Abschlussprüfung Winter 2016

Fachinformatiker für Anwendungsentwicklung

Dokumentation zur betrieblichen Projektarbeit

Webapplikation zur Verwaltung der Zugriffsrechte der Benutzeraccounts externer Services

Webbasiertes Tool zur Unterstützung der Entwickler

Abgabetermin: Nürnberg, den 15.12.2016

Prüfungsbewerber:
Farah Schüller
Straße 123
1337 Stadt
Prüflingsnummer: 23057

Ausbildungsbetrieb:
SUSE Linux GmbH
Maxfeldstraße 5
90409 Nürnberg

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
MVC	Model View Controller
SQL	Structured Query Language
STI	Single Table Inheritance
RPM	RPM Package Manager
RoR	Ruby on Rails
TDD	Test Driven Development
LDAP	Lightweight Directory Access Protocol
JSON	JavaScript Object Notation

1 Einleitung

1.1 Projektumfeld

Die SUSE Linux GmbH wurde 1992 gegründet und ist seit 2014 eine Tochtergesellschaft der Micro Focus PLC. Weltweit beschäftigt das Unternehmen etwa 750 Mitarbeiter, welche auf die Standorte Nürnberg, Prag, Peking und Provo (USA) verteilt sind. Der Hauptsitz der Firma befindet sich in Nürnberg, ebenso wie der Hauptteil der Softwareentwicklung.

Das Kerngeschäft der SUSE Linux GmbH umfasst die Entwicklung einer Linux-basierten Distribution. Für Privatkunden werden hierzu die Distributionen der openSUSE-Familie, die größtenteils von der Community entwickelt werden, bereitgestellt, für Geschäftskunden die Produkte der SUSE Linux Enterprise-Familie.

Die Entwicklung erfolgt nach dem Open-Source-Prinzip. Durchgeführt wurde das Projekt im Team SUSE-IT, die neben der Infrastruktur auch die Entwicklungsumgebung, die sogenannten Engineering Services, bereitstellen.

1.2 Projektbeschreibung

Zur Entwicklung der Open-Source-Software setzt die SUSE Linux GmbH in Teilen der Entwicklung auf externe Tools und Services ein. Dies wird besonders im Bereich der Codeentwicklung und -pflege deutlich, da neben den firmeneigenen Entwicklern auch Partner sowie Open-Source-Community-Mitglieder mitarbeiten. Hierbei kann es zu der Diskrepanz kommen, dass Entwickler neben ihrem Firmenaccount auch private Accounts zur Entwicklung benutzen, wodurch eine genaue Zuordnung der Accounts und ein Überblick über Zugriffsberechtigungen nahezu unmöglich wird.

Zur Lösung dieser Problematik soll eine Applikation entwickelt werden, welche es ermöglicht eine Zuordnung der firmeninternen Accounts auf mögliche Accounts externer Services durchzuführen. Das Ergebnis soll mittels einer API abrufbar sein. Dabei soll die Applikation in ihrer Grundfunktion dem Gedanken der *Self-Service-Technologies* folgen, damit Latenzen bei Änderungen der Accountinformationen minimiert und administratives Personal entlastet wird. Um dies möglichst plattformunabhängig zu gestalten, erfolgt die Realisierung als Webapplikation.

Da externe Services eine Zulassung durch den Betriebsrat und die IT-Sicherheit benötigen, beschränkt sich die erste Version dieser Applikation darauf, die Accountinformationen der Applikationen Github (verzeichnisorganisierte Versionsverwaltung von Quellcode) und Trello (Tool zur Steuerung von agilen Software-Projekten) einzubinden. Diese erhielten bereits im Vorfeld dieser Arbeit eine Zulassung.

1.3 Ist-Analyse

Bei den verwendeten externen Services Github und Trello können Unternehmen zur Abgrenzung und Verwaltung ihrer firmeninternen Ressourcen und Informationen Organisationen anlegen, zu denen Accounts von Mitarbeitern hinzugefügt werden und ihnen besondere Zugriffsrechte einzuräumen. So haben beispielsweise nur Mitarbeiter Schreibrechte auf den Quellcodeverzeichnissen bei Github und Mitglieder der Open-Source-Community sowie Partner können in der Regel nur Vorschläge, sogenannte Pull Requests, bei dem jeweiligen Projekt einreichen.

