ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Datenbanken und Informationssysteme

Konzeption eines web-basierten Repositories zur Verwaltung von Prozessen und Prozesskollektionen

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Albert Bub
albert.bub@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Andreas Lanz

2014

Fassung 11. Oktober 2014

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

Im Rahmen dieser Bachelorarbeit wird ein web-basiertes Repository zur Verwaltung von Prozessdokumentationen und Prozesskollektionen konzipiert und entwickelt. Hierfür wird zunächst anhand entsprechender Anwendungsfälle eine grundlegende Anforderungsanalyse durchgeführt. Anschließend wird ein passendes erweiterbares Datenmodell konzipiert, welches es erlaubt die einzelnen Beschreibungen der Prozesse und ihrer Prozesskollektionen mit möglichen Zusatzinformationen zu speichern, sowie die dazugehörigen Dateien zu versionieren. Um die Prozesse den Benutzern des Repositories zur Verfügung zu stellen, wird eine web-basierte Benutzerschnittstelle entworfen. Diese soll die im Datenmodell gehaltenen Daten auf eine geeignete Art und Weiße zur Verfügung stellen sowie deren Erstellung und Verwaltung ermöglichen. Für die Benutzerschnittstelle werden zunächst verschiedene Mockups erstellt, die bei der Implementierung als Grundlage dienen. Nach der Konzeption des Systems werden geeignete Technologien ausgewählt, mit deren Hilfe das entwickelte Repository schlussendlich implementiert wird.

Inhaltsverzeichnis

1

Einleitung

In jedem Unternehmen existieren wiederkehrende Arbeitsvorgänge. Solche wiederkehrende Arbeitsvorgänge bestehen oft aus einem einzigen Arbeitsschritt, können aber auch aus mehreren Arbeitsschritten zusammengesetzt sein. Eine Online-Bestellung stellt zum Beispiel einen solchen Arbeitsvorgang dar, der aus mehreren Arbeitsschritten besteht. Der Ablauf der Arbeitsschritte eines Arbeitsvorgangs in einem Unternehmen wird als ein *Geschäftsprozess* oder kurz als *Prozess* bezeichnet.

Während in kleineren Unternehmen die Prozesse möglicherweise noch von einzelnen Mitarbeiter nachvollziehbar sind, trifft dies bei größeren Unternehmen nicht mehr zu. Hierdurch wird oft der Blick auf den Gesamtprozess vernachlässigt, da bestimmte Arbeitsschritte eines Prozesses von der zugehörigen Abteilung übernommen werden aber keine Gesamtübersicht existiert. Dieser Umstand macht eine gute Beschreibung der Prozesse notwendig. Eine solche Beschreibung wird als *Prozessdokumentation*

bezeichnet und besteht in den meisten Fällen aus mehreren Dateien die den jeweiligen Prozess textuell, graphisch oder sogar in Form von sogenannten *Prozessmodellen* beschreiben. Hierbei stellt ein Prozessmodell eine Modellierung eines Prozesses in einer bestimmten Notation dar.

Fasst man mehrere zusammengehörige Prozesse zusammen so erhält man eine “Prozesskollektion“. Diese kann die Prozesse beispielsweise nach verschiedenen Domänen, Unternehmensbereichen oder Wertschöpfungsketten zusammenfassen. Größere Prozesskollektionen können mehrere kleinere beinhalten, die ebenfalls wieder Prozesskollektionen enthalten dürfen. Die komplexe Struktur die Prozesskollektionen annehmen können ist ein weiterer Grund für die Notwendigkeit einer guten Beschreibung und Dokumentation der Prozesse.

1.1 Motivation

Die Anzahl an existierenden Prozessen ist sehr groß. Durch die heutige prozessorientierte Ausrichtung von Industrie und Forschung steigt die Anzahl der Prozesse weiterhin unaufhörlich. Ältere Prozesse werden durch effizientere ersetzt oder neue konzipiert um bestimmte Aufgaben zu erleichtern. Jeder Prozess wird dabei in einer Dokumentation beschrieben und diese entstandene Prozessdokumentation daraufhin archiviert. Letzteres geschieht bisher jedoch dezentral, da ein zentrales Verwaltungssystem für solche Prozessdokumentationen nicht vorhanden ist. Das hat zur Folge das die Archivierung und Wiederauffindung der Prozessdokumentationen sich immer schwieriger gestaltet. Viele Prozessdokumentationen bestehen dabei aus mehreren Dateien, deren Dateiformate enorm variieren können. Zusätzlich können diese Dateien in mehreren Versionen vorhanden sein, was den Verwaltungsaufwand weiter steigert und die Übersicht über die vorhandenen Dateien mindert.

Um den Verwaltungsaufwand für Prozessdokumentationen zu minimieren soll daher im Rahmen dieser Bachelorarbeit ein web-basiertes Prozess Repository entworfen und implementiert werden. Das Repository wird dabei als ein zentrales Verwaltungssys-

tem für Prozesskollektionen, Prozessdokumentationen und die dazugehörigen Dateien konzipiert und entwickelt.

1.2 Zielsetzung

Ziel des zu entwickelnden Repositories ist es Prozessdokumentationen und Prozesskollektionen, samt all ihrer Beschreibungsinformationen, dauerhaft zu archivieren und sie für die Benutzer übersichtlich darzustellen. Die zugehörigen Dateien einer Prozessdokumentation sollen bei Bedarf versionierbar und die Versionierung nachvollziehbar sein. Der Verwaltungs- und Suchaufwand für die Prozessdokumentationen soll dabei auf ein Minimum reduziert werden, was wiederum durch eine effiziente Suche und Speicherung der nötigen Daten realisiert werden soll. Zudem soll das Repository so simpel wie möglich gehalten werden, um seinen Benutzern einen schnellen Einstieg in das System zu ermöglichen.

1.3 Aufbau dieser Arbeit

Dieses Kapitel beschreibt die Problemstellung sowie die Motivation und Zielsetzung dieser Bachelorarbeit. Zusätzlich werden wichtige Begriffe wie “Prozessdokumentation“ und “Prozesskollektion“ eingeführt und erläutert.

Das Kapitel 2 stellt drei weitere, bereits existierende Ansätze von Prozess Repositories vor und grenzt das zu entwickelnde von ihnen ab.

In Kapitel 3 wird daraufhin eine komplette Anforderungsanalyse des Prozess Reopsitory erstellt, sowie die funktionalen- und nicht-funktionalen Anforderungen erhoben.

