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

Glossar

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 werden die Produkte der SUSE Linux Enterprise-Familie gepflegt. Die Softwareentwicklung 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 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. Dies hat zur Folge, dass eine genaue Zuordnung der Accounts sowie ein Überblick über die 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 IT-Sicherheit und den Datenschutzbeauftragten 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 genannten externen Services können Unternehmen zur Abgrenzung und Verwaltung ihrer firmeninternen Entwicklungsprojekte Gruppierungen als sogenannte Organisationen anlegen. Die externen Accounts von Mitarbeitern können in diese Organisationen aufgenommen werden, um ihnen damit erweiterte Zugriffsrechte einzuräumen. So haben beispielsweise nur interne Mitarbeiter Schreibrechte auf den Quellcodeverzeichnissen, während Mitglieder der Open-Source-Community sowie Partner nur Vorschläge, sogenannte *Pull Requests*, bei dem jeweiligen Projekt einreichen können.

Der Abgleich der internen Mitarbeiter- mit den Userlisten innerhalb der angelegten Organisationen 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.

Der Abgleich der internen Mitarbeiter- mit den Userlisten innerhalb der angelegten Organisationen erfolgt bisher manuell. Das bedeutet, neue Mitarbeiter müssen persönlich oder virtuell kontaktiert werden, damit persönliche User-Aliase zur jeweiligen Organisation hinzugefügt werden können. Hierduch wird der Zugriff entsprechend der festgelegten Berechtigung ermöglicht.

Ein Vorgang, der mit steigender Nutzer- und Tool-Zahl an Aufwand und Komplexität zunimmt. Zusätzlich ist durch die dezentrale Beschaffung der benötigten Informationen das bisherige Konzept sehr fehleranfällig und nur ineffizient durchzuführbar.

1.4 Soll-Konzept

Ziel des Projektes ist es, eine geeignete Webapplikation zu entwickeln, die den bisherigen, oben aufgezeigten Prozess zentralisiert und vereinfacht. Hierdurch wird ebenfalls die Entlastung des administrativen Personals angestrebt. Die Lösung dafür stellt eine personenbezogene Zuordnung der internen Mitarbeiterdaten zu den jeweiligen externen Nutzerkonten dar. Realisiert werden soll dies als *Self-Service*. Hierbei sind Änderungen sowie die Verwaltung der Accountinformationen durch den Mitarbeiter selbst durchzuführen. Ausnahme stellt das Hinzufügen zu und Entfernen aus den entsprechenden Organisationen dar.

Zur Abfrage der mitarbeiterspezifischen Informationen, wie Name, Standort oder Vorgesetzter, soll das firmeneigene *eDirectory* als Datenquelle genutzt werden.

Die Applikation soll zur Funktionsdarstellung die die externen Services “Github” und “Trello” einbinden.

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.

Da das Tool firmenweite eingesetzt werden soll, muss eine intuitive und lokalisierbare Bedienbarkeit gewährleistet sein. Die Realisierung als Webapplikation ist daher bevorzugt.

Die Entwicklung soll auf ein modulares Framework zurückgreifen. Gleichzeitig soll die Applikation anpassbar und erweiterbar sein. Dies stellt ebenso eine Anforderung an den Quellcode, so dass dieser in einer entsprechend nachvollziehbaren Form geschrieben sein muss.

Die Informationen die das Tool mittels seiner API bereitstellt, müssen für das Administrationstool ët-sync"konsumierbar und verarbeitbar sein. Dies ist bereits in der Organisation vorhanden und ist nicht Teil dieses Projektes.

1.5 Projektverantwortlicher

Durchgeführt wird das Projekt innerhalb der SUSE Linux GmbH im Team SUSE-IT, Engineering Services Department. Ansprechpartner für das Projekt ist Cornelius Schumacher.

1.6 Anmerkung zur Dokumentation

Einschlägige Fachbegriffe der IT werden ohne Ausschrift im Text verwendet. Zu fachspezifische Begriffe werden kursiv abgebildet und sind im Glossar erläutert. Eigen- und Modulnamen werden in „ “ geschrieben. Begriffe, in diesem Fall besonders aus dem Bereich der verwendeten Programmiersprache, werden in ihrer englischen Urform gelassen.