Der Abgleich der internen Mitarbeiter- mit den Userlisten innerhalb der angelegten Firmenorganisationen erfolgt bisher manuell. Dies bedeutet, dass beispielsweise für jeden neuen Mitarbeiter persönlich oder schriftlich angefragt werden muss welches User-Alias er bei den jeweiligen Services führt, um dessen Account dann zur Organisation hinzuzufügen und ihm Zugriff zu ermöglichen. Ist dieser Aufwand bereits groß, erhöht er sich bei Verlassen eines Mitarbeiters der Firma, wenn es gegebenenfalls auf Grund fehlender Kontaktmöglichkeit sehr mühsam wird eine Zuordnung nachzuvollziehen.

Bei Organisationen mit mehreren hundert Mitgliedern entsteht dadurch ein großer zeitlicher und personeller Aufwand. Durch die ungenaue Beschaffung der benötigten Informationen ist das bisherige Konzept sehr fehleranfällig und ineffizient durchzuführen.

1.4 Soll-Konzept

Wie in 1.3 aufgezeigt benötigt der bisherige Prozess viel Zeit und ist kompliziert durchzuführen. Ziel ist die Vereinfachung des Prozesses und die Entlastung des Personals. Die Lösung dafür stellt eine gesammelte Zuordnung der internen Mitarbeiterdaten zu ihren jeweiligen extern Nutzerkonten dar, über welche der Abgleich leichter gestaltet werden kann.

Zur Abfrage der mitarbeiterspezifischen Informationen, wie Name, Standort oder Vorgesetzter, kann das firmeneigene eDirectory genutzt werden.

Die Personalbelastung kann minimiert werden, in dem die Mitarbeiter selbst die jeweils benötigten Informationen eintragen und verwalten. Dadurch bleibt alleine der Abgleich mit den Nutzerlisten der bei den externen Services angelegten Organisationen und ein eventuelles Hinzufügen oder Entfernen des Mitarbeiters aus der Organisation übrig.

Um dem Mitarbeiter Möglichkeit zur Eigeninitiative einzuräumen, können die Schnittstellen der genannten externen Tools Github und Trello genutzt werden, um bei Hinzufügen der Nutzerinformation die Organisationszugehörigkeit abzufragen. Durch eine Abfrage im Hintergrund und Anzeige eines passenden Hinweises kann der jeweilige Mitarbeiter den Administratoren selbst eine Anfrage schicken.

Um Erreichbarkeit und einfache Bedienbarkeit zu gewährleisten, bietet sich die Umsetzung als Webapplikation an. Um diese leicht wartbar und für zukünftige Funktionen erweiterbar zu halten, soll ein

modulares Framework genutzt werden. Die Entwicklung einer zugehörigen Testsammlung ermöglicht die spätere Nachvollziehbarkeit der Entwicklungsschritte.

Die Schnittstelle der Applikation, welche die gesammelten Informationen über den Mitarbeiter bereitstellt, wird in dem Administrationstool etsync konsumiert, welches für die Administratoren der externen Organisationen bei Github und Trello entwickelt wurde.

2 Projektplanung

2.1 Projektphasen

Für das Projekt wurde folgende Zeitplanung aufgestellt:

Analysephase		**5 h**
Analyse des Ist-Zustands	2 h	
Formulierung des Soll-Zustands	3 h	
Projektplanung		**10 h**
Planung der Entwicklungs- und Datenmodelle	7 h	
Planung des Layouts der Benutzeroberfläche	1 h	
Planung der Paketierung	2 h	
Implementierungsphase		**42 h**
Entwicklung des Backends	20 h	
Implementierung der Benutzeroberfläche	3 h	
Anbindung externer Schnittstellen	5 h	
Programmierung der internen Schnittstelle	2 h	
Testphase	5 h	
Paketierung	5 h	
Deployment	2 h	
Einführungsphase		**3 h**
Projektabnahme	2 h	
Firmenweite Einführung	1 h	
Erstellen der Dokumentation		**10 h**
Gesamt		**70 h**

Tabelle 1: Zeitplanung

2.2 Ressourcenplanung

Die Planung der benötigten Ressourcen untergliedert sich in Personal- und Kostenplanung.

