reagieren soll. Außerdem ist hier eine `component` angegeben, die aufgerufen werden soll, wenn diese Route abgerufen wird. Die letzte Konfiguration dieser Route wird mit dem `canActivate` Schlüssel erledigt. An dieser Stelle wird ein Array von Klassen angegeben, dass die canActivate Methode implementiert. Anhand der canActivate Methode wird dann entschieden ob ein Benutzer diese Route aufrufen darf oder nicht.
Die Dritte und letzte Route in diesem Beispiel ist hat einen etwas anderen Pfad. Ihr Pfad enthält den Teil `:id` welcher anzeigt, dass der Text nach dem Schrägstrich eine Variable ist, die den Namen id bekommen soll. Der `resolve` Block in Zeile 11ff wird genutzt um der Komponente Daten zu übergeben die sie anzeigen soll. Die Komponente erwartet bei erstellung darauf,dass ihr Daten zum Anzeigen übergeben werden, und zwar ein Objekt mit den Schlüsseln `profile`, `user` und `skills`. In diesem Block sind nun ResolverServices angegeben, die dieses Objekt mit Werten füllen. So wird der `ProfileDetailResolverService` die Variable id, aus der Route extrahieren, und das Profil für diese Id von der REST-API holen. Analog dazu werden die anderen ResolverServices ebenfalls Daten holen und der Komponente bereitstellen. Das Nutzen dieser Resolver hat den Vorteil, dass die Komponente nun nicht mehr Abhängigkeiten zu Services hat, die diese Daten holen, sondern nur noch zu dem Router der die Daten der Resolver bereitstellt. So ist es sehr einfach möglich die Resolver auszutauschen, ohne dass die Komponente angepasst werden muss. Das bedeutet, dass das Bereitstellen der Daten in Services gekapselt ist, und die Komponente nur eine Verantwortung hat: Das Anzeigen von Daten.

```
1 const id = route.paramMap.get('id');
2 return forkJoin([
3   this.profileService.getFullProfile(id),
4   this.categoryService.getAllCategories(),
5 ]).pipe(
```

```
6   map(data => {
7     const profile = data[0];
8     profile.profileSkills = profile.profileSkills.map(ps => {
9       ps.skill = this.skillService.associateSkillWithCategory(ps.skill, data
            [1]);
10      return ps;
11    });
12    return profile;
13  }));
```

Auflistung 4.12: Der teilweise Code der profileDetailResolver

Ein so genutzter Resolver muss das Resolve Interface implementieren, was bedeutet er muss die resolve Methode bereitstellen. Eine solche Methode ist in Code-Beispiel 4.12 zu sehen. Hierbei ist zu beachten, dass Angular tief mit RxJS verwoben ist, und alle HTTP-Anfragen an die Schnittstelle asynchron passieren. Daher wird in Zeile 2 ein forkJoin benutzt der besagt, dass die darauffolgenden Methoden erst ausgeführt werden sollen wenn die Aufrufe aus Zeile 3 und 4 abgeschlossen sind. In Zeile 5 wird das Ergebnis dieser Anfragen in eine weitere Methode per Pipe übergeben, so lassen sich mehrere Transformationen nach einander ausführen. Die Transformationen der `map` Methode kombinieren die Informationen aus den Beiden HTTP-Anfragen zu einem ProfilObjekt, und geben dieses als ein Observable zurück. Dieses Observable landet dann über den Router in der ProfileDetailComponent, welches dieses Observable abonniert um es dann anzuzeigen.
Abschließend wird über die Kommunikation zwischen Komponenten gesprochen. In Codeausschnitt 4.13 ist zu sehen wie die Komponente zum Bearbeiten der Allgemeinen Informationen eingebunden wird. Dort sind drei Attribute zu sehen `*ngIf`, `(submitEmitter)` und `[commonInformation]`.
Das Attribut `*ngIf` ist eine Angular (ng) interne Anweisung, die das Element nur rendert wenn der Ausdruck in `edit` wahr ist. Die Komponente benötigt, die allgemeinenInformationen die es bearbeiten soll.
Dies wird `[commonInformation]` bewerkstelligt. Attribute die in eckigen Klammern stehen bedeuten im Angular Kontext immer, dass etwas an die entsprechende Komponente übergeben wird.
Die Komponente die die Allgemeinen Informationen anzeigt, muss benachrichtigt werden, wenn sich die Informationen ändern. Um dies zu bewerkstelligen, stellt die Bearbeitende Komponente einen Emitter bereit. Der `(submitEmitter)` erwartet eine Methode die aufgerufen werden kann, wenn das Formular abgeschickt wurde. In diesem Beispiel wird die `onSave` Methode aufgerufen, um die veränderungen zu verwalten, und die bearbeitende