Danach werden in Kapitel 4 die Technologien vorgestellt, die ausgewählt werden um das Repository zu implementieren. Auch das verwendete Framework “Ruby on Rails“ wird vorgestellt.

In Kapitel 5 wird die Implementierung des Prozess Repository anhand des eingesetzten Model- View- Controller Entwurfsmusters beschrieben und mit Beispielen veranschau-

licht. Es wird auf die zugrunde liegende Systemarchitektur, das konzipierte Datenmodell, das Design der Benutzerschnittstellen und auf einige Controller Klassen eingegangen.

Das Kapitel 6 enthält zum Schluss eine Zusammenfassung über die Themen dieser Bachelorarbeit.

2

Abgrenzung zu existierenden Ansätzen

In diesem Kapitel wird das zu entwickelnde Repository mit bereits existierenden Ansätzen verglichen und von ihnen abgegrenzt. Es wird dabei auf die Ansätze APROMORE [LRRVDA+11], *Business Process Model Repository* [YDG12] und *Semantic Business Process Repository* [MWA+07] eingegangen.

2.1 APROMORE

Der Name APROMORE steht für “Advanced Process Model Repository“ [LRRVDA+11]. Hierbei handelt es sich um ein web-basiertes Repository welches konzipiert wurde um eine breite Anzahl an Funktionen für die Analyse, die Verwaltung und die Nutzung von Prozessmodellen anzubieten [LRRVDA+11]. Der Schwerpunkt von APROMORE liegt dabei in anspruchsvollen state-of-the-art Funktionen für die Verwaltung und Analyse von

Prozessmodellen wie zum Beispiel der erweiterten modellbasierten Analyse oder der Filterung von Prozessmodellen. Diese Funktionen können auf einzelne Prozessmodelle oder auf eine Gruppe ähnlicher Prozessmodelle angewendet werden.

Der erste Funktionsbereich von APROMORE ist die *Evaluation* von Prozessmodellen. Sie bietet Funktionen an für die Korrektheits-, Performance- und Nutzbarkeitsanalyse von Prozessmodellen. Der zweite Funktionsbereich ist die *Filterung* von Prozessmodellen. Sie enthält Funktionen um den Prozessmodellen einen Rang auf Basis ihrer Gleichheit oder dem Grad ihrer Ähnlichkeit zuzuordnen. Dieser Rang kann zur Abfrage verwendet oder zur Identifikation bestimmter Muster verwendet werden. Ein weiterer Funktionsbereich im APROMORE ist das *Design*. Hier werden Funktionen zum erstellen und bearbeiten von Prozessmodellen angeboten, wobei neue Prozessmodelle aus mehreren ähnlichen Prozessmodellen erstellt werden können. Der letzte Funktionsbereich ist die *Präsentation*, in welchem Funktionen angeboten werden die das Verständnis für Prozessmodelle und Prozesskollektionen verbessern sollen. Die Algorithmen des APROMORE können auf Prozessmodellen operieren die in diversen Notationen vorliegen. Dies geschieht auf der Grundlage eines kanonisches Formats, welches jedes Prozessmodell in XML darstellt und die Besonderheiten der verschiedenen Notationen abbilden kann [LRRVDA+11].

2.2 Business Process Model Repository - Framework

Beim *Business Process Model Repository* [YDG12] handelt es sich um ein Framework für die Verwaltung von Prozesskollektionen bestehend aus Prozessmodellen. Das Framework wurde konzipiert um die Verwaltung großer Prozesskollektionen zu unterstützen und besteht aus einem Model Management und einer Referenzarchitektur.

Das Model Management besteht aus drei Teilen, dem *Prozess-Daten-Modell*, dem *Prozess-Funktionsmodell* und dem *Prozess-Management-Modell* [YDG12]. Das *Prozess-Daten-Modell* besteht wiederum aus drei Teilen und ist für eine effiziente Speicherung der Prozessmodelle zuständig. Das *Prozess-Funktionsmodell* bietet die Grundfunktionen für die Manipulation von Prozessmodellen und Funktionen für die Navigation

beziehungsweise Suche nach Prozessmodellen. Das *Prozess-Management-Modell* bietet erweiterte Verwaltungsfunktionen für Prozessmodelle. Die Versions-, Konfigurations- und Lebenszyklus Verwaltung stellt hierbei nur ein Teil der angebotenen Verwaltungsfunktionen dar [YDG12].

In der Referenzarchitektur wird beschrieben wie die einzelnen Komponenten des *Business Process Model Repository* in Verbindung stehen. Sie ist in fünf Schichten unterteilt: die *Präsentationsschicht*, die *Prozess-Repository-Management* Schicht, die *Repository-Management* Schicht, die *Datenbank Management* Schicht und die *Speicherschicht* [YDG12]. Die *Präsentationsschicht* bietet eine Graphische Oberfläche für die Interaktion mit den Benutzern und dem Repository an. Die *Prozess-Repository-Management* Schicht und die *Repository-Management* Schicht bieten die Funktionen für die Verwaltung von Prozessmodellen an. Die *Datenbank Management* Schicht ist für die Änderungsoperationen und die *Speicherschicht* für die Archivierung zuständig [YDG12]. In Abbildung 2.1 werden die einzelnen Komponenten des *Business Process Model Repository*, ihre Verbindungen und ihre bereitgestellten Funktionen aufgezeigt.

2.3 Semantic Business Process Repository

Im *Semantic Business Process Management* (SBPM) basieren die Prozessmodelle auf *Prozess-Ontologien* die wiederum von anderen Ontologien wie zum Beispiel Organisations-Ontologie Gebrauch machen [MWA+07]. Das *Semantic Business Process Repository* (SBPR) ist ein Prozess Repository das in der Lage ist semantische Prozessmodelle, die mit solchen Ontologien beschrieben werden, zu speichern und wieder auszulesen. *Semantische Prozessmodelle* stellen hierbei Prozessmodelle dar, die mit semantischen Informationen annotiert werden.
SBPR ist integraler Bestandteil von SBPM und wird für die Speicherung und die Verwaltung von semantischen Prozessmodellen verwendet [MWA+07].