2 Projektplanung

2.1 Zeitplanung

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

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. Einen Sonderstatus nimmt dabei der Projektverantwortliche ein. Im Bezug auf Anforderungen nimmt er eine kundenähnliche Rolle ein, ist aber ansonsten nicht Teil der Entwicklung.

2.2.2 Kostenplanung

Um die Übersichtlichkeit zu erhöhen, werden die Kosten wie folgt gegliedert und erläutert:

- **Personalkosten**: Belaufen sich auf die Höhe der Ausbildungsvergütung für den Zeitraum des Projekts.
- **Entwicklungskosten**: Belaufen sich auf Summe der anfallenden Kosten für die Entwicklung der Applikation. Hierzu gehören notwendige IT, Internetanschluss, Büro usw. Im Zuge dieses Projektes lassen sich die Kosten nicht von den regulären Betriebskosten trennen. Sie werden daher der Einfachheit halber für das Projekt als gegeben angesehen.
- **Softwarelizenzen**: Durch die ausschließliche Nutzung einer Entwicklungsumgebung und Werkzeugen aus dem Open-Source-Bereich entstehen keine zusätzlichen Kosten für Nutzung von Software.

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 eigentlich 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 wurde das sogenannte *Model-View-Controller-Schema* (MVC) verwendet. 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.

Über diese Aufteilung ist eine strukturierte und übersichtliche Entwicklung möglich. Funktionslogik, Datenbankoperationen und Ausgabelogik werden klar getrennt und können jeweils namentlich zugehörig gekennzeichnet werden.

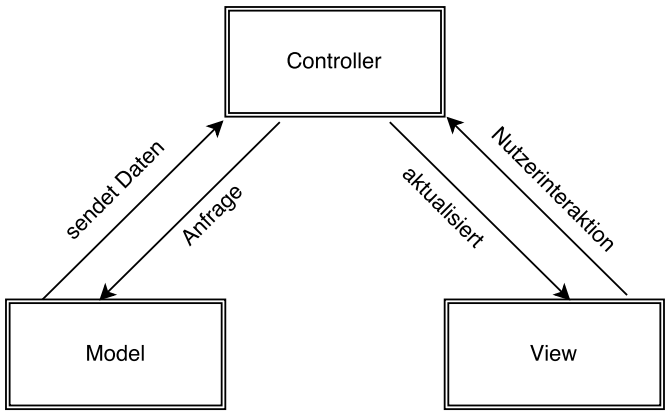

Abbildung 1: MVC-Schema

2.5 Wahl der Programmiersprache

Als Programmiersprache wurden *Ruby* und sein Web-Framework *Ruby on Rails* gewählt. Die entscheidenden Aspekte werden im folgenden Absatz erläutert.

Die objektorientierte Programmiersprache Ruby 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 leicht nachvollziehbar. Zusätzlich werden Ruby und RoR quelloffen verwaltet sowie entwickelt und sind gut dokumentiert.

Neben den vorhandenen Programmbibliotheken existieren zusätzlich von der Community bereitgestellte Module, sogenannte *Gems*. Die Einbindung dieser Gems in ein bestehendes Projekt vermeidet Duplikation und erspart viel Entwicklungszeit.

2.6 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.

2.7 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, welches die einfachste Methode ist. Aus sicherheitsrelevanter Sicht ist diese aber als nicht optimal zu bewerten, da sie sich als leicht manipulierbar erwiesen hat. Aus diesem Grund wurde entschieden, die Rechteverteilung auf Datenbankebene zu realisieren.

2.8 Datenmodell

Das Projekt verfügt über zwei Informationsträger, die in der Datenbank abgebildet werden sollen: den *User*-Objekten und den jeweils zugehörigen *Tool*-Objekten.