Komponente wieder auszublenden. Das bedeutet, dass Attribute in normalen Klammern eine Möglichkeit bieten von der entsprechenden Komponente etwas nach außen zu geben.

```
1  <app-common-information-edit (submitEmitter)="onSave($event)" *ngIf="edit"
2                  [commonInformation]="commonInformation"></app-common-
                       information-edit>
```

Auflistung 4.13: Template Code zum einbinden der CommonInformationEditComponent

Beide Klammertypen sind miteinander kombinierbar, sodass sich mehrere Komponenten die leichen Variablen Teilen können.

5 Evaluierung

In diesem Kapitel wird das entwickelte System betrachtet. Es wird dabei betrachtet wie Tests genutzt wurden um sicherzustellen, dass die gewünschten Funktionalitäten umgesetzt wurden. Es wird abschließend betrachtet was für Probleme es in der Entwicklung des System gab.

5.1 Tests

5.1.1 Backend Tests

Als Teil der Umsetzung des Backends wurden auch automatisierte Tests, die die Funktionalität des Servers prüfen, entwickelt. Diese Tests beschleunigen längerfristig die Entwicklung eines Systems, da man zum einen nicht manuell testen muss, und zum anderen, bei guten und ausreichend vielen Tests, schnell sehen kann, wo genau sich Fehler eingeschlichen haben. Dies hat unter anderem zur Folge, dass man keine Angst haben muss etwas zu refactoren, weil man sicher sein kann, dass alles danach weiterhin funktioniert, wenn die Tests funktionieren. Der Server wurde durch PHPUnit Tests getestet. PHPUnit ist die standard Testbibliothek im PHP Bereich.
Alle Tests sind integrations Tests, das bedeutet sie rufen einen API Endpunkt auf, und erwarten ein bestimmtes Ergebnis. Um die Tests einfacher zu gestalten, arbeitet jeder Test auf der gleichen Datenbasis. Dies wird sichergestellt, indem eine vordefinierte Datenmenge vor jeden Test in eine SQLite Datenbank geschrieben wird. Dadurch wird ausgeschlossen dass einzelne Tests sich gegenseitig beeinflussen, egal in welcher reihenfolge sie ausgeführt werden.
Während die echte Anwendung ihre Nutzer sich gegen einen firmeninternen LDAP-Server authentifizieren lässt, wird beim Testen ein selbstgeschriebener Service benutzt der lediglich drei Benutzer kennt, die einfach zum Testen benutzt werden können. Diese drei nutzer spiegeln Leute mit unterschiedlichen Berechtigungen wieder, so kann einfach getestet werden, ob die Funktionalität und die Autorisierung richtig funktionieren.
Der Ablauf eines Tests soll anhand des Beispiels der Profilsuche erläutert werden.

```
1  $client = $this->makeClient();
2  // unauthorized search should return an 401
3  $client->request('GET', '/api/profiles/search.json');
4  $this->assertStatusCode(401, $client);
5
6  $this->login($client);
7  // Empty search should return an empty array
8  $client->request('GET', '/api/profiles/search.json');
9  $this->assertStatusCode(200, $client);
10 $this->assertEquals([], json_decode(
11     $client->getResponse()->getContent(), true
12 ));
13 // Searching nonexistent string should return empty values
14 $client->request('GET',
15     '/api/profiles/search.json?term=dsjkhfkjsdhkoisdhfdskjsdfk'
16 );
17 $this->assertStatusCode(200, $client);
18 $this->assertEquals([], json_decode(
19     $client->getResponse()->getContent(), true
20 ));
21
22 // Search should be case insensitive
23 $client->request('GET', '/api/profiles/search.json?term=alex');
24 $this->assertStatusCode(200, $client);
25 $json = json_decode($client->getResponse()->getContent(), true);
26 $this->assertAlex($json);
```