SBPR bietet seinen Nutzern die Möglichkeit neue Prozessmodelle dem Repository hinzuzufügen und bereits bestehende mit Hilfe von Änderungsoperationen zu manipu-

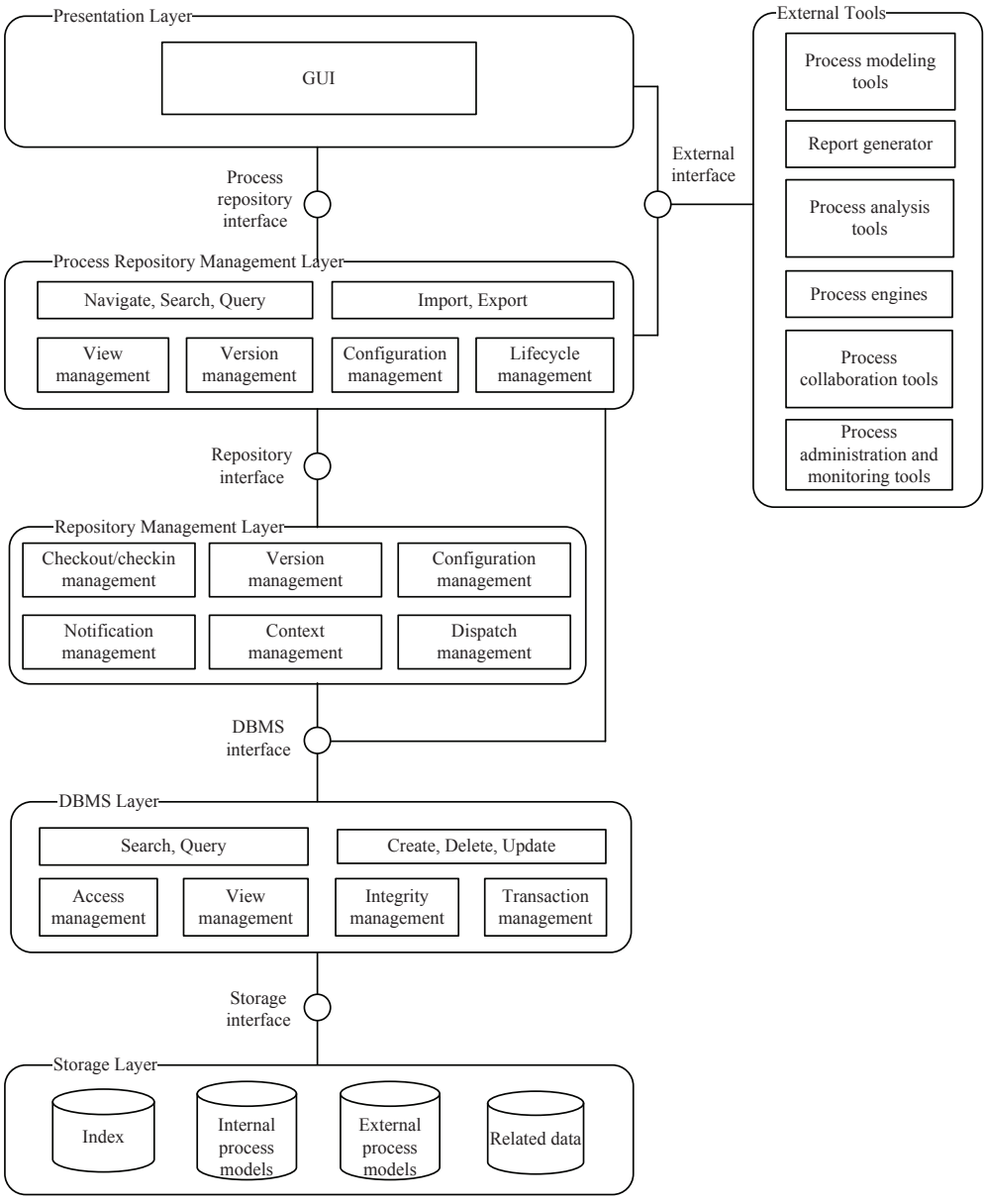

Abbildung 2.1: Referenzarchitektur des Business Process Model Repository

lieren [MWA+07]. Die Suche nach Prozessmodellen wird vom SBPR in zwei Varianten angeboten. Eine Suche unter Einbeziehung von semantischen Informationen und eine ohne [MWA+07]. Benutzer des SBPR sind ebenfalls in der Lage Modifikationen an den Prozessmodellen vorzunehmen. Eine Reihe von zusammengehörigen Modifikationen wird hierbei als “Change History“ eines Prozessmodells gespeichert. In SBPR wird diese Change History zur Versionierung verwendet. Eine Version stellt damit einen Schnappschuss in der Change History eines Prozessmodells dar [MWA+07].

2.4 Zusammenfassung

Alle drei in diesem Kapitel vorgestellten Ansätze für Prozess Repositories wurden entwickelt um mit Prozessmodellen zu arbeiten.

APROMORE wurde entwickelt um Prozessmodelle zu analysieren, zu modifizieren und neue Prozessmodelle zu erstellen. Es kann ebenfalls dazu verwendet werden um neue Prozessmodelle, aus einer Gruppe sich ähnelnder Prozessmodelle, zu entwickeln.
Das *Business Process Model Repository* Framework bietet viele Funktionen, für die Arbeit mit Prozessmodellen, an wie zum Beispiel das Erstellen und manipulieren von Prozessmodellen sowie deren Konfigurations- und Lebenszyklus Verwaltung. Ziel des *Business Process Model Repository* Frameworks ist es große Prozesskollektionen zu archivieren und zu verwalten.
Beim *Semantic Business Process Repository* (SBPR) werden, anders als bei den ersten beiden Ansätzen, die Prozessmodelle anhand von Ontologien beschrieben. SBPR bietet die Möglichkeit Modifikationen an Prozessmodellen vorzunehmen, die durch eine Versionierung nachvollziehbar oder Rückgängig gemacht werden kann.

Das zu entwickelnde Repository zielt im Gegenzug zu diesen Ansätzen auf die Verwaltung von Prozessdokumentationen und Prozesskollektionen. Es wird entwickelt um Prozessbeschreibungen zu strukturieren und um Prozessdokumentationen und ihre zugehörigen Dateien zu archivieren. Zusätzlich soll es einen schnellen Überblick über die jeweiligen Prozesse geben und sie anhand ihrer Eigenschaften effizient suchbar machen.

3

Anforderungserhebung

In diesem Kapitel wird eine Grundanalyse der Anforderungen vorgenommen, die an das System gestellt werden. Zunächst wird der Funktionsbereich des Systems grob beschrieben und daraufhin ein umfassende Anforderungsanalyse durchgeführt. Anschließend werden die bestehenden funktionalen und nicht-funktionalen Anforderungen an das System erhoben.

