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

Ausbildungsbetrieb:
SUSE Linux GmbH
Maxfeldstraße 5
90409 Nürnberg

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
MVC	Model View Controller
SQL	Structured Query Language
STI	Single Table Inheritance
RoR	Ruby on Rails
TDD	Test Driven Development

1 Einleitung

1.1 Projektumfeld

Die SUSE Linux GmbH wurde 1992 gegründet und ist seit 2014 eine Tochtergesellschaft der Micro Focus AG. 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. Dies wird besonders im Bereich der Codeentwicklung und -pflege deutlich, da neben den firmeneigenen Entwicklern auch Partner sowie Open-Source-Community 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 darauf, die Accountinformationen der Applikationen Github (verzeichnisorganisierte Versionsverwaltung von Quellcode) und Trello (Tool zur Steuerung von agilen Software-Projekten) einzubinden.

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 Mitarbeiteraustritten, 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, welcher bisher von einigen wenigen Administratoren gestemmt wird. Durch die ungenaue Beschaffung der benötigten Informationen ist das bisherige Konzept sehr fehleranfällig und ineffizient durchzuführen.

1.4 Soll-Konzept

Um Personal zu entlasten und den Prozess wesentlich zu vereinfachen, soll eine zentrale Plattform geschaffen werden, die eine gesammelte Zuordnung der Mitarbeiter zu ihren externen Nutzerkonten zum Abgleich bereitstellt. Die Datensätze sollen von den jeweiligen Mitarbeitern selbst verwaltet und gepflegt werden. 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.

Die Plattform soll als Webapplikation gestaltet werden, was eine plattformunabhängige Nutzung ermöglicht. Zusätzlich soll die Implementierung modular und leicht erweiterbar sein, um zukünftig geplante Funktionen einfach hinzufügen und warten zu können. Ebenso soll die Applikation über Tests verfügen, welche jene für andere Entwickler in Zukunft nachvollziehbar macht.

1.5 Projektschnittstellen

Die Applikation nutzt mehrere APIs um z.B. die Betriebszugehörigkeit oder verschiedene Informationen aus den Datenbanken der externen Services abzufragen.

Mitarbeiterspezifische Informationen, wie Name, Standort oder Vorgesetzter, werden aus dem firmeneigenen eDirectory abgefragt. Zur Abfrage der Organisationszugehörigkeit werden die externen Schnittstellen der genannten Tools Github und Trello verwendet.

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

TODO: enter planning table here

2.2 Ressourcenplanung

2.2.1 Personalplanung

Das für die Entwicklung benötigte Personal besteht aus einem Auszubildenden, der dafür von seiner regulären Mitarbeit im derzeitigen Team freigestellt wird.

2.2.2 Kostenplanung

Die Personalkosten belaufen sich auf ein Auszubildendengehalt, wodurch lediglich Stromkosten für die zur Projekterstellung eingesetzte Hardware anfallen. Durch die ausschließliche Nutzung einer Entwicklungsumgebung und Werkzeugen aus dem Open-Source-Bereich entstehen keine zusätzlichen Kosten wie Software-Lizenzen.

2.3 Entwicklungsprozess

Der Entwicklungsprozess erfolgt *test-driven*, was bedeutet dass vor dem Schreiben des eigentichen Quellcodes einer Funktionalität zunächst der dazugehörige Test entwickelt wird. Diese Vorgehensweise ermöglicht einen abstrakteren und zielorientierteren Blick auf die eigentlichen geforderten Funktionen der Applikation, 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.

TODO: enter figure of MVC schematics

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 ohne die Mitwirkung eines darauf spezialisierten *UI/UX*-Designers 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*.

TODO: insert ER-model

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.

TODO: insert UML-Class-Model feat. 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. Zum Hinzufügen 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 das Schreiben von Tests, in denen die grundlegend erwarteten Verhaltensweisen der Controller getestet werden.

Das Kernstück 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 Datenbankoperationen 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. Das Kernstück dieses Controllers ist die `list user`-Action, 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

- Schreiben von Feature-Tests (capybara) (code example)
- Einbindung von Bootstrap
- Gestaltung des Layouts als Dashboard, Panel und Tabelemente (LH?) (screenshot)
- Nutzung von Partials zur Verringerung der Response-Time
- Admin-Tabs nur sichtbar und anzeigbar wenn als Admin authentifiziert (controller-restrained)

3.4 Anbindung externer Schnittstellen

- Everybody loves tests
- LDAP-Konfiguration für Devise zur Mitarbeiterauthentifizierung
- first login: Abfrage statischer Informationen wie Name, Firmenemail, Standort; Speichern in DB
- bei jedem weiteren Login: dynamische Abfrage des Managers (auch bei Detailansicht in Admin-View)
- dynamische Abfrage des Mitarbeiterstatus einmal wöchentlich
- Team-view
- vgl. entwickelte LDAP-Methode (code example)
- Github/Octokit: Abfrage der Organisationszugehörigkeit und Anzeige von mailto an admins
- Gleiche Methode für Trello auf Grund von Zeitmangel nicht mehr implementiert

- Sidekiq/Sidetiq Jobs (code example)

3.5 Interne Schnittstelle

- Test it!
- Formatierung des JSON-Outputs für etsync
- Controller response to JSON
- rabl-gem zur einfachen Formatierung von JSON mit einfachen Active Record-Queries
- Da in Admin-Namespace nur ansprechbar bei Authentifizierung als Administrator

3.6 Testphase

- Gering gehalten, da TDD inkl. Feature-Tests
- Rerun der Testsuite
- Manueller Testrun mit Usern zum Test der GUI

3.7 Paketierung

- rpm
- specfile (excerpt of specfile)
- Automatisierung der Paketierung bei neuem Commit in Repository, nach Run der Tests
- Hält Paket auf neuestem Stand

3.8 Deployment

- Deployment in Kollaboration mit SUSE IT-Team, Bereitstellung der Produktivmaschine und Produktivdatenbank
- Konfigurieren eines Apache-Servers für vhosts
- Manuelle Installation der Applikation auf Produktivmaschine