Auflistung 5.1: Test der Suchfunktion im Backend

Hierfür soll der code aus 5.1 dienen

1. Man erstellt einen Client mit dem man die Anwendung aufrufen kann. (Zeile 1)
2. Mit diesem Client ruft man den Endpunkt der Suche auf (Zeile 3)
3. Man überprüft die Antwort des Servers ob sie richtigerweise einen 401 Fehler zurückgibt, weil man nicht angemeldet ist. (Zeile 4)
4. Mit dem Client schickt eine Anfrage um sich einzuloggen. (Zeile 6)
5. Anschließend wird erneut der Endpunkt der Suche aufgerufen, ohne einen Suchbegriff anzugeben. (Zeile 8)
6. Man überprüft dass die Antwort keinen Fehlercode enthält und prüft ob sie ein leeres Ergebnis liefert. (Zeile 9ff)

7. Anschließend wird erneut der Endpunkt der Suche mit einem Suchbegriff aufgerufen der keine Ergebnisse liefern soll. (Zeile 14ff)
8. Man überprüft dass die Antwort keinen Fehlercode enthält und prüft ob sie ein leeres Ergebnis liefert. (Zeile 17ff)
9. Anschließend wird der Endpunkt mit einem Suchbegriff aufgerufen. (Zeile 23)
10. Man überprüft dass die Antwort keinen Fehlercode enthält und prüft ob sie die Ergebnisse hat, die man erwartet. (Zeile 24ff)

5.1.2 Frontend Tests

Bei Angular Projekten gibt es grundsätzlich zwei Typen von Test.

Komponenten Tests Diese Tests testen immer nur einzelne Komponenten. Alle Abhängigkeiten die eine Komponente benutzt, werden gemockt, also durch Dummies ersetzt.

End-to-End Tests Diese Tests stellen in der Regel Abläufe einer User-Story nach. Sie werden hierbei in einem Browser durchgeführt, mit dem ein echter Benutzer imitiert wird. So werden Eingaben und Klicks nachgeahmt, und die Seite auf Rückmeldungen und darstellungen geprüft.

Tests werden in der Jasmine [23] Syntax geschrieben.

```
1  describe('VariableTest', function (){
2      var a;
3      it('should check true = true', function() {
4        a = true;
5        expect(a).toBe(true);
6      });
7  });
```

Auflistung 5.2: Ein einfacher Beispieltest in Jasmine

Diese form ist in Auflistung 5.2 zu sehen. Ein Test fängt an mit `describe` dort steht der Name der Testgruppe. Anschließend werden Variablen erstellt die man über mehrere Tests hinweg nutzen möchte. Anschließend wird beschrieben was `it` machen soll. In dem Test wird logik ausgeführt, und Ergebnisse überprüft.
In Codebeispiel 5.3 ist Teil eines realen Tests zu sehen. In Zeile 1 ist beschrieben wie die Gruppe der folgenden Tests heißen soll. In Zeile 4 startet der Test, der Prüft ob man sich ein und ausloggen kann. Dabei wird in Zeile 7 ein Mock-Objekt erstellt welches auf die methode `post`

reagieren soll. In Zeile 9 wird definiert was passieren soll, wenn die post-Methode aufgerufen wird. Dieser MockService kann nun dank Dependency Injection an den AuthenticionService übergeben werden, sodass man ihn gekapselt testen kann. Im folgenden werden Operationen ausgeführt, und deren Ergebnisse getestet. Bei diesem Test handelt es sich um einen Komponenten Test, oder auch Unit-Test.

```
1  describe('AuthenticationService', () => {
2    let authService: AuthenticationService;
3
4    it('should login and logout', (done: DoneFn) => {
5      const token = 'eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.....';
6      // create 'post' spy on an object representing the HttpClient
7      const httpServiceSpy = jasmine.createSpyObj('HttpClient', ['post']);
8      // set the value to return when the 'post' spy is called.
9      httpServiceSpy.post.and.returnValue(of({token: token}));
10
11     const msg = {};
12     authService = new AuthenticationService(httpServiceSpy, msg as
           MessageService);
13     expect(authService).toBeTruthy();
14     authService.login('foo', 'bar').subscribe(value => {
15       expect(value).toBe(token, 'Login returned token');
16       done();
17     });
18   });
19 });
```

Auflistung 5.3: Teil des Tests des Authentication Services

End-to-End Tests sind nicht als Teil dieser Bachelor-Arbeit umgesetzt, und werden in einem nächsten Release umgestetzt. Diese Tests werden mithilfe von Protractor [42] ausgeführt. Protractor wird genutzt um Tests in einem echten Browser auszuführen. Protractor benutzt dafür wird Selenium [46] und ist auf Angular ausgelegt. Dies bedeutet, dass es Angular spezifische Anweisungen gibt, die das Testen erleichtern.