2.2.1 Personalplanung

Außer dem Auszubildenden werden keine weiteren Mitarbeiter für das Projekt eingesetzt. Dieser wird dafür von seiner regulären Mitarbeit im Team freigestellt.

2.2.2 Kostenplanung

Die Personalkosten belaufen sich auf die Ausbildungsvergütung für den Zeitraum des Projekts.

Als operative Kosten fallen lediglich die Stromkosten für die zur Projekterstellung eingesetzte Hardware an. Es wird keine zusätzliche Hardware zur Entwicklung benötigt. Durch die ausschließliche Nutzung einer Entwicklungsumgebung und Werkzeugen aus dem Open-Source-Bereich entstehen keine zusätzlichen Kosten wie Software-Lizenzen.

2.3 Entwicklungsprozess

Als Entwicklungsmethode wird *Test Driven Development* (TDD) eingesetzt. Diese Methode sieht vor, dass vor dem Schreiben des eigentichen Quellcodes einer Funktionalität zunächst der dazugehörige Test entwickelt wird. Dadurch wird ein einen abstrakterer und zielorientierterer Blick auf die eigentlichen geforderten Funktionen der Applikation ermöglicht, in dem als erstes eine Überlegung über das Ergebnis einer Funktion und daraufhin die eigentliche Implementierung des jeweiligen Algorithmus erfolgt, welcher zu dem gewünschten Ergebnis führt.

2.4 Architekturdesign

Zur Umsetzung der geforderten Modularität der Applikation bietet sich das sogenannte *Model-View-Controller-Schema* (MVC) an. Dieses ist ein gängiges Schema beim Aufbau von Webapplikationen und besteht aus drei Teilen:

- das *Model*, welches grob die einzelnen Tabellen einer Datenbank repräsentiert. Es enthält in der Regel zusätzliche Logik und Regeln, die zur Verwaltung und Zusammensetzung von Attributen eines Datenbankobjektes notwendig sind.
- der *View*, welcher eine Ausgabe der gewünschten Daten bereitstellt. Im Kontext einer Webapplikation ist dies die eigentlich dargestellte Webseite in *HTML/CSS*.
- der *Controller*, welcher die Schnittstelle zwischen Model und View darstellt. Im Controller werden Eingaben verarbeitet und Befehle an das Model weitergegeben, welche die angeforderten Informationen an den Controller zurückgibt, der diese wiederum an den entsprechenden View verteilt.

Diese Aufteilung ermöglicht ein strukturierte und übersichtliche Entwicklung, da Funktionslogik, Datenbankoperationen und Ausgabelogik klar getrennt und jeweils namentlich zugehörig gekennzeichnet sind.

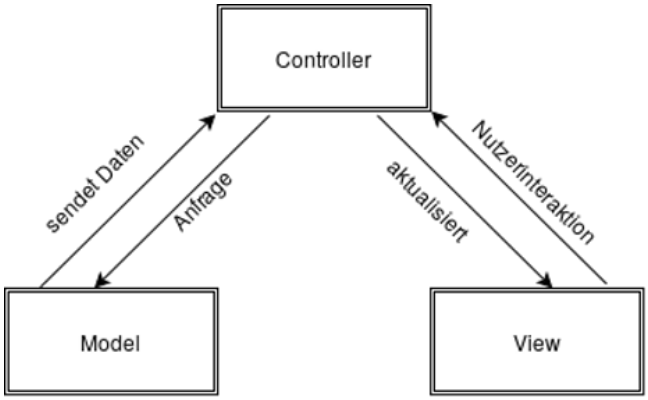

Abbildung 1: MVC-Schema

2.4.1 Wahl der Programmiersprache

Als Programmiersprache wurden *Ruby* und das zugehörige Web-Framework *Ruby on Rails* gewählt.

Ruby basiert auf den Programmiersprachen Perl, Smalltalk, Eiffel, Ada und Lisp und legt Wert auf die Balance von *funktionaler* und *imperativer Programmierung*. So verzichtet Ruby beispielsweise auf Klammern zur Kennzeichnung von Codeblöcken und behandelt jeden Bestandteil des Codes, wie Variablen,als ein *Objekt*, welches eigene Methoden besitzt. Das macht die Verarbeitung und Manipulation von Informationen sehr intuitiv und unterstreicht das Prinzip Rubys, den Programmierer nicht durch Restriktionen und sprachlichen Eigenheiten bei seiner eigentlichen Arbeit zu behindern.