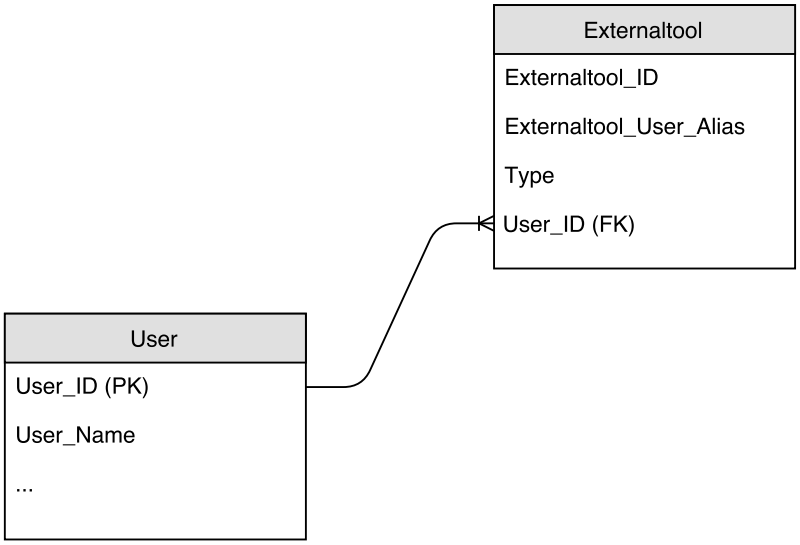

Abbildung 2: Datenmodell

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.

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.

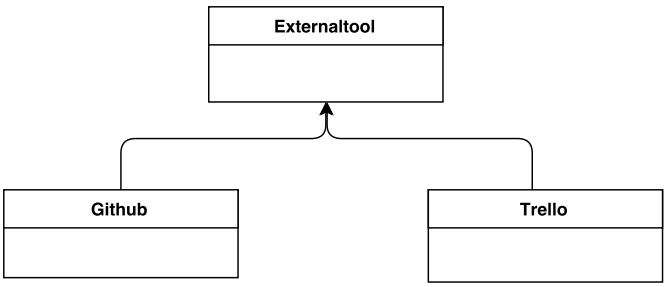

Abbildung 3: Beispielhafte Darstellung der STI

2.9 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 von einem Administrator angesprochen werden, der sich zuvor als solcher bei der Applikation identifiziert hat.

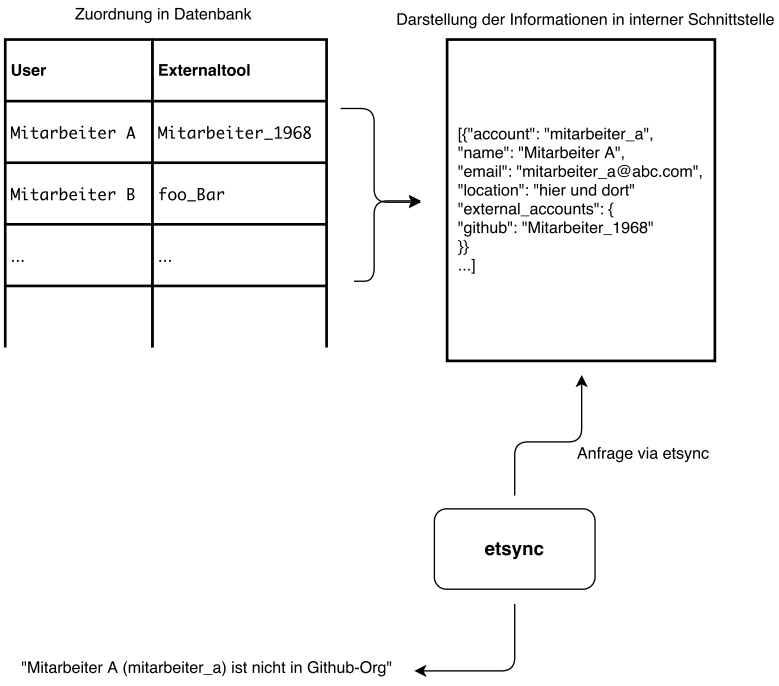

Abbildung 4: Visualisierung der internen Schnittstelle

2.10 Paketierung

Die Richtlinie des Teams SUSE-IT sieht die Paketierung einer entwickelten Applikation als *RPM* vor. 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 Neuinstallation auf einen anderen Server möglich.

3 Implementierungsphase

3.1 Einleitung

Nach der Planungsphase wurde im direkten Anschluss die Implementierungsphase eingeleitet. Diese umfasst die folgenden Schritte, welche in diesem Kapitel einzeln näher erläutert werden:

1. Implementierung des Backends
2. Implementierung der Benutzeroberfläche
3. Anbindung externer Schnittstellen
4. Implementierung der internen Schnittstelle
5. Durchführung einer Testphase
6. Paketierung der Applikation
7. Deployment auf Produktivmaschine

3.2 Implementierung des Backends

Als erster Schritt erfolgt die initiale Erstellung eines RoR-Projektes. Nach Installation der zugehörigen Entwicklungsumgebung geschieht dies einfach mit Ausführung des Befehls `rails new` im Arbeitsverzeichnis. Mit Hilfe dieses Befehls wird eine standardisierte Projektmappe erzeugt.