5.2 Probleme

Bei der analyse des Altsystems sind einige Punkte aufgetaucht, dessen Sinnhaftigkeit sich nicht direkt ergeben hat. Dadurch musste viel Zeit investiert werden um sicher zu stellen, dass

wirklich alle Funktionen festgehalten werden. So gab es beispielsweise Artefakte wie die Rolle Hangaround (siehe Kapitel 2.2.1), die keinerlei auswirkungen haben.
Der Export der Profile war ebenfalls problematisch. Die genutzten Templates wurden wurden mit Apache Freemarker [8] erstellt. Freemarker ist eine kostenlose Java-basierte Templating Sprache. Es ist daher nicht einfach möglich diese Templates in einem PHP-Basierten Projekt zu benutzen. Daher war es nötig im Detail den Quellcode des alten Systems zu durchforsten, und den entsprechenden Code so zu kapseln, dass man ein wiederverwertbares Tool bekommt. Dieses Programm ist durch mangelndes Wissen im Grails Kontext wahrscheinlich viel zu groß und wäre einfacher umsetzbar gewesen. Zusätzlich hat diese Lösung das Problem, dass die Laufzeit, von Anfrage am API-Endpunkt bis Antwort an den Client, ziemlich groß ist.
Ein weiteres Problem während der Umsetzung war die Einarbeitung in die Docker-Welt. Während es einfach ist schnell mit Docker produktiv zu werden, sind hier viele Fallstricke die bei Detailfragen zu Problemen führen können. So war es hier, durch fehlende Expertise nicht möglich den PHP-Debugger XDebug zu nutzen, weil die Konfiguration zu komplex war. Zusätzliche Probleme kamen durch die Entscheidung auf als Basis für die Docker-Container Alpine Linux zu benutzen. Da Alpine Linux [8] andere Pakete benutzt als ein Debian basiertes Linux wie Beispielsweise Ubuntu. Explizit gab es hierbei bei der Bibliothek ImageMagick [1] in PHP, weil die unterschiedlichen Pakete dafür sorgten, dass es nicht einfach einzubinden war. Alpine Linux hat den Vorteil dass es deutlich kleiner ist als übliche Debian Distributionen, und dadurch kleinere Docker Images produziert. Durch die Probleme wurde jedoch davon abgesehen Alpine Linux als Grundlage zu nutzen.
Grundsätzlich gab es ebenfalls Probleme bei der Umsetzung des Clients mit Angular, wegen der Nutzung der RxJS Bibliothek. Diese erfordert ein umdenken vom klassischen imperativen objektorientierten Paradigma zu einem Deklarativen.
Ein Problem welches am Ende der Entwicklung deutlich geworden ist, ist das Fehlen von End-To-End Tests. Es wurden einzelne Komponenten im Client abgetestet , ebenso die Endpunkte der Schnittstelle. Jedoch wurde das System als ganze Einheit nicht automatisiert getestet. Dadurch ist nicht durch Tests sichergestellt, dass User Stories korrekt umgesetzt sind.

6 Schluss

In diesem Kapitel werden die Ergebnisse und gelernten Erkenntnisse zusammengefasst. Abschließend werden Möglichkeiten beschrieben um das System weiterzuentwickeln.