Ruby und sein Framework Ruby on Rails werden quelloffen verwaltet und entwickelt. Neben den existierenden Programmbibliotheken existieren zusätzlich von der Community bereitgestellte Module, sogenannte *Gems*, welche wie kleine zusätzliche Bibliotheken behandelt werden können. Die Einbindung dieser Gems in ein bestehendes Projekt vermeidet Duplikation und erspart viel Entwicklungszeit.

2.5 Benutzeroberfläche

Da nicht auf Mitarbeiter aus dem Team der Webdesigner zurückgegriffen und möglichst zeiteffizient eine für Nutzer verständliche Oberfläche realisiert werden musste, fiel die Entscheidung auf *Twitter Bootstrap*. Das CSS-Framework Twitter Bootstrap wird häufig zur Gestaltung von Webseiten und -applikationen verwendet, da es ein über eine umfangreiche Bibliothek an Gestaltungsvorlagen verfügt und sehr leicht in ein Projekt zu integrieren ist. Auch unerfahrenen Anwendern gelingt damit mühelos ein ansprechende und responsive Nutzeroberfläche.

2.6 Rechteverteilung

Die Applikation soll von zwei verschiedenen Arten von Usern verwendet werden: dem regulären Mitarbeiter, der einzig Zugriff auf seine verwalteten Informationen zu externen Services hat, und einem Administrator, der neben der Bearbeitung seines eigenen Datensatzes auch jene aller in der Datenbank vorhandenen Mitarbeiter einsehen, abfragen und löschen kann.

Diese Rechteverteilung kann über ein *Flag* gelöst werden, was die einfachste Methode ist. Sie ist allerdings aus sicherheitsrelevanter Sicht sehr anfällig und leicht zu manipulieren, weswegen entschieden wurde, die Rechteverteilung auf Datenbankebene zu realisieren.

2.7 Datenmodell

Das Projekt verfügt über zwei Informationsträger, die in der Datenbank abgebildet werden sollen: einen *User* und ein zugehöriges *Tool*.

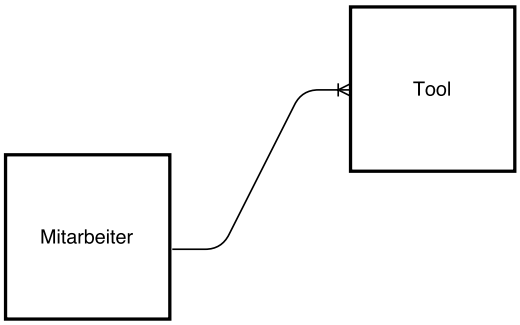

Abbildung 2: Entity-Relationship-Modell

Da für die Applikation zusätzliche Spezialisierungen, wie eine Administrator-Rolle und verschiedene externe Services, notwendig sind, wurden diese Beziehungen mit Hilfe der sogenannten *Single Table Inheritance* (STI) realisiert.

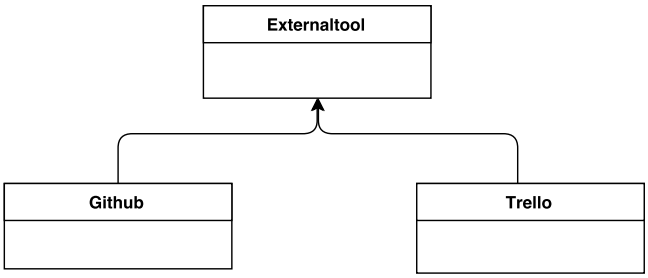

Abbildung 3: Beispielhafte Darstellung der STI

Diese spezielle Vererbungsstrategie lässt sich mit Hilfe der in Rails eingebauten *Active Record*-Engine durch das einfache Hinzufügen eines Typ-Attributs an das Model umsetzen.

2.8 Schnittstellen

Die Hauptaufgabe der Applikation ist die Bereitstellung der gesammelten Informationen zu den Mitarbeitern und deren eingetragenen Informationen zu ihren genutzten Services. Dazu soll eine eigene Schnittstelle innerhalb der Applikation entwickelt werden, welche die Informationen im *JSON*-Format ausgibt. Diese Schnittstelle kann nur authentifiziert angesprochen werden, d. h. der User muss eine Administratorrolle besitzen, um Zugriff zu erhalten.