RoR bringt viele eingebaute Kommandozeilenbefehle mit, welche die Ausführung von Entwicklungsschritten automatisiert. Beispielsweise kann das Anlegen von relevanten Dateien im korrekten Verzeichnis direkt über einen solchen Befehl erledigt werden. Dadurch beschleunigt sich der Entwicklungsprozess massiv.

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.

```
1 class Externaltool < ActiveRecord::Base
2   belongs_to :user, inverse_of: :externaltools
3   TYPES = %w(Github Trello)
4   before_save :set_type
5   validates :type, presence: true, inclusion: { in: TYPES }
6   validates :alias, format: { with: /\A[^@\s]+\Z/ }
```

Abbildung 5: externaltool.rb

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. 2.8).

3.2.2 Erstellen der Controller

Per Konvention wird für jedes Model mindestens ein Controller erstellt, allerdings benötigt nicht jeder Controller ein Model. Jedoch befinden sich in jedem Controller eine oder mehr Methoden, sogenannte Actions. Ruft ein Nutzer beispielsweise 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).

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.

Zunächst wird der „Externaltools Controller“ 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. Dies wird über das Objekt `current user` realisiert. Das Objekt stammt aus dem eingesetzten Authentifizierungs-Gem „Devise“ (vgl. 3.4).

```
1  def new
2    @tool = current_user.externaltools.new
3    @current_manager = Manager.find_manager(current_user.username) unless Rails.env.test?
4    render :index
5  end
6
7  def create
8    @tool = current_user.externaltools.new(tool_params)
9    if @tool.save
10     GithubPublicAccessJob.perform_later(@tool.id) unless @tool.type == "Trello" || Rails.env.test?
11     flash [:notice] = "Success! Tool #{@tool.type} has been added. Checking access... #{undo_link}"
12   else
13     flash [:error] = "Something went wrong. Check if you entered a valid username (e.g. not the email address you log
           in with, an '@' in your alias or whitespace)."
14   end
15   redirect_to action: :index
16 end
```

Abbildung 6: externaltools.controller.rb

Zur Abgrenzung des Administratorbereichs der Applikation wird ein Administratorverzeichnis angelegt. Um die Controller der dort zu implementierenden Funktionen vor einem unberechtigten Zugriff zu schützen, wird ein „Admin Controller“ 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 „Users Controller“ 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.

```
1  def list_user
2    @users = User.all.order(:last_name)
3    render template: "admin/users/list_user.json.rabl"
4  end
```

Abbildung 7: users.controller.rb

In ähnlicher Weise wird der „Underlings Controller“ 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, die sogenannten *Feature Tests*.

```
1  scenario "User views homepage" do
2    visit root_path
3    expect(page).to have_content "Logged in as: schfueller"
4    expect(page).to have_content "Welcome"
5  end
```

Abbildung 8: employee.signin.spec.rb

Wie in der Projektplanung (vgl. 2.6) niedergelegt, wird für die Implementierung das Framework Twitter Bootstrap verwendet.

Die gewünschte Darstellung als *Dashboard* (vgl. 1.4) mit Panel-Elementen lässt sich zeitsparend mit den in Bootstrap vorhandenen Vorlagen erstellen. Die Farbgebung der Elemente wird gemäß der Firmenrichtlinien gestaltet.