6.1 Zusammenfassung

In dieser Arbeit wurde ein Informationssystem entwickelt, das ein bestehendes System innerhalb des Unternehmens Silpion IT Solutions ablösen soll.
Es wurde das bisher bestehende System analysiert und dessen Aufgaben festgehalten. Der Funktionsumfang des neu entwickelten Systems ist dabei der Gleiche geblieben. Das neue System benutzt jedoch modernere, und bei Silpion weiterverbreitete Technologien. Dadurch ist eine weitere Entwicklung neuer Funktionen besser möglich.
Das Analyse Kapitel untersucht das bestehende System, und alle Aufgaben die das System erfüllt in User Stories festgehalten. Es wurde ebenfalls die technische Umsetzung des Systems betrachtet und Probleme dabei aufgezeigt. Es wurde außerdem das bestehende Datenmodell beschrieben. Anschließend wurde abgewägt wie sinnvoll es ist, das bestehende System durch ein Neues zu ersetzen
Es wurde ein Softwaredesign für eine Umsetzung des Systems entwickelt. Dabei wird eine strikte Aufteilung des Systems in zwei Anwendungen, REST-API und Client, eingeführt. Es wird die Kommunikation zwischen Client und API betrachtet und wie diese sicher gestaltet werden kann. Das Backend wird dabei unabhängig vom Client entwickelt und definiert eine Schnittstelle, die beliebige Clients benutzen können. Es wurden die Komponenten von Backend und Client erläutert, und interne Abläufe mit Hilfe von beispielhaften Szenario in Sequenzdiagrammen aufgezeigt. Abschließend wurden die verwendeten Frameworks vorgestellt.
Für die Umsetzung relevante Komponenten, Methoden zur Qualitätssicherung wurden vorgestellt. Es wurde die Programmstruktur anhand von Beispielen des Quellcodes erklärt. Es wurde erklärt welche Typen von Tests bestehen, und wie diese umgesetzt wurden.

6.2 Fazit

Als Teil dieser Arbeit ist ein Informationssystem entstanden, welches einen bereits bestehendes, funktionierendes Werkzeug ablösen soll. Die Neuentwicklung des Systems bietet bereits einen Mehrwert an. Es wurde eine aktuelle Technologie auf aktuellen Plattformen genutzt, sodass die neue Version von Sicherheitsupdates profitiert. Da eine Architektur benutzt wurde, die sehr auf Kapselung achtet, und Technologien benutzt werden, die weit Verbreitet sind, entsteht außerdem ein Mehrwert darin, dass die Anwendung von nun an einfacher Erweiterbar ist
Das alte System hat das Problem, dass es nicht sehr gerne von Mitarbeiterinnen benutzt wird. Falls die Mitarbeiterinnen durch ein neues, moderneres Frontend mehr mit dem Tool interagieren ist ein zusätzlicher Mehrwert geschaffen.
Jedoch ist das in dieser Arbeit entwickelte System nur als erster Schritt in einem länger laufenden Projekt zu verstehen. In weiteren Schritten soll der Nutzen der Plattform für Endbenutzer gesteigert werden indem neue Funktionen implementiert werden.

6.3 Ausblick

In diesem Abschnitt wird ein Ausblick auf die Zukunft des Systems ermöglicht.
Bevor dieses neue System genutzt werden kann, sollte es ausgiebig getestet werden. Dafür kann man das neue System parallel zum Alten bereitstellen und die Mitarbeiter für einen Zeitraum ihre Profile im neuen System pflegen lassen. So sollte Fehlverhalten, das nicht durch Tests entdeckt wurde entdeckt werden.
Der entwickelte Client ist in seinem Design sehr minimal gehalten wurden. Dadurch ist es möglich, dass die Benutzbarkeit an Stellen gelitten hat. Um ein befriedigendes Benutzererlebnis sicher zu stellen, ist es sinnvoll einen Experten für User-Experience das Frontend analysieren zu lassen.
Die Entwicklung eines neuen Systems ist die Grundlage für eine Weiterentwicklung mit neuen Funktionalitäten oder Verbesserungen. Da seit Jahren nicht mehr an dem alten System gearbeitet wurde, haben sich viele Wünsche nach neuen Funktionalitäten aufgestaut. Außerdem gibt es Ideen wie das System technisch weiterhin verbessert werden kann.