Das System soll die Archivierung, die Suche- und Verwaltung von Prozessdokumentationen erleichtern. Dazu soll das System Änderungsoperationen für Benutzerdaten und die Beschreibungsinformationen der Prozesse anbieten. Die Dateien der Prozessdokumentationen sollen vom System gespeichert, versioniert, kommentiert und bei Bedarf gelöscht werden können.

Die verwaltenden Prozessdokumentationen sollen den Benutzern über eine passende Benutzeroberfläche zugänglich gemacht werden. Der Zugriff auf diese Informationen

soll dabei je nach Benutzerrolle eingeschränkt werden, so dass nur autorisierte Benutzer die Prozessdokumentationen lesen beziehungsweise verändern dürfen. Das System soll hierfür auch eine dynamische Vergabe von Lese- und Schreibrechten für Prozessdokumentationen anbieten, die jedoch nur der speziellen Benutzerrolle Administrator zur Verfügung steht.

Die Suche nach Prozessdokumentationen soll in einer einfachen und einer erweiterten Variante im System vorhanden sein und muss insbesondere die Einschränkungen der jeweiligen Benutzerrolle berücksichtigen.

3.1 Anforderungsanalyse

Das System muss in der Lage sein neue Benutzer anzulegen, ihre Benutzerdaten zu ändern und sie über Änderungen zu informieren. Prozessdokumentionen und Prozesskollektionen müssen erstellt, geändert und gelöscht werden können, sowie vom Benutzer effizient durchsuchbar sein.

3.1.1 Anforderungen an die Benutzerverwaltung

Das System muss die Möglichkeit zum anlegen, ändern und löschen von Benutzern und Benutzerdaten anbieten. Die Benutzer werden dabei anhand ihrer E-Mailadresse eindeutig identifiziert und so für das System unterscheidbar gemacht.

Darüber hinaus unterliegen die Prozessdokumentationen unterschiedlichen Restriktionen bezüglich der Schreib- und Leseberechtigung der Benutzer, was eine Benutzer und Rechteverwaltung notwendig macht. Jeder Benutzer wird deshalb einer Benutzerrolle zugeteilt, welche wiederum verschiedene Schreib- und Leseberechtigungen für Prozessdokumentationen zugeordnet sind. Dadurch wird sichergestellt dass die jeweiligen Benutzer die Rechte der zugehörigen Benutzerrolle nicht überschreiten. Prozessdokumentationen für die ein Benutzer keine Schreib- oder Leserechte besitzt werden vom System verborgen und können von diesem Benutzer durch die Suche auch nicht gefunden werden. Nur Benutzer mit der Benutzerrolle *Administrator* besitzen alle Rechte.

Im System soll kein öffentlich zugängliches Registrierungsformular existieren. Aus diesem Grund erfolgt die Registrierung durch Einladungen. Das bedeutet, dass auf Wunsch eines Benutzers eine Einladungsmail verschickt wird, in der ein Link auf eine entsprechende Registrierungsseite eingebettet ist. Für die Einladung neuer Benutzer in das System sind dabei die Administratoren verantwortlich. Ein Administrator wählt dazu die Benutzergruppe aus, zu der der eingeladene Benutzer gehören soll und verschickt die Einladung an dessen E-Mail Adresse.

Benutzerrollen

Das System soll die in Tabelle 3.1 gelisteten Benutzerrollen unterstützen. In der rechten Hälfte der Tabelle sind dabei die standardmäßigen Lese- und Schreibrechte der jeweiligen Benutzer aufgeführt.

Rolle	**Schreibrechte**	**Leserechte**
Administrator	alle	alle
Editor	eigene Prozesse	alle für Prozesse
Benutzer	persönliche Daten	alle für Prozesse
Gast	persönliche Daten	Prozesse ohne Copyright
Anonym	keine	öffentliche Prozesse ohne Copyright
Ansprechpartner	keine	keine

Tabelle 3.1: Benutzerrollen

Die Rolle “Anonym“ stellt einen öffentlichen Zugang zum System dar. Über diesen können nicht registrierte Benutzer nach Prozessdokumentationen suchen, für die keine Einschränkungen der Leserechte vorliegen.

Die Rolle “Ansprechpartner“ ist eine Hilfsrolle die keinerlei Rechte besitzt. Sie wird verwendet um Personen, die im System nicht registriert sind, als Ansprechpartner für einzelne Prozessdokumentationen angeben zu können.

3.1.2 Anforderungen an die Prozessverwaltung

Prozessdokumentationen bestehen zum einen aus Metainformationen zu den jeweiligen Prozessen und zum anderen aus einer oder mehreren Dateien, in denen der Prozess, d.h. insbesondere dessen einzelne Arbeitsschritte, graphisch oder textuell beschrieben sind. Zu den Metainformationen gehören die Eigenschaften des Prozesses, sowie Allgemein- und Zusatzinformationen. Während die Allgemeininformationen einen Prozess eindeutig identifizieren und beschreiben, dokumentieren die Eigenschaften und die Zusatzinformationen den Funktionsumfang und technische Details der Prozessdokumentation. Diese Informationen spielen für das System eine wichtige Rolle. Insbesondere soll durch sie das Archiv des Systems nach gewünschten Prozessdokumentationen durchsucht werden können.

Die Verwaltung von Prozessdokumentationen und Prozesskollektionen beinhaltet zum einen die Änderungsoperationen für die Metainformationen und zum anderen das Anlegen und Löschen sowie die Versionierung der zugehörigen Dateien. Die Eigenschaften und die Zusatzinformationen die zu einer Prozessdokumentationen gehören sollen keiner Beschränkung unterliegen und dynamisch hinzugefügt, gelöscht oder geändert werden können. Diese Verwaltungsarbeit kann nur von Benutzern der Benutzerrolle “Editor“ ausgeführt werden. Zusätzlich zu der Rechteverwaltung mit Hilfe der Benutzerrollen soll zu jeder Prozessdokumentationen die Möglichkeit bestehen Benutzern dynamisch Schreib- und Leserechte zu vergeben. Diese dynamische Rechteverwaltung kann nur von Benutzern der Benutzerrolle “Administrator“ ausgeführt werden.

Mehrere verwandte Prozessdokumentationen sollen zu einer Prozesskollektion zusammengefasst werden können. Eine Prozesskollektion soll dabei wie Prozessdokumentationen Dateien, Eigenschaften, Zusatz- und Allgemeininformationen besitzen. Die Struktur der Prozesskollektionen kann hierarchisch aufgebaut sein. Das bedeutet, dass Prozesskollektionen wiederum Prozesskollektionen beinhalten können. Einzelne Prozesskollektionen ohne hierarchischen Aufbau treten ebenfalls auf. Das System soll keine Begrenzung in der Verschachtlungstiefe der Prozesskollektionen haben und soll dadurch jeden möglichen Hierarchiebaum der Prozesskollektionen abbilden können.