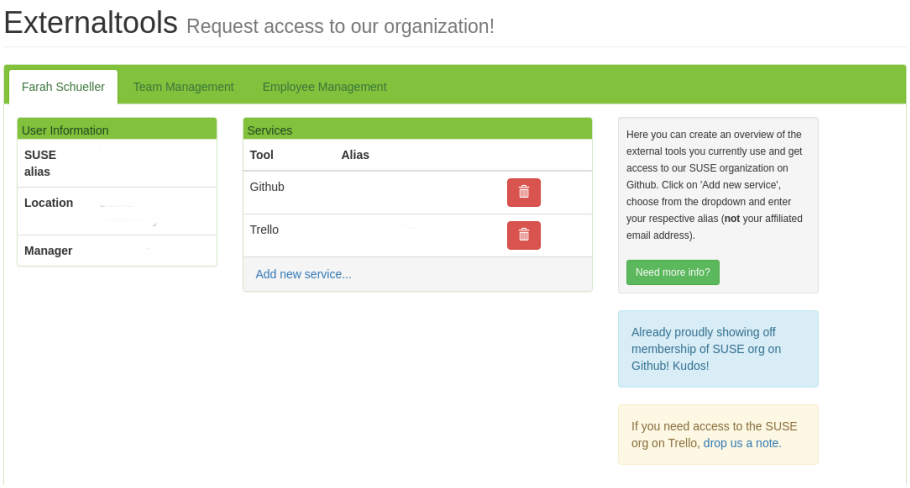

Abbildung 9: Nutzeroberfläche

Als Strategie zur Verbesserung der Performance der Applikation werden sogenannte *Partials* eingesetzt. Bei dieser Herangehensweise werden bei Bedarf nur die sich verändernden Elemente auf einer Webseite neu geladen, statt die komplette Seite neu abzurufen. Dies verringert die benötigte Antwortzeit des Webservers maßgeblich.

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

3.4 Anbindung externer Schnittstellen

Als Authentifizierungsmethode wird eine Authentifizierung über *LDAP* gewählt. Damit 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 die interne e-Mail-Adresse in der Datenbank abgespeichert. Informationen zum jeweiligen Vorgesetzten des Nutzers werden einzeln bei Bedarf abgefragt und angezeigt.

Der Beschäftigungsstatus eines Mitarbeiters soll möglichst aktuell als Information vorliegen. Da diese Information allein in der Administratoransicht bzw. über die Schnittstelle abgefragt wird, würde eine gesammelte Abfrage des Status für mehrere hundert Mitarbeiter gleichzeitig den LDAP-Server 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.

```
1  class EmployeeStatusJob
2    include Sidekiq::Worker
3    include Sidetiq::Schedulable
4
5    recurrence { weekly }
6
7    def perform
8      @users = User.all.order(:last_name)
9      @users.each do |user|
10       user.update_columns(employee_active: EmployeeStatus.check_employee_status(user.id) == "Active")
11     end
12   end
13 end
```

Abbildung 10: employee.status.job.rb

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

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.

```
1   def find_param(user_param, search_param, result_param)
2     result_attrs = [ search_param, result_param ]
3     search_filter = Net::LDAP::Filter.eq( search_param, user_param )
4     result = []
5     Net::LDAP.open(:host => "pan.suse.de", :port => 389, :encryption => :start_tls, :base => "o=Novell") do |ldap|
6       ldap.search(: filter => search_filter, : attributes => result_attrs ) do | entry |
7         result << entry.send(result_param).first
8       end
9     end
10    result
11  end
```

Abbildung 11: ldap.search.rb

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 Schnittstelle wird für die Verarbeitung durch das Kommandozeilenwerkzeug „etsync“ entwickelt. Anhand des bestehenden Codes in „etsync“ wird die Ausgabe der Schnittstelle modelliert. Realisiert wird die Ausgabe in *JSON*, welches 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 Datenbankabfragen 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 da eine genaue Kenntnis der JSON-Syntax überflüssig ist.

Die zugehörige Controller-Action befindet sich im Administratorverzeichnis, weshalb auf die Schnittstelle nur von Administratoren authentifiziert zugegriffen werden kann (vgl. 3.2.2).

3.6 Testphase

Nach Implementierung des Projektes konnte eine finale Testphase zeitlich gering gehalten werden. Durch den Einsatz von TDD wurde die Applikation während der Entwicklung bereits laufend getestet und eventuelle Fehler wurden behoben. Neben der Durchführung von *Usability-Tests* der Nutzeroberfläche mit Probanden und realen Mitarbeiterdaten fand abschließend ein kompletter Durchlauf der Testsammlung statt.

3.7 Paketierung der Applikation

Wie bereits in der Projektplanung 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.

Um das Paket immer auf dem neuesten Stand zu halten, wird über ein Skript bei einer Codeaktualisierung des Projekts 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 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 Vergleich der Zeitplanung

Beinahe alle Projektphasen konnten im geplanten Zeitraum ausgeführt werden. Lediglich bei der Anbindung externer Schnittstellen wurde mehr Zeit benötigt, da die 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.

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	1 h	1 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

4.2 Soll-/Ist-Vergleich

4.3 Projektabnahme

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

4.4 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.

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)