6.3.1 Weitere Funktionalitäten

Die Ideen für neue Funktionalitäten sind zahlreich, daher werden hier nur ein paar Wünsche vorgestellt.
Es ist gewünscht den Client so zu designen, dass man durch wenig Aufwand den Betreiber des Systems tauschen kann. So soll es möglich sein, das man Themes erstellt, die für unterschiedliche Kunden genutzt werden können. Das soll es ermöglichen mit wenig Aufwand ein Produkt zu erschaffen, dass man auch verkaufen kann.
Es ist gewünscht eine Funktionalität zu erstellen, die es ermöglicht die Templates für Profilexporte über ein Webinterface zu pflegen. Aktuell sind die Templates als Dateien fest im Code hinterlegt, und nur durch neue Releases änderbar. Wenn man es ermöglicht Templates in dem System zu hinterlegen, und diese durch Designer pflegen zu lassen, ist es einfacher eine aktuelle Corporate Identity zu präsentieren.
Da Mitarbeiter oft vergessen ihre Projekte oder neu gelernten Fertigkeiten zu pflegen, wurde es gewünscht E-Mail Benachrichtigungen zu verschicken. So soll, nach einem Bestimmten Zeitraum nachdem ein Benutzer sein Profil nicht bearbeitet hat, eine E-Mail ausgelöst werden die den Nutzer auffordert sein Profil zu aktualisieren.
Es wurde gewünscht eine Funktion für Abteilungsleiter zu entwickeln die es vereinfacht Teams zusammenzustellen. Die Idee dahinter ist, dass man angibt wie viele Teammitglieder man benötigt und welche Fertigkeiten gebraucht sind. Anschließend soll das System Teams vorschlagen die die notwendigen Fertigkeiten abdecken. Dabei soll es möglich sein, anzugeben ob jedes Teammitglied diese Fertigkeit besitzen soll, oder nur eine bestimmte Anzahl Personen. Mit dieser Funktion können Abteilungsleiter schnell feststellen ob es genug Kompetenzen gibt um ein Projekt zu bearbeiten.
Ein weiteres großes Feature das gewünscht ist, ist das sogenannte *Tech-Radar*. Bei der SkillDB geben Mitarbeiterinnen an wie gut sie etwas können. Das kann zur Folge haben, dass Mitarbeiterinnen Fertigkeiten die sie besitzen aber nicht gerne benutzen, nicht angeben. So kann es dazu kommen, dass Entwickler zwar Assembler Code schreiben können, aber dies nicht gerne machen und daher diese Fähigkeit nicht angeben. Das Tech-Radar soll dort Abhilfe schaffen. Hierdurch sollen Benutzerinnen ihre Fertigkeiten nicht nur nach Kompetenz pflegen sondern auch sagen was sie davon halten. Andere Nutzer können dann die Meinungen zu einzelnen Fertigkeiten sehen und sich ein Bild darüber machen, was andere Mitarbeiter darüber denken.

6.3.2 Technische Weiterentwicklungen

In diesem System gibt es weitere Möglichkeiten die Technische Umsetzung zu verbessern. Ein paar dieser Möglichkeiten sollen hier vorgestellt werden.
Der Client besitzt keine Frontend, oder End-to-End Tests. Dies erschwert es sicher zu sein, dass ein neues Feature alte Funktionalitäten nicht behindern. End-To-End tests bieten eine gute Möglichkeit zu prüfen ob User Stories erfolgreich abgeschlossen sind. Daher ist es sinnvoll diese Tests in einem nächsten Release umzusetzen.
Das System benutzt aktuell einen LDAP-Server um die Benutzer zu authentifizieren. Dies bedeutet, dass Anmeldedaten vom Client an die REST-API geschickt werden und von dort aus an den LDAP-Server. In einem nächsten Schritt ist es möglich das neu eingeführte Keycloak System zu nutzen, um ein Single-Sign-On zu implementieren, und Zugangsdaten nicht mehr an die REST-Schnittstelle zu schicken.
Es ist denkbar die Applikation performanter zu gestalten indem man HTTP/2 in Frontend und Backend nutzt. Dies erlaubt es dem Frontend wenn es eine Resource an der API abfragt, mehrere Resourcen gleichzeitig abzufragen. Dabei schickt das Frontend eine Anfrage an den Endpunkt um ein Profile abzufragen, und teilt dem Server gleichzeitig mit, dass es zusätzlich noch die Daten von anderen Endpunkten haben möchte, die es benötigt. Der Server antwortet dann mit den Daten des Profils und schickt anschließend in weiteren Antworten die weiteren angefragten Objekte. dies ermöglicht es die Anzahl der Verbindungen zu reduzieren, was schnell ein Flaschenhals werden kann.
Es ist möglich die Containerisierung des Systems weiter voran zu bringen, indem die Container so aufgebaut werden, dass sie in einem System zur Skalierung, Bereitstellung und Verwaltung von Containern benutzt werden können. Dafür ist es möglich Kubernetes zu nutzen, und bei hoher Last automatisch weitere Backend Server zu starten. Dies ermöglicht es auch die Applikation über eine Cloud bereitzustellen.