Metainformationen

Metainformationen sind Informationen die einen kurzen Überblick über einen Prozess ermöglichen. Durch sie können sich Benutzer des System einen schnellen Überblick über die dokumentierten Prozesse verschaffen, ohne die eigentliche Prozessdokumentation lesen zu müssen. Die vom System unterstützten Metainformationen sind in Tabelle 3.2 beschrieben.

Titel	Titel des Prozesses.
Domäne	Fachbezogene Zuordnung des Prozesses.
Beschreibung	Kurzbeschreibung des Prozesses.
Ansprechpartner	Verfasser des Prozesses oder der Prozessdokumentation.
Sprache	Sprache in der der Prozess beschrieben ist.
Copyright Notice	Urheberrecht, Einschränkungen denen der Prozess unterliegt.
Kollektion	Eine Referenz auf die Prozesskollektion der der Prozess angehört, falls vorhanden.
Eigenschaften	Auflistung der vorhandenen Eigenschaften des Prozesses.
Zusatzinformation	(Eigenschaft:Wert) Paare die die Zusatzinformationen des Prozesses angeben.

Tabelle 3.2: Metainformationen zu Prozessdokumentationen und Prozesskollektionen

Dateiverwaltung

Die Dateien einer Prozessdokumentation können in diversen Dateiformaten vorliegen und sollen dementsprechend vom System nicht eingeschränkt werden. Viele der Dateien sind normale Dokumente mit textueller Beschreibung der Prozesse, es können jedoch auch Dateien existieren die nur mit bestimmter Software zu öffnen sind. Beispiele für solche Formate in denen die Dateien vorliegen können sind pdf, jpg, txt und doc.

Die Dateien die zu einer Prozessdokumentation gehören sollen versioniert und jede Version permanent gespeichert werden. Die Versionsnummer wird hierbei von autorisierten

Benutzern manuell gesetzt. Zusätzlich soll zu jeder Version ein Kommentar existieren, der die Änderungen beziehungsweise Neuerungen an den Dateien beschreibt. Zu Jeder Version existiert nur ein Kommentar, der bei Änderung der Version übernommen oder angepasst wird.

Die vom System unterstützten Dateiformationen sind in Tabelle 3.3 beschrieben.

Titel	Titel einer Datei.
Mimetype	MIME Typ einer Datei.
Software	Software mit der die Datei geöffnet werden kann.
Versionsnummer	Versionsnummer einer Datei.
Kommentar	Interner Kommentar, zu jeder Version einer Datei vorhanden.

Tabelle 3.3: Dateiverwaltung

Prozess Suche

Das System soll eine einfache Datenbankgestützte Live-Suche nach Prozess-Titeln sowie eine erweiterte Suche nach Prozessdokumentation anhand der zu einem Prozess gehörende Eigenschaften, Zusatzinformationen, Sprache, Copyright und Domänen anbieten. Die Suche soll dabei nur diejenigen Prozessdokumentationen finden und anzeigen für die der suchende Benutzer auch das Leserecht besitzt.

3.2 Funktionale Anforderungen

In den funktionalen Anforderungen wird die Funktionalität vorgestellt die das System leisten soll. Anhand des Anwendungsfalldiagramms aus Abblidung 3.1 werden Funktionalitäten ermittelt die den Benutzern in der Benutzeroberfläche angeboten werden sollen. Anschließend werden die funktionalen Anforderungen für die Benutzer- und Prozessverwaltung aufgeführt.

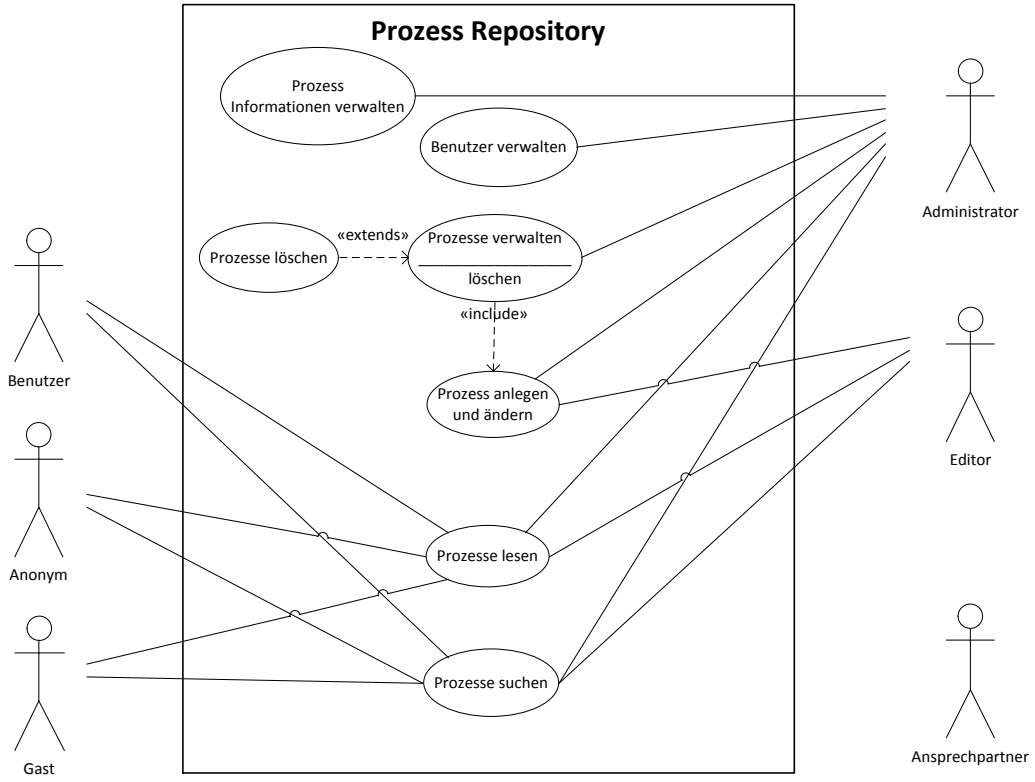

Abbildung 3.1: Anwendungsfalldiagramm

Anwendungsfalldiagramm

Ein Anwendungsfalldiagramm, oder auch als Use-Case Diagramm bezeichnet, zeigt das System aus Sicht des Benutzers. Es stellt mögliche Funktionen beziehungsweise Interaktionen des Benutzers mit dem System dar. Dies hilft einen Überblick über die erforderliche Funktionalität zu erhalten die das System anbieten soll.