2.9 Paketierung

Gemäß der Richtline des Teams, in dem die Applikation entwickelt wurde, wird jene als *RPM* paketiert. Dies ermöglicht eine einfache und sofort funktionsfähige Installation, da alle notwendigen Programmabhängigkeiten im Paket definiert und während des Installationsprozesses abgerufen und mitinstalliert werden. Dadurch ist beispielsweise auch bei Bedarf eine problemlose Migration auf einen anderen Server möglich.

3 Implementierungsphase

3.1 Einleitung

Nach der Planungsphase wurde im direkten Anschluss die Implementierungsphase mit folgenden Schritten eingeleitet:

1. Implementierung des Backends (Datenbank, Controllerlogik, Administratorbereich)
2. Implementierung der Benutzeroberfläche (Bootstrap, Tab-Layout)
3. Anbindung externer Schnittstellen (Github API (octokit), LDAP-Authentifizierung (devise))
4. Implementierung der internen Schnittstelle (rabl-gem)
5. Durchführung einer Testphase
6. Paketierung der Applikation (Schreiben des Specfiles, Patch)
7. Deployment auf Produktivmaschine (Konfigurieren des Apache-Webservers, vhost, SSL)

3.2 Implementierung des Backends

Die Entwicklung der Applikation begann mit der initialen Erstellung eines RoR-Projektes. Nach Installation der zugehörigen Entwicklungsumgebung geschieht dies einfach mit Ausführung des Befehls `rails new` in dem gewünschten Verzeichnis. Mit Hilfe dieses Befehls wird eine standardisierte Projektmappe erzeugt.

Hier sei anzumerken dass RoR viele eingebaute Befehle mitbringt, welche die Ausführung von Entwicklungsschritten, wie bspw. das Anlegen von bestimmten Klassen, automatisiert. Ein manuelles Anlegen von relevanten Dateien im korrekten Verzeichnis entfällt dadurch und der Entwicklungsprozess kann massiv beschleunigt werden.

Das *Gemfile* bildet die zentrale Auflistung der von der Applikation genutzten Gems. Da die Entwicklung der Applikation im Sinne des TDD (Vgl. 2.3) erfolgt, wurde hier zunächst die Testsuite *rspec* hinzugefügt und die ersten Tests für die Models (Vgl. 2.4) geschrieben.

3.2.1 Erstellen der Models

Erst nach Schreiben der Tests erfolgte die Erstellung der eigentlichen Modeldateien. samt jeweils zugehörigen Attributen. Dies geschieht mit Hilfe sogenannter *Migrationen*, Datenbankoperationen geschrieben in Ruby. Zum Hinzufügen der so definierten Tabellen und Attribute zur Datenbank wird der Befehl `rake db:migrate` ausgeführt, welcher die Ruby-Syntax in *SQL* übersetzt.

Um die Integrität der Datenbanktabellen zu erhalten und ggfs. Falscheingaben des Anwenders abzufangen, werden in den Models Validierungen implementiert. Diese werden bei jeder Datenbankoperation ausgeführt und verhindern bei Eingabe fehlerhafter Daten ein Abspeichern in der Datenbank.

TODO: insert code example of user model validation/tool model validation

Im Anschluss dessen wurde eine weitere *Migration* ausgeführt, welche ein Typ-Attribut zu beiden Models hinzufügt. Dadurch wurde die in der Projektplanung niedergelegte Vererbungsstrategie der STI realisiert (Vgl. 2.7).

3.2.2 Erstellen der Controller

Wie auch bei den Models erfolgt vor Erstellung der Controller und während der Implementierung das Schreiben von Tests, in denen die grundlegend erwarteten Verhaltensweisen der Controller getestet werden.

Den Einstiegspunkt der Applikation bildet der *Application Controller*. In diesem wurden verschiedene Strategien implementiert, welche den Betrieb der gesamten Applikation betreffen, bspw. das initiale

Aufrufen der Nutzerauthentifizierung. Alle anderen Controller erben von diesem Application Controller und verfügen damit bei Aufruf über alle dort niedergelegten Methoden.