Literaturverzeichnis

[1] : *ImageMagick php.net* . – URL https://www.php.net/manual/en/book.imagick.php. – Zugriffsdatum: 2020-01-30

[2] : *Action Domain Responder*. – URL http://pmjones.io/adr/. – Zugriffsdatum: 2020-01-30

[3] : *Agile Alliance User Story Template*. – URL https://www.agilealliance.org/glossary/user-story-template/. – Zugriffsdatum: 2020-01-30

[4] : *Angular Homepage*. – URL https://angular.io/. – Zugriffsdatum: 2020-01-30

[5] : *angular Lizenz*. – URL https://github.com/angular/angular/blob/master/LICENSE. – Zugriffsdatum: 2020-01-30

[6] : *Angular Material Design Homepage*. – URL https://material.angular.io. – Zugriffsdatum: 2020-01-30

[7] : *AngularJS Homepage*. – URL https://angularjs.org/. – Zugriffsdatum: 2020-01-30

[8] : *Apache Freemarker*. – URL https://www.alpinelinux.org/. – Zugriffsdatum: 2020-01-30

[9] : *Apache homepage*. – URL http://httpd.apache.org/. – Zugriffsdatum: 2020-03-05

[10] : *Apache Lizenz 2.0*. – URL https://github.com/apache/httpd/blob/trunk/LICENSE. – Zugriffsdatum: 2020-11-15

[11] : *Api Platform*. – URL https://api-platform.com/. – Zugriffsdatum: 2020-01-30

[12] : *Bundeszentrale für politische Bildung - Wirschaftswachstum.* – URL https://www.bpb.de/nachschlagen/lexika/lexikon-der-wirtschaft/21136/wirtschaftswachstum. – Zugriffsdatum: 2020-01-30

[13] : *Composer Homepage.* – URL https://getcomposer.org. – Zugriffsdatum: 2020-01-30

[14] : *Desktop vs Mobile vs Tablet Market Share.* – URL https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/#monthly-201201-202010. – Zugriffsdatum: 2020-01-30

[15] : *docker Homepage.* – URL https://www.docker.com/. – Zugriffsdatum: 2020-03-05

[16] : *Doctrine Homepage.* – URL https://www.doctrine-project.org. – Zugriffsdatum: 2020-01-30

[17] : *DQL Dkumentation.* – URL https://www.doctrine-project.org/projects/doctrine-orm/en/2.7/reference/dql-doctrine-query-language.html/. – Zugriffsdatum: 2020-01-30

[18] : *Grails 3.0 Dokumentation.* – URL http://www.erlang.org/doc/. – Zugriffsdatum: 2020-01-30

[19] : *Grails Homepage.* – URL https://grails.org/. – Zugriffsdatum: 2020-01-30

[20] : *Groovy Homepage.* – URL http://groovy-lang.org/. – Zugriffsdatum: 2020-03-05

[21] : *HTTP/1.1 RFC Methoden.* – URL https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html. – Zugriffsdatum: 2020-03-05

[22] : *HTTP/1.1 RFC Statuscode.* – URL https://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html. – Zugriffsdatum: 2020-03-05

[23] : *Jasmine Homepage.* – URL https://jasmine.github.io/. – Zugriffsdatum: 2020-01-30

[24] : *JSON-HAL Draft.* – URL https://tools.ietf.org/html/draft-kelly-json-hal-08. – Zugriffsdatum: 2020-03-05

[25] : *JSON-LD Spezifikation*. – URL https://www.w3.org/TR/json-ld/. – Zugriffsdatum: 2020-03-05

[26] : *JSON RFC*. – URL https://tools.ietf.org/html/rfc8259. – Zugriffsdatum: 2020-03-05

[27] : *JSON Web Token (JWT) RFC*. – URL https://tools.ietf.org/html/rfc7519. – Zugriffsdatum: 2020-03-05

[28] : *Kubernetes homepage*. – URL https://kubernetes.io/. – Zugriffsdatum: 2020-03-05

[29] : *Lightweight Directory Access Protocol (LDAP): The Protocol*. – URL https://tools.ietf.org/html/rfc4511. – Zugriffsdatum: 2020-03-20