Das Anwendungsfalldiagramm in Abbildung 3.1 zeigt die sechs im System existierenden Benutzerrollen. Die Benutzerrolle “Ansprechpartner“ besitzt keinerlei Rechte, da sie nur dazu verwendet wird, um Benutzer als Ansprechpartner angeben zu können, die im

System nicht registriert sind. Alle anderen Benutzerrollen haben das Recht Prozessdokumentationen zu suchen und diese zu lesen. Die Rolle “Editor“ hat zusätzlich das Recht neue Prozessdokumentationen anzulegen und bereits vom Benutzer erstellte Prozessdokumentationen zu ändern. Die Benutzerrolle “Administrator“ besitzt alle oben genannten Rechte und zusätzlich dazu das Recht Benutzer- und Prozessdaten zu verwalten, sowie neue Benutzer in das System einzuladen.
Nachfolgend werden die einzelnen in Abbildung 3.1 dargestellten funktionalen Anforderungen genauer betrachtet.

3.2.1 Benutzerverwaltung

Das System soll Funktionen für die Verwaltung von Benutzern bereitstellen. Dazu gehören Änderungsoperationen für die Benutzerdaten und ein Benachrichtigungsdienst. Insbesondere soll das System die folgenden Funktionen anbieten.

Benutzer hinzufügen

Beteiligte	Administrator
Ablauf	Der Administrator wählt in einer Auswahlliste die gewünschte Gruppe des neuen Benutzers und gibt dessen E-Mailadresse an um ihn einzuladen.

Benutzerdaten ändern

Beteiligte	Benutzer oder Administrator
Ablauf	Der Benutzer erhält eine Eingabemaske in der seine eigenen Daten dargestellt sind. Ändert der Benutzer ein Datum und klickt auf speichern so werden die geänderten Daten in die Datenbank übernommen.

Benutzerkonto aktivieren

Beteiligte	Administrator
Ablauf	Der Administrator erhält in der Auflistung aller Benutzer einen Button, welcher bei einem Klick das Konto des Benutzers wieder aktiviert.

Benutzerkonto deaktivieren

Beteiligte	Administrator
Ablauf	Der Administrator erhält in der Auflistung aller Benutzer einen Button, welcher bei einem Klick das Konto des Benutzers deaktiviert.

Benutzer benachrichtigen

Beteiligte	System
Ablauf	Werden Benutzerdaten geändert, so erfolgt der Versand einer E-Mail an den Benutzer.

Da einige Benutzer als Verfasser von Prozessdokumentationen eingetragen werden, können sie nicht einfach aus dem System gelöscht werden, da sonst Datenbankinkonsistenzen entstehen. Daher wird das entsprechende Benutzerkonto nicht gelöscht sondern nur deaktiviert und das Konto bleibt weiter im System vorhanden.

3.2.2 Prozessverwaltung

Für die Metainformationen sollen Änderungsoperationen vom System angeboten werden. Zusätzlich soll eine Suche nach Prozesstitel und Prozess Eigenschaften im System vorhanden sein. Die Dateien von Prozessdokumentationen sollen gespeichert, versioniert und gelöscht werden können.

Folgende Funktionen soll das System unterstützen um die Prozessdokumentationen zu verwalten.

Prozess hinzufügen

Beteiligte	Editor oder Administrator
Ablauf	Der Benutzer gibt die Prozessdaten in die dafür vorgesehene Eingabemaske ein und klickt auf speichern, um die Daten in der Datenbank zu hinterlegen.

Prozess ändern

Beteiligte	Editor, Administrator oder Benutzer mit zusätzlicher Schreibberechtigung
Ablauf	Der Benutzer ändert die Prozessdaten in einer Eingabemaske, wobei ihm bereits existierende Prozessdaten angezeigt werden, und klickt anschließend auf speichern, um die Daten in der Datenbank zu hinterlegen.

Prozess löschen

Beteiligte	Editor oder Administrator
Ablauf	Der Benutzer klickt auf einen Button um die Prozessdaten aus der Datenbank zu löschen.

Prozess suchen

Beteiligte	Alle Benutzer
Ablauf	Nach Eingabe eines Begriffes in das Suchfeld, wird die Datenbank nach Prozessen durchsucht, die der Eingabe entsprechen.

Dateien hinzufügen

Beteiligte	Editor, Administrator oder Benutzer mit zusätzlicher Schreibberechtigung
Ablauf	Der Benutzer wählt in einer Eingabemaske eine Datei aus, die anschließend als neue Datei in Datenbank gespeichert und dem aktuellen Prozess zugeordnet wird.

Dateien versionieren

Beteiligte	Editor, Administrator oder Benutzer mit zusätzlicher Schreibberechtigung
Ablauf	Für eine bereits in der Datenbank existierende Datei wird eine Eingabemaske angeboten, in der ein Benutzer eine Datei auswählen und nach dem senden als neue Dateiversion anlegen kann.

Dateien löschen

Beteiligte	Editor, Administrator oder Benutzer mit zusätzlicher Schreibberechtigung
Ablauf	Der Benutzer klickt auf einen Button innerhalb der Dateiliste, um die Datei aus der Datenbank zu löschen.

Die Funktion “Prozess ändern“ beinhaltet das erstellen, hinzufügen und entfernen der zu einem Prozess gehörenden Daten wie Eigenschaften, Zusatzinformationen, Ansprechpartnern, Domänen und weiteren Prozessdokumentationen. Sollte die Funktion “Dateien löschen“ ausgeführt werden, so hat dies zur Folge das alle Versionen der Datei ebenfalls gelöscht werden. Dies gilt auch für Prozessdokumentationen, wird eine Prozessdokumentation gelöscht so werden auch alle zur Prozessdokumentation gehörenden Daten kaskadierend entfernt.

3.3 Nicht-Funktionale Anforderungen

Unter dem Begriff “Nicht-Funktionale Anforderungen“ sind Anforderungen an das System zu verstehen, die keine Funktionen repräsentieren. Sie machen Aussagen über

Eigenschaften oder das mögliche Verhalten des Gesamtsystems bei seiner Ausführung. Folgende nicht-funktionale Anforderungen soll das System unterstützen.

