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

1 Einleitung 3
1.1 Projektumfeld . . . 3
1.2 Projektbeschreibung . . . 3
1.3 Ist-Analyse . . . 4
1.4 Soll-Konzept . . . 4
1.5 Projektverantwortlicher . . . 5
1.6 Anmerkung zur Dokumentation . . . 5

2 Projektplanung 6
2.1 Zeitplanung . . . 6
2.2 Ressourcenplanung . . . 6
2.2.1 Personalplanung . . . 6
2.2.2 Kostenplanung . . . 7
2.3 Entwicklungsprozess . . . 7
2.4 Architekturdesign . . . 7
2.5 Wahl der Programmiersprache . . . 8
2.6 Benutzeroberfläche . . . 8
2.7 Rechteverteilung . . . 9
2.8 Datenmodell . . . 9
2.9 Schnittstellen . . . 10
2.10 Paketierung . . . 11

3 Implementierungsphase 11
3.1 Einleitung . . . 11
3.2 Implementierung des Backends . . . 11
3.2.1 Erstellen der Models . . . 12
3.2.2 Erstellen der Controller . . . 12
3.3 Implementierung der Benutzeroberfläche . . . 13
3.4 Anbindung externer Schnittstellen . . . 14
3.5 Interne Schnittstelle . . . 16
3.6 Testphase . . . 16
3.7 Paketierung der Applikation . . . 16
3.8 Deployment . . . 17

4 Projektabschluss 18
4.1 Vergleich der Zeitplanung . . . 18
4.2 Soll-/Ist-Vergleich . . . 19
4.3 Projektabnahme . . . 19

4.4 Firmenweite Einführung 19

5 Fazit **20**

5.1 Ausblick 20

Abbildungsverzeichnis **21**

Tabellenverzeichnis **21**

Abkürzungsverzeichnis **21**

Glossar **22**

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.

Für dieses Projekt hervorzuheben ist das Department „Engineering Services“, welches sich besonders mit dem „SUSE-IT“ Team um die reibungslose Funktion der intern eingesetzten Applikationen kümmert.

1.2 Projektbeschreibung

Zur Entwicklung von 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 Mitarbeiteraccounts zu deren externen Accounts sowie ein Überblick über die Zugriffsberechtigungen nahezu unmöglich ist.

Zur Lösung dieser Problematik soll eine Applikation entwickelt werden, welche es ermöglicht, eine Zuordnung der firmeninternen Accounts zu möglichen Accounts externer Services durchzuführen. Das Ergebnis soll mittels einer Schnittstelle (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, IT-Sicherheit und den Datenschutzbeauftragten benötigen, beschränkt sich die erste Version dieser Applikation darauf, die Accountinformationen der Services 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, 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 Änderungsvorschläge, sogenannte „Pull Requests“, bei dem jeweiligen Projekt einreichen können.

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

Dieser Vorgang nimmt mit steigender Nutzer- und Tool-Zahl an Aufwand und Komplexität zu. Zusätzlich ist durch die dezentrale Beschaffung der benötigten Informationen das bisherige Konzept sehr fehleranfällig und nur ineffizient durchführbar.

1.4 Soll-Konzept

Ziel des Projektes ist es, eine geeignete Applikation 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, welche die zusätzlichen folgenden Anforderungen erfüllen soll:

- Die Realisierung erfolgt 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 externen Services „Github“ und „Trello“ einbinden.
- Um dem Mitarbeiter Möglichkeit zur Eigeninitiative einzuräumen, können die Schnittstellen der genannten externen Tools 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 die Applikation firmenweit eingesetzt werden soll, muss eine intuitive und lokalisierbare Bedienbarkeit gewährleistet sein. Die Realisierung als Webapplikation ist daher bevorzugt. Die eigentliche Lokalisierungsarbeit ist nicht Teil der Projektarbeit.
- 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, welche die Applikation mittels seiner API bereitstellt, müssen für das Administrationstool `etsync` konsumierbar und verarbeitbar sein. Dieses ist bereits im Unternehmen 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 in die Entwicklung involviert.

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 des Unternehmens 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.

2.3 Entwicklungsprozess

Als Entwicklungsmethode wird *Test Driven Development* (TDD) eingesetzt. Diese Methode sieht vor, dass vor dem Schreiben des eigentlichen Quellcodes zuerst der dazugehörige Test entwickelt wird. Dadurch wird ein abstrakter und zielorientierter Blick auf die eigentlichen Anforderungen an die Applikation ermöglicht. Hierbei wird als erstes eine Überlegung über das Ergebnis einer Funktion angestellt und darauf aufbauend die eigentliche Implementierung des jeweiligen Algorithmus gestartet, welcher das gewünschte Ergebnis erzielen soll.