Per Konvention wird für jedes Model mindestens ein Controller erstellt, jedoch benötigt nicht jeder Controller ein Model. In jedem Controller befinden sich sogenannte *Actions*. Ruft der Nutzer bspw. die Webseite zum Hinzufügen eines neuen Alias für einen externen Service auf, wird seine Anfrage an die `new`-Action des jeweiligen Controllers weitergeleitet (Vgl. 2.4).

Zunächst wird der *ExternaltoolsController* erstellt. Alle relevanten Operationen wie Anzeige, Hinzufügen und Löschen von Daten des Tool-Models werden von diesem Controller verarbeitet. Dazu fragt der Controller Informationen zu dem eingeloggten Nutzer aus der User-Tabelle ab, welches über das Objekt `current user` realisiert wird. Dieses Objekt stammt aus dem eingesetzten Authentifizierungs-Gem *Devise* (Vgl. 3.4).

TODO: code example of externaltools controller

Anschließend werden in einem Unterverzeichnis die Controller für den Administratorbereich der Applikation angelegt. Um diese vor einem unberechtigten Aufruf zu schützen, wird ein *AdminController* erstellt, dessen einzige Methode prüft, ob ein Administrator eingeloggt ist. Ist dies nicht der Fall, erfolgt eine Weiterleitung zur Index-Seite.

Da nur die Administratoren die Möglichkeit besitzen sollen, Nutzer aus der Datenbank zu entfernen, befindet sich der *UsersController* in diesem Unterverzeichnis. Als Kernstück dieses Controllers wird die `list user`-Action implementiert, welche die gesammelten Informationen zu allen Mitarbeitern aus der Datenbank abfragt und die API bereitstellt.

TODO: code example of users controller

In ähnlicher Weise wird der *UnderlingsController* implementiert, welcher aber nur Informationen zu Teammitgliedern abfragt.

3.3 Implementierung der Benutzeroberfläche

Nach der Fertigstellung des grundlegenden Backends erfolgt die Programmierung der Benutzeroberfläche. Auch für diese werden Tests geschrieben, sogenannte *Feature Tests*.

TODO: code example of feature test

Wie in der Planung (Vgl. 2.5) niedergelegt, wird für die Implementierung das Framework Twitter Bootstrap verwendet. Die Einbindung in das Projekt erfolgt als *Gem*.

Die gewünschte Darstellung als *Dashboard* mit Paneel-Elementen lässt sich zeitsparend mit den in Bootstrap vorhandenen Vorlagen erstellen. Die Farbgebung der Elemente wird gemäß der Firmenrichtlinien gestaltet.

TODO: Screenshot basic externaltools view

Als spezielle Strategie zur Verbesserung der Performanz der Applikation werden sogenannte *Partials* eingesetzt. Bei dieser Vorgehensweise werden bei Webseiten bei Bedarf nur die sich verändernden Elemente neu geladen, statt die komplette Seite neu abzurufen. Dies verringert die benötigte Antwortzeit des Webservers maßgeblich.

TODO: Screenshot externaltools new action view

Für Zugang zu den Administratoransichten wird die reguläre Mitarbeiteransicht um weitere Tab-Elemente erweitert. Diese sind nur bei Login als Administrator sichtbar.

TODO: Screenshot admin view

3.4 Anbindung externer Schnittstellen

Wie bei jedem Entwicklungsschritt beginnt die Anbindung von externen Schnittstellen mit dem Schreiben von Tests.

Als Authentifizierungsmethode wird eine Authentifizierung über *LDAP* gewählt. Darüber kann das firmeneigene eDirectory zum Abgleich genutzt und gleichzeitig Informationen über den Mitarbeiter abgerufen werden.

Das Gem *Devise* bietet eine umfassende Sammlung an Konfigurationsmöglichkeiten und wird daher als Lösung eingesetzt. Beim erstmaligen Einloggen eines Mitarbeiters werden gleichzeitig dessen Name, Mitarbeiterkürzel und Informationen zu seinem Standort sowie der internen e-Mail-Adresse in der Datenbank abgespeichert. Informationen zu seinem Vorgesetzten werden einzeln dynamisch bei Bedarf abgefragt und angezeigt, da sich dieses Attribut öfter ändern kann.