[30] : *Material Design Homepage*. – URL https://material.io. – Zugriffsdatum: 2020-01-30

[31] : *Mozilla Developers - Mobile First*. – URL https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Responsive/Mobile_first. – Zugriffsdatum: 2020-01-30

[32] : *NGINX Homepage*. – URL https://www.nginx.com/. – Zugriffsdatum: 2020-03-05

[33] : *NodeJS Homepage*. – URL https://nodejs.org/. – Zugriffsdatum: 2020-01-30

[34] : *packagist Homepage*. – URL https://packagist.org/. – Zugriffsdatum: 2020-01-30

[35] : *PATCH Method for HTTP*. – URL https://tools.ietf.org/html/rfc5789. – Zugriffsdatum: 2020-03-05

[36] : *PHP Homepage*. – URL https://www.php.net/. – Zugriffsdatum: 2020-03-05

[37] : *PHP Market Share and Competitor Report*. – URL https://www.datanyze.com/market-share/programming-languages--67/Alexa%20top%201M/php-market-share. – Zugriffsdatum: 2020-04-02

[38] : *PHP Repository*. – URL https://github.com/php/php-src/. – Zugriffsdatum: 2020-03-05

[39] : *PHP.net - Was ist PHP?*. – URL https://www.php.net/manual/de/intro-whatis.php. – Zugriffsdatum: 2020-03-05

[40] : *PostgreSQL Homepage*. – URL https://www.postgresql.org/. – Zugriffsdatum: 2020-03-05

[41] : *PostgreSQL nicht unterstützte SQL Features*. – URL https://www.postgresql.org/docs/current/unsupported-features-sql-standard.html. – Zugriffsdatum: 2020-11-15

[42] : *Protractor Homepage*. – URL https://www.protractortest.org/. – Zugriffsdatum: 2020-01-30

[43] : *React Homepage*. – URL https://reactjs.org/. – Zugriffsdatum: 2020-01-30

[44] : *REST - statelessness - Restful API Tutorial*. – URL https://restfulapi.net/statelessness/. – Zugriffsdatum: 2020-03-05

[45] : *RxJS Dokumentation*. – URL https://rxjs.dev/guide/overview. – Zugriffsdatum: 2020-01-30

[46] : *Selenium Homepage*. – URL https://www.selenium.dev/. – Zugriffsdatum: 2020-01-30

[47] : *Sonatype Aether Github*. – URL https://github.com/sonatype/sonatype-aether. – Zugriffsdatum: 2020-01-30

[48] : *Symfony Downloads Stats*. – URL https://symfony.com/stats/downloads. – Zugriffsdatum: 2020-04-02

[49] : *The OAuth 2.0 Authorization Framework*. – URL https://tools.ietf.org/html/rfc6749. – Zugriffsdatum: 2020-03-20

[50] : *Typescript Homepage*. – URL https://www.typescriptlang.org/. – Zugriffsdatum: 2020-01-30

[51] : *Usage of web servers broken down by ranking*. – URL https://w3techs.com/technologies/cross/web_server/ranking. – Zugriffsdatum: 2020-03-05

[52] : *Vue JS Einführung*. – URL https://v3.vuejs.org/guide/introduction.html#composing-with-components/. – Zugriffsdatum: 2020-01-30

[53] : *Vue JS Homepage*. – URL https://vuejs.org/. – Zugriffsdatum: 2020-01-30

[54] : *w3techs Webservernutzungsstatistik*. – URL https://w3techs.com/technologies/overview/web_server. – Zugriffsdatum: 2020-11-15

[55] : *Webkomponenten Spezifikation*. – URL https://www.webcomponents.org/specs. – Zugriffsdatum: 2020-01-30

[56] : *What is LDAPS*. – URL https://www.jerichosystems.com/technology/glossaryterms/lightweight_directory_access_protocol_over_secure_socket_links.html. – Zugriffsdatum: 2020-03-20

[57] : *XML Spezifikation*. – URL https://www.w3.org/TR/REC-xml/. – Zugriffsdatum: 2020-03-05

[58] Silpion IT Solutions GmbH: *Silpion | Das sind wir. Das macht uns aus.* 2017. – URL https://www.silpion.de/wir-sind-silpion. – Zugriffsdatum: 2019-07-16

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Analyse und Neuentwicklung eines bestehenden webbasierten Kompetenzmanagementsystems

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original