Sicherheit

Das System soll mehrere Sicherheitsmechanismen enthalten um den verbreitetsten Angriffsmethoden im Web, wie “SQL Injection“,“Cross-Site-Scripting“ oder “Cross-Site-Request-Forgery“ entgegenzuwirken.

Wartbarkeit

Das System soll gut strukturiert und modular implementiert sein, um eine effiziente Wartbarkeit zu gewährleisten und gegebenfalls die Einarbeitungszeit zu minimieren.

Benutzbarkeit

Die graphische Oberfläche des System soll so einfach wie möglich gehalten werden und intuitiv zu bedienen sein.

Stabilität

Das System soll auf Eingabefehler der Benutzer geeignet reagieren und im Fehlerfall Hilfestellung leisten.

4

Technologie Auswahl

Dieses Kapitel gibt einen kleinen Überblick über existierende Webframeworks und führt die Technologien ein, die für die Implementierung des Prozess Repository ausgewählt wurden. Server-seitig viel die Wahl dabei auf das Ruby on Rails Framework [rai] mit einer PostgreSQL Datenbank [pos]. Client-seitig wurde Zurb Foundation 5 [zur] und Coffeescript [cof] gewählt. Als Softwarearchitektur viel die Wahl auf das Model-View-Controller Entwurfsmuster. Nach einer allgemeinen Einführung wird das Ruby on Rails Framework vorgestellt, die Prinzipien dahinter erklärt und die Funktionsweise einiger ausgewählter Module erläutert.

4.1 Überblick Webframeworks

Dieser Abschnitt stellt einige Frameworks zur Entwicklung von Webapplikationen vor, die für die Umsetzung des Prozess Repository in Frage kommen. Derartige Frameworks haben den Vorteil, dass sie die Entwicklung von Applikationen unterstützen und somit den Entwicklungsprozess beschleunigen. Es wird das PHP Framework CodeIgniter vorgestellt, sowie das Google Web Toolkit welches Java als Programmiersprache nutzt. Zusätzlich wird auf die Node.js Plattform und das Framework Ruby on Rails eingegangen.

CodeIgniter

CodeIgniter ist ein auf PHP basierendes Open-Source Framework [cod]. Es ist schlank gehalten, leicht zu erlernen und enthält viele Bibliotheksklassen für häufig benötigte Aufgaben, wie beispielsweise Datenbankzugriff. CodeIgniter ist im Vergleich zu anderen PHP Frameworks sehr schnell und die Konfiguration sehr einfach. Bei der Architektur verwendet CodeIgniter das Model-View-Controller Entwurfsmuster um seine Projekte zu Strukturieren. Ziel von CodeIgniter ist es eine Entwicklung von Web Applikationen zu ermöglichen, die schneller ist als die Entwicklung von Grund auf [cod].

Google Web Toolkit

Das Google Web Toolkit (“GWT“) ist ein Framework um komplexe Browser basierte Applikationen zu entwickeln [gwt]. Entwickelt wird beim GWT ausschließlich in der Programmiersprache Java. Der fertige Programm-Code wird am Ende von einem Java-to-Javascript Compiler in Javascript Code übersetzt und anschließend client-seitig zum Einsatz gebracht. Der Vorteil von GWT ist das Java zur Entwicklung eingesetzt wird. Dadurch können alle Vorteile von Java, wie zum Beispiel Objektorientierung, verwendet und existierende Bibliotheken sowie Entwicklungsumgebungen eingesetzt werden. Das Ziel von GWT ist es die Entwicklung von hochperformanten Web Applikationen zu ermöglichen, ohne das die Entwickler sich mit Javascript und Ajax auskennen müssen.

Node.js

Node.js ist eine Plattform zur Entwicklung von schnellen und skalierbaren Netzwerk Applikationen [nod]. Insbesondere können mit Node.js leistungsstarke Webserver entwickelt werden. In Node.js wird client- und server-seitig Javascript als Programmiersprache eingesetzt. Dies hat den Vorteil das bei der Entwicklung von Web Applikationen weniger Programmiersprachen zu Einsatz kommen. Auf dem Server läuft der Javascript Code in der von Google entwickelten V8-Laufzeitumgebung [v8e]. Bei V8 handelt es sich um einen Just-in-Time Compiler der Javascript Code bei der Ausführung in nativen Maschinencode übersetzt. Node.js arbeitet ereignisgesteuert und blockiert keine Ressourcen bei Anfragen an die Datenbank, wodurch sehr viele Anfragen schnell abgearbeitet und Ressourcen gespart werden können. Die Schnelligkeit von Node.js ist vor allem bei Applikationen, die große Datenmengen mit der Datenbank austauschen, deutlich zu erkennen.

Ruby on Rails

Ruby on Rails (“Rails“) ist ein Framework zur schnellen und sicheren Entwicklung von Web Applikationen [rai]. Die Programmierung findet dabei in Ruby statt. Rails bietet viele Funktionen wie Migrationen, Scaffolding und OR-Mapping an [rai], um die Realisierung von Web Applikationen schnell und für die Entwickler angenehm zu machen. Dabei setzt Rails bei der Architektur auf das MVC-Entwurfsmuster und bringt viele weitere erprobte Technologien mit, die die Entwicklung erleichtern.

Verwendetes Framework

Für die Implementierung des Prozess Repository viel die Wahl schlussendlich auf das Ruby on Rails Framework, da mit Rails schnell, sichere Webanwendungen realisiert werden können und die Entwicklung sehr angenehm ist.

4.2 Verwendete Technologien

Das Prozess Repository wird mit den folgenden Technologien entwickelt. Zur server-seitigen Programmierung wird das Framework “Ruby on Rails“ verwendet mit PostgreSQL als Datenbank für die Datenspeicherung. Client-seitig werden Technologien wie Ajax, Sass und Responsive Design zum Einsatz gebracht. Implementiert wird das Prozess Repository dabei nach dem Model-View-Controller Entwurfsmuster.

4.2.1 MVC-Entwurfsmuster

Das Model-View-Controller Entwurfsmuster (“MVC“) ist ein Architekturprinzip in der Softwareentwicklung. Es wird verwendet um die Projektdaten in der Entwicklung zu Strukturieren und sie nach Verantwortlichkeiten einzuteilen. Das Model ist dabei für die Datenspeicherung und Wiederbeschaffung verantwortlich. Die View für die Repräsentation der Daten und der Controller für die Programmlogik. Ziel des MVC-Endwurfmusters ist es eine lose Kopplung zwischen den Komponenten des Systems zu schaffen um eine spätere Änderung oder Erweiterung der Applikation zu erleichtern. Das MVC-Entwurfsmuster wird in späteren Kapitel ausführlicher betrachtet.