Ebenso soll der Mitarbeiterstatus möglichst aktuell als Information verfügbar sein. Da diese Information allein in der Administratoransicht bzw. bei Ansprechen der Schnittstelle abgefragt wird, würde eine gesammelte Abfrage des Status für mehrere hundert Mitarbeiter gleichzeitig den Webserver überlasten. Aus diesem Grund wird ein Hintergrundprozess implementiert. RoR bietet dafür den sogenannten *Active Job*-Mechanismus an. Dort wird ein Prozess als Klasse implementiert, welcher einmal pro Woche für alle eingetragenen Mitarbeiter einzeln den Mitarbeiterstatus anfragt und als *boolean flag* in der Datenbank aktualisiert.

TODO: code example employee status job

Als Kernstück wird die Klasse `LDAPSearch` implementiert, welche für alle zusätzlichen LDAP-Abfragen außerhalb des User-Models verwendet wird.

TODO: code example ldap search.rb

Gewünscht war ebenfalls die Möglichkeit, dass Mitarbeiter ggfs. in Eigeninitiative eine Beitrittsanfrage an die Administratoren der Organisationen stellen können. Zu diesem Zweck wird der Status der Organisationszugehörigkeit ebenfalls durch einen ActiveJob bei der API von Github angefragt, welcher beim Hinzufügen eines Github-Nutzernamens ausgelöst wird. Bei nicht vorhandener Zugehörigkeit wird ein Link bereitgestellt, welcher eine fertige e-Mail-Vorlage an die Administratoren generiert.

Aus Zeitmangel konnte eine ähnliche Strategie für den externen Service Trello während des Projektverlaufs nicht mehr implementiert werden. Da dies aber nicht als kritisch bewertet wurde, behinderte dies nicht den Abschluss des Projektes.

3.5 Interne Schnittstelle

Die Progammierung der Schnittstelle begann zunächst mit der Analyse des Kommandozeilenwerkzeugs *etsync*, für welches die Schnittstelle entwickelt werden soll. Anhand des Codes in etsync wird die Ausgabe der Schnittstelle modelliert. Realisiert wird die Ausgabe in *JSON*, was ein gebräuchliches Datenformat zum Austausch von Informationen zwischen Anwendungen ist.

Um die gewünschten Informationen korrekt und einfach zu modellieren, wird das Gem *rabl* eingesetzt. Mit diesem Gem kann die JSON-Notation einfach als Active Record-Anfragen in Ruby geschrieben werden und vermeidet so Fehler bei der sonst manuellen Erstellung einer JSON-Vorlage, welche viele Klammern und Satzzeichen verwendet. Auch können spätere Anpassungen so leichter durchgeführt werden und eine genaue Kenntnis der JSON-Syntax ist überflüssig.

Die zugehörige Controller-Action befindet sich im Administratorverzeichnis, weshalb sie nur von Administratoren authentifiziert angesprochen werden kann (Vgl. 3.2.2).

3.6 Testphase

Nach Implementierung des Projektes konnte eine finale Testphase zeitlich gering gehalten werden, da die Applikation während der Entwicklung bereits durch das Prinzip des TDD laufend getestet und Fehler behoben wurden. Es wurden neben einem abschließenden kompletten Durchlauf der Testsammlung *Usability-Tests* der Nutzeroberfläche mit Probanden und realen Mitarbeiterdaten durchgeführt.

3.7 Paketierung der Applikation

Wie bereits in 2.9 dargelegt erfolgt nach Entwicklung und finaler Testphase der Applikation die Paketierung als RPM. Die Kontrollstruktur eines RPM-Pakets ist das sogenannte *Specfile*. In diesem werden alle relevanten Informationen zu dem Softwarepaket, wie bspw. Versionsnummer oder Abhängigkeiten auf andere Pakete, niedergelegt.

TODO: excerpt of specfile

Um das Paket immer auf dem neuesten Stand zu halten, wird über ein Skript bei einer Codeaktualisierung des Projekts auf der firmeneigenen Versionsverwaltungsplattform automatisch die Ausführung der Testsammlung ausgelöst. Sind diese erfolgreich mit den neuesten Änderungen ausgeführt worden, wird die Paketierung automatisiert angestoßen. Dies verhindert zusätzlich, dass fehlerhafte oder ungetestete Funktionalität in das endgültige Paket gelangt.