Neben den Tests für den Quellcode werden ebenfalls Tests für die Benutzeroberfläche entwickelt.

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.

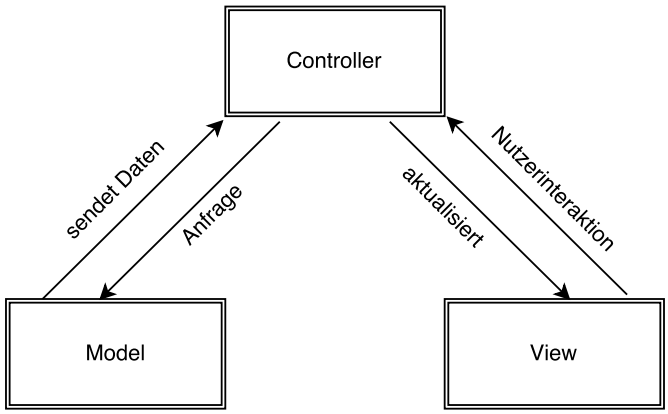

Abbildung 1: MVC-Schema

2.5 Wahl der Programmiersprache

Als Programmiersprache wurden *Ruby* und sein Web-Framework *Ruby on Rails* (RoR) 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 beispielsweise 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 es sich um kein teamübergreifendes Projekt handelt und möglichst zeiteffizient eine für Nutzer verständliche Oberfläche realisiert werden muss, 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.

2.7 Rechteverteilung

Die Applikation wird durch zwei verschiedenen Arten von Usern verwendet:

- Den regulären Mitarbeitern, die einzig Zugriff auf die eigenverwalteten Informationen über die jeweiligen externen Services haben
- Den Administratoren, die alle Datensätze der in der Datenbank vorhandenen Mitarbeiter einsehen, abfragen und löschen können.

Die Rechteverwaltung kann mittels des Einsatzes von *Flags* realisiert werden, welches auch die einfachste Methode darstellt. Aus Sichtweise der IT-Sicherheit ist diese jedoch nicht optimal, da eine Manipulation nicht auszuschließen ist. Daher 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.

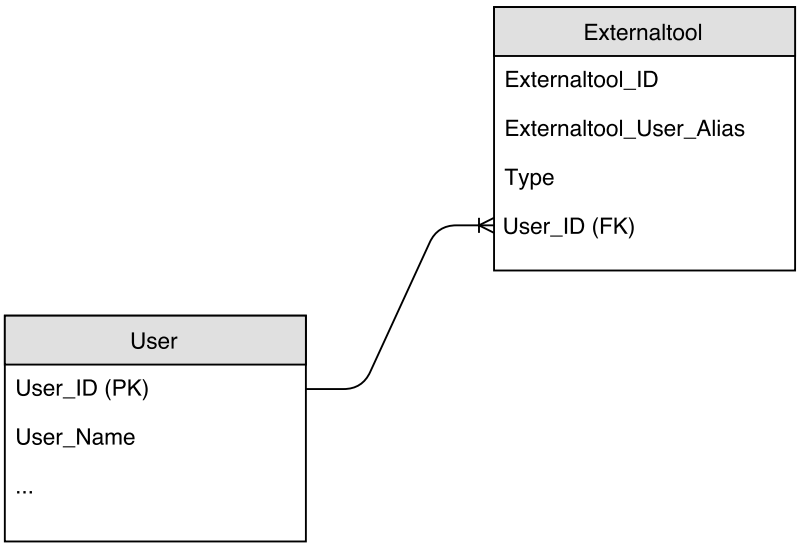

Abbildung 2: Datenmodell

Da für die Applikation zusätzliche Spezialisierungen der Datenbankobjekte notwendig sind, werden diese Beziehungen mit Hilfe der sogenannten *Single Table Inheritance* (STI) realisiert. Damit können beispielsweise unterschiedliche externe Tools in der Datenbank abgebildet werden.

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

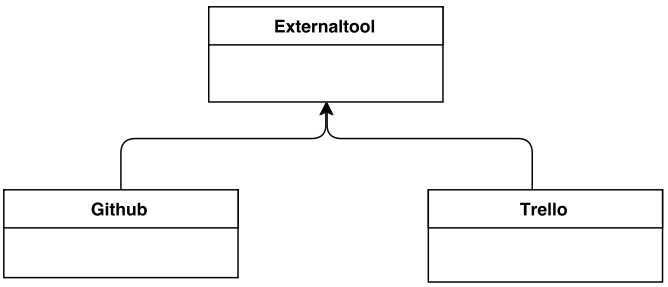

Abbildung 3: Beispielhafte Darstellung der STI

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 authentisiert hat.