4.2.2 Ruby

Als server-seitige Programmiersprache wird Ruby [rub] eingesetzt. Ruby ist eine rein objektorientierte Programmiersprache und mit dem in Ruby programmierten Framework “Ruby On Rails“ [rai] ein mächtiges Werkzeug in der Webprogrammierung. In das Ruby On Rails Framework wurden viele erprobte Methoden der Webentwicklung integriert. Das führt dazu dass allein durch die Verwendung von Ruby On Rails bereits verschiedene aktuelle Technologien in die Entwicklung eingehen. Zusätzlich nimmt Ruby On Rails dem Entwickler einige Implementierungsarbeit für die nicht-funktionalen Anforderungen ab. Insbesondere existieren bereits diverse Sicherheitsmechanismen, die zum Beispiel vor SQL-Injection, Cross- Site- Scripting oder vor Cross-Site-Request-Forgery schützen.

Auch die Wartbarkeit wird durch das Framework gefördert, da Ruby On Rails Projekte alle die gleiche Struktur aufweisen und der Programmablauf immer identisch ist.

4.2.3 Coffeescript

Als client-seitige Programmiersprache wird Javascript verwendet und hier insbesondere die Javascript Bibliothek JQuery [jqu]. Geschrieben wird der Javascript Code, der zur Anwendung gebracht wird, jedoch in Coffeescript [cof]. Coffeescript bietet eine leichtgewichtigere Syntax als Javascript und ist daher angenehmer zu lesen und zu programmieren [cof]. Dies erleichtert die Codeübersicht und damit die Entwicklung bzw. Fehlerfindung. Die Coffeescript Dateien werden von einem Compiler in Javascript Code übersetzt und auf dem Client zur Ausführung gebracht. Dieser Coffeescript-to-Javascript Compiler ist bereits Bestandteil von Ruby On Rails [rai].

4.2.4 Ajax

Eine weitere Technologie, die im Repository sehr oft angewendet wird, ist das auf Javascript basierte Ajax [G+05]. Ajax steht für “Asynchronous JavaScript and XML“. Mit Ajax ist es möglich Anfragen an den Server zu schicken und die Daten, die in der Antwort enthalten sind, in die bestehende HTML Seite zu integrieren ohne das die Seite neu geladen werden muss. Ajax selbst ist allerdings nicht XML abhängig, wie der Name vermuten lässt. Im Prozess Repository werden insbesondere nur JSON Daten zwischen Client und Server per Ajax ausgetauscht.

4.2.5 Zurb Foundation 5

Für die Gestaltung des Repositories wird ein so genanntes CSS-Framework Namens “Foundation 5“ genutzt [zur]. CSS steht für “Cascading Style Sheets“, es wird benutzt um die Elemente einer HTML Seite zu positionieren, zu animieren oder ihr Aussehen zu verändern. Foundation 5 bringt eine große Anzahl an CSS Klassen mit, mit deren Hilfe

die HTML Elemente verändert werden können. Zusätzlich arbeitet es mit einem Zeilen-Spalten Konzept um die Elemente zu positionieren.

4.2.6 Responsive Design

Responsive Design [Zil12] ist ebenfalls eine auf CSS basierte Art Webseiten zu gestalten. Hierbei handelt es sich um eine Technik die je nach Bildschirmgröße des Endgerätes andere CSS-Regeln auf die HTML Seite anwendet. Dies erhöht die Benutzbarkeit von Webseiten für Benutzer mit Mobilen Endgeräten, da sie je nach Bildschirmgröße passend dargestellt werden können. Responsive Design ist in Foundation 5 integriert und beruht auf dem vorhandenen Zeilen- Spalten Konzept.

4.2.7 Sass

Sass steht für “Syntactically Awesome Stylesheets“ [sas]. Hierbei handelt es sich um eine Erweiterung des CSS Sprachraums, welcher die Erzeugung von CSS Code erleichtert. Sass erweitert die Funktionalitäten von CSS unter anderem um Variablen, Codeblock Verschachtelung und Vererbung. Diese Erweiterungen bieten den Entwicklern viele Vorteile, da der CSS Code beim programmieren kleiner und übersichtlicher ist [MO12]. Ebenfalls kann ein Zeitgewinn bei notwendigen Änderungen des Designs entstehen. Für die Verarbeitung verwendet Sass einen Präprozessor, der in Ruby On Rails [MO12] bereits integriert ist.

4.2.8 PostgreSQL

Zur Speicherung der Daten wird die PostgreSQL Datenbank [pos] verwendet. PostgreSQL ist ein freies objektrelationales Datenbankmanagementsystem und ist weitgehend konform mit dem SQL-Standard [pos]. Dies hat zur Folge das viele Funktionen wie sie in SQL definiert sind, in PostgreSQL definitionsgemäß umgesetzt wurden. PostgreSQL unterstützt somit SQL Funktionen wie Joins, Views, Trigger oder Stored Procedures.

Intern in SQL Datenbanken werden die Daten in Tabellen gespeichert, die Zeilen- und Spaltenweise organisiert sind.

4.3 Ruby on Rails

Ruby On Rails oder kurz “Rails“ ist ein quelloffenes Framework zur schnellen und sicheren Entwicklung von Webapplikationen [rai]. Die Architektur einer Rails Applikation basiert auf dem Model-View-Controller Entwurfsmuster und ist in Rails auch fest vorgegeben. Das gesamte Framework ist geprägt von den Prinzipien `don't repeat yourself` (DRY) und `convention over configuration` (Konvention vor Konfiguration) die im weiteren Verlauf noch näher erläutert werden [MO12].

Rails Architekturdiagramm

Das Architekturdiagramm in Abbildung 4.1 ist ein Ausschnitt der Architektur einer Ruby on Rails Applikation. Da sie fest vorgegeben ist, sind alle Rails Applikationen von ihrer Architektur her gleich. Ein normaler Webseitenaufruf im Browser läuft daher folgendermaßen ab. Der Client schickt eine Anfrage an den Server, welcher sie an den Action Dispatcher weiterleitet. Der Action Dispatcher vergleicht die URL der Anfrage mit seinen Routing Einträgen und leitet bei einem Treffer die Anfrage an den jeweiligen Controller weiter. Um im Controller an die Daten aus der Datenbank zu kommen, werden die Model Klassen des Active