3.8 Deployment

Die Bereitstellung der Applikation geschieht in Kollaboration mit dem Team, in welchem das Projekt durchgeführt wurde. Auf einer dedizierten Produktivmaschine, die bereits als Webserver für andere Applikationen fungiert, wird eine weitere Konfiguration angelegt und das Software-Paket der Applikation installiert. Nach Neustart der zugehörigen Services und Prozesse ist die Webapplikation intern über den Browser zu erreichen.

4 Projektabschluss

4.1 Soll-/Ist-Vergleich

Beinahe alle Projektphasen konnten im geplanten Zeitraum ausgeführt werden. Lediglich bei der Anbindung externer Schnittstellen wurde mehr Zeit benötigt, da das Testentwicklung der LDAP-Verbindung aufwändiger war als zuvor angenommen. Auch die Implementierung eines effizienten Algorithmus für die `LDAPSearch`-Klasse erwies sich als zeitintensiver als geplant. Die komplette Paketierung samt aller Abhängigkeiten nahm auch marginal mehr Zeit in Anspruch, da nicht alle Gems in der gewünschten Version paketiert vorlagen.

Diese Differenz konnte allerdings durch die stark verkürzte Testphase ausgeglichen werden.

4.2 Projektabnahme

Die Projektabnahme erfolgte mit einer kurzen Präsentation und Demonstration der Applikation für den Projektbetreuer sowie die in Frage kommenden Administratoren der Organisationen.

4.3 Firmenweite Einführung

Nach Abnahme des Projekts wurde zu einem festgelegten Stichtag eine Ankündigungsmail auf den internen Mailinglisten verschickt, welche kurz den Nutzen der Applikation erklärt und alle Mitarbeiter zur Eintragung ihrer Daten aufruft. Desweiteren wurde das Anlegen des Datensatzes für jeden neuen Mitarbeiter verpflichtend als Richtlinie eingeführt.

Phase	Soll	Ist	Differenz
Analysephase	**5 h**	**5 h**	
Analyse des Ist-Zustands	2 h	2 h	
Formulierung des Soll-Zustands	3 h	3 h	
Projektplanung	**10 h**	**10 h**	
Planung der Entwicklungs- und Datenmodelle	7 h	7 h	
Planung des Layouts der Benutzeroberfläche	3 h	3 h	
Planung der Paketierung	2 h	2 h	
Implementierungsphase	**42 h**	**42 h**	
Entwicklung des Backends	20 h	20 h	
Implementierung der Benutzeroberfläche	3 h	3 h	
Anbindung externer Schnittstellen	5 h	7 h	+2 h
Programmierung der internen Schnittstelle	2 h	2 h	
Testphase	5 h	2 h	–3 h
Paketierung	5 h	6 h	+1 h
Deployment	2 h	2 h	
Einführungsphase	**3 h**	**3 h**	
Projektabnahme	2 h	2 h	
Firmenweite Einführung	1 h	1 h	
Erstellen der Dokumentation	**10 h**	**10 h**	
Gesamt	**70 h**	**70 h**	

Tabelle 2: Soll-/Ist-Vergleich

5 Fazit

In der vorgegebenen Zeit konnten die definierten Projektziele umgesetzt werden. Bis auf Verzögerungen bei der Anbindung externer Schnittstellen sind alle relevanten Funktionen der Webapplikation implementiert worden. Nach der firmenweiten Einführung fanden bereits mehrere Überprüfungen der Mitgliederlisten in den externen Organisationen statt und der Prozess wurde maßgeblich beschleunigt. Statt der vorherigen zeitintensiven Informationsbeschaffung genügt nun die Ausführung eines Befehls auf der Kommandozeile, welcher die gesammelten Daten aus der Webapplikation abruft. Das Projekt kann somit also als erfolgreich gewertet werden.

5.1 Ausblick

Die Webapplikation wird nach Abschluss weiterhin von mir als Hauptentwickler gepflegt und laufend gewartet.

Folgende zusätzlichen Funktionen sind geplant oder vorgeschlagen worden:

- Einbindung in die Authentifizierungsinfrastruktur der Micro Focus PLC, ähnlich des Intranets
- Vergabe des Administratorstatus an Mitarbeiter innerhalb der Applikation
- Auslagerung aller administrativen Arbeiten an die Webapplikation (Abgleich der Mitarbeiterlisten, Hinzufügen und Entfernen von Mitarbeitern in den externen Organisationen)