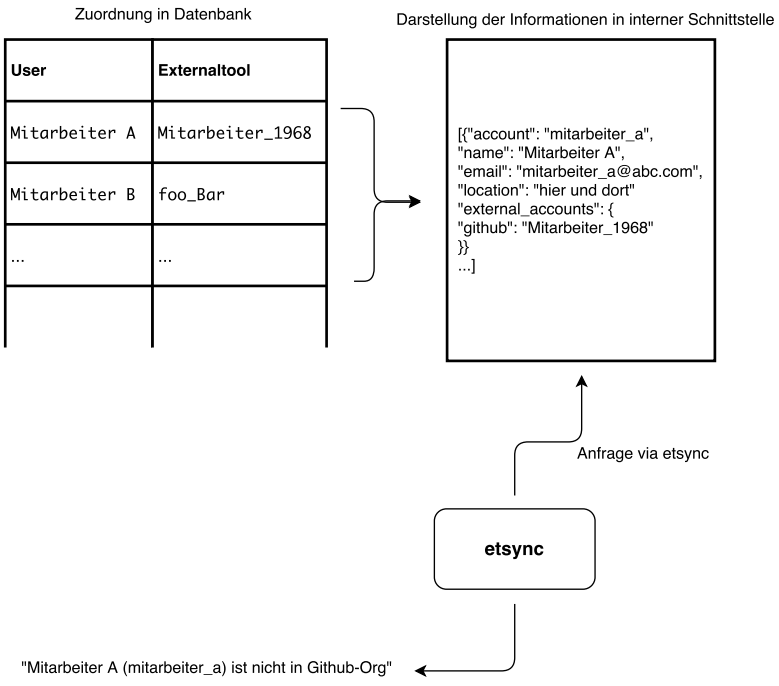

Abbildung 4: Visualisierung der internen Schnittstelle

2.10 Paketierung

Entsprechend der Richtlinien der SUSE-IT müssen entwickelte Applikationen als *RPM Package Manager*-Paket (RPM) gebaut und bereitgestellt werden. Dies ermöglicht eine einfache und automatisierbare Installation, da alle notwendigen Programmabhängigkeiten im Paket definiert werden und durch die eingesetzte Paketverwaltungs-Software automatisch aufgelöst und nachinstalliert werden können.

3 Implementierungsphase

3.1 Einleitung

Nach der Planungsphase wird 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 Ruby on Rails-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 besitzt viele integrierte Kommandozeilenbefehle, mit denen es möglich ist, die Ausführung von Entwicklungsschritten zu automatisieren. Beispielsweise kann das Anlegen von relevanten Dateien innerhalb der entsprechenden Verzeichnisstruktur direkt über diese Befehle erledigt werden, was zu einer Beschleunigung des Entwicklungsprozesses führt.

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

3.2.1 Erstellen der Models

Erst nach Schreiben der Tests erfolgt die Erstellung der eigentlichen Model-Dateien 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. Fehleingaben der Anwender abzufangen, werden in den Models Validierungen implementiert. Diese werden bei jeder Datenbankoperation ausgeführt und sollen verhindern, dass fehlerhafte Daten in der Datenbank abgespeichert werden.

```
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 wird eine weitere Migration ausgeführt, welche ein Typ-Attribut zu den beiden Models hinzufügt. Hiermit wird die in der Projektplanung erfasste Vererbungsstrategie STI realisiert (vgl. 2.8).

3.2.2 Erstellen der Controller

Entsprechend der Konvention wird für jedes Model mindestens ein Controller erstellt, jedoch benötigt nicht jeder Controller ein Model. Allerdings ist in jedem Controller mindestens eine Methode, die als „Action“ bezeichnet wird, zwingend erforderlich. Wird beispielsweise durch einen Benutzer die Webseite zum Hinzufügen eines neuen Alias aufgerufen, wird diese Anfrage an die `new`-Action des entsprechenden Controllers weitergeleitet. (vgl. Abschnitt 2.4)

Den Einstiegspunkt der Applikation bildet der „Application Controller“. In diesem werden verschiedene Strategien implementiert, welche den Betrieb der gesamten Applikation betreffen, beispielsweise das initiale Aufrufen der Nutzerauthentifizierung. Alle anderen Controller erben von diesem Application Controller und verfügen damit beim Aufruf über alle dort definierten 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. Abschnitt 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 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

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*.

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

```
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

Die Nutzeroberfläche lässt sich zeitsparend mit den in Bootstrap vorhandenen Vorlagen erstellen. Die Farbgebung der Elemente wird gemäß der Firmenrichtlinien gestaltet.

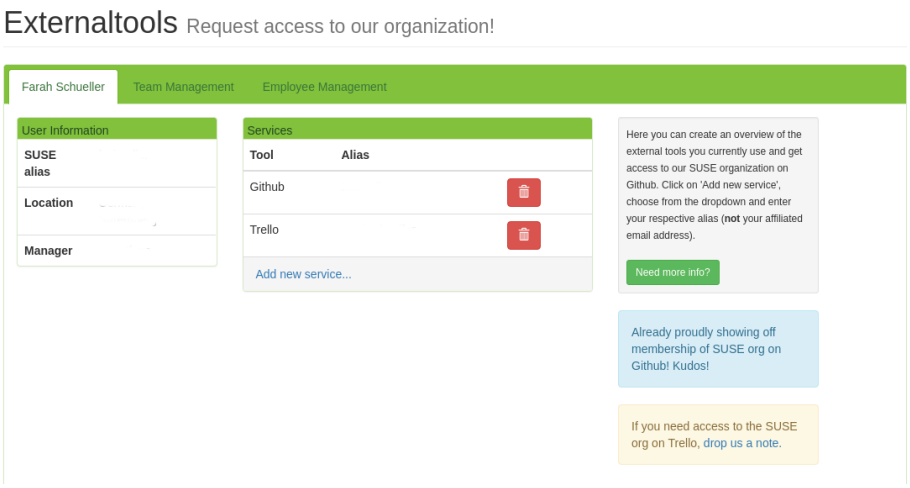

Abbildung 9: Nutzeroberfläche

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

Für Zugang zu den Administratoransichten wird die reguläre Mitarbeiteransicht um weitere Tab-Elemente erweitert. Diese sind nur beim 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 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.

```
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 => host_name, :port => port_number, :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

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 Schnittstelle wird für die Verarbeitung durch das Kommandozeilenwerkzeug `etsync` entwickelt. Anhand des bestehenden Codes in `etsync` wird die Ausgabe der Schnittstelle modelliert. Wie bereits in der Planungsphase erwähnt, erfolgt die Realisierung 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 Mitarbeitern und ihren Daten 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 dieser Datei werden alle relevanten Informationen zu dem Softwarepaket niedergelegt, wie beispielsweise Versionsnummer oder Abhängigkeiten auf andere Pakete.

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 Zusammenarbeit 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, welche im Abschnitt Testphase (vgl. 3.6) näher ausgeführt ist.

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

Zum Abschluss des Projektes erfolgt eine Kontrolle, dass alle im Soll-Konzept aufgeführten Anforderungen erfüllt werden:

- Zusammenfassend kann angeführt werden, dass eine Webapplikation im Stil eines Self-Service geschaffen wurde. Diese wurde mit Ruby sowie Ruby on Rails in einem modularen Aufbau erstellt.
- Als Datenquelle für die Mitarbeiterinformationen wird das interne eDirectory verwendet.
- Über die realisierten Benutzerrollen ist die Verwaltung der eigenen Informationen für jeden Benutzer möglich. Jedoch ist nur Administratoren zusätzlich Einsicht in die Datenbank sowie das Entfernen von Benutzern aus dem Tool gestattet.
- Die API ist nur als authentifizierter Administrator benutzbar.

4.3 Projektabnahme

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

Bereits während dieser Phase war eine deutliche Verbesserung gegenüber dem in der Ist-Analyse aufgezeigten Prozesses zu vermerken.

4.4 Firmenweite Einführung

Die firmenweiter Einführung des Tools erfolgte in mehreren Schritten:

1. Der operative Betrieb der Applikation sowie die Freigabe und Bekanntgabe wird durch SUSE-IT sichergestellt.
2. Zu einem festgelegten Termin ist eine E-Mail mit einer Anleitung für die Benutzer versendet worden.
3. Nach der Einführungsphase ist das Tool in die Firmenrichtlinie „Nutzung interner Entwicklungstools“ aufgenommen worden.

5 Fazit

In der vorgegebenen Zeit konnten alle 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 konnte, wie bereits dargestellt, erheblich beschleunigt werden. 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)
- Anbindung an die Trello API

Abbildungsverzeichnis

1 MVC-Schema . . . 8
2 Datenmodell . . . 9
3 Beispielhafte Darstellung der STI . . . 10
4 Visualisierung der internen Schnittstelle . . . 10
5 externaltool.rb . . . 12
6 externaltools.controller.rb . . . 13
7 users.controller.rb . . . 13
8 employee.signin.spec.rb . . . 14
9 Nutzeroberfläche . . . 14
10 employee.status.job.rb . . . 15
11 ldap.search.rb . . . 15

Tabellenverzeichnis

1 Zeitplanung . . . 6
2 Soll-/Ist-Vergleich . . . 18

Abkürzungsverzeichnis

API	Application Programming Interface
CSS	Cascading Style Sheets
HTML	Hypertext Markup Language
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

Self-Service-Technologies

Technische Kommunikationsmöglichkeiten, die Nutzer zur selbstständigen Inanspruchnahme einer Dienstleistung befähigen. Sie werden hauptsächlich zur Entlastung des Personals von Standardaufgaben eingesetzt.

eDirectory

Von Novell entwickelter Verzeichnisdienst, welcher ähnliche Funktionen wie das Active Directory aufweist.

Test Driven Development

Testgesteuerte Programmierung, ist eine Software-Entwicklungsprozess. Bei dieser Methode werden Software-Tests vor den zu testenden Komponenten entwickelt.

Ruby

Höhere Programmiersprache, welche Mitte der 1990er Jahre von Yukihiro Matsumoto entwickelt wurde. Ruby ist vollständig objektorientiert und wird zur Laufzeit interpretiert. Sie folgt dem sogenannten „Prinzip der geringsten Überraschung“, welches besagt, dass Programmierer eine Sprache möglichst intuitiv und ohne Verhaltensanomalien der Sprache nutzen können.

Ruby on Rails

Das Web Application Framework von Ruby. Es ist von den Prinzipien „don’t repeat yourself“ (DRY) und „convention before configuration“ geprägt, welche besagen, dass Code möglichst ohne Duplikation geschrieben und die Konventionen bei der Namensgebung von Objekten eingehalten werden sollen.

Funktionale Programmierung

Programmierparadigma, bei welchem Programme ausschließlich aus Funktionen bestehen. Beispiele für funktionale Programmiersprachen sind *Haskell* oder *Scala*.

Imperative Programmierung

Programmierparadigma, bei welchem Programme aus einer Folge von Anweisungen bestehen. Es ist das am längsten bekannte Programmierparadigma. Beispiele für imperative Programmiersprachen sind *C* oder *Ada*.

Objekt (objektorientierte Programmierung)

Stellt ein Exemplar eines Datentyps oder einer Klasse dar. Sie werden zur Laufzeit erzeugt und haben einen Zustand (Attribute), ein Verhalten (Methoden) und eine Identität (Einzigartigkeit des Objekts trotz Existenz gleichartiger Objekte)

Gem

Auch RubyGems genannt, stellt das Paketsystem der Programmiersprache Ruby dar. Mit der Nachinstallation von Gems kann Ruby um weitere Funktionen erweitert werden.

Flag

In der Informatik als Statusindikator eingesetzt. Häufig wird dafür eine boolesche Variable genutzt, welche die Zustände „wahr“ oder „falsch“ annehmen kann.

Twitter Bootstrap

Ein von Twitter entwickeltes CSS-Framework, welches Gestaltungsvorlagen für Typografie, Formulare, Buttons, Tabellen, Grid-Systeme, Navigations- und andere Oberflächengestaltungselemente umfasst und zusätzliche Erweiterungen für JavaScript enthält.

Single Table Inheritance

Tabelle pro Vererbungshierarchie, verwendet eine einzige Tabelle für jede Klasse, um einen Klassenbaum in einer Datenbank abzubilden.

Partial View

Dynamische Komponenten eines sonst statischen HTML/CSS-Views.

Active Record

Konzept für objektorientierte Software zur Speicherung von Daten in einer relationalen Datenbank.

Active Job

Klassentyp in Ruby on Rails, welcher zur Ausführung von Hintergrundprozessen, beispielsweise externe API-Anfragen, existiert.

Feature Test

Testmethode zum Testen einer kompletten Funktionalität (Feature) und dem Zusammenwirken der Softwarekomponenten im Hintergrund.

Usability Test

Testmethode zum Testen der Gebrauchstauglichkeit einer Software mit den potentiellen Benutzern.

RPM Package Manager

Freies Paketverwaltungssystem, ursprünglich entwickelt von Red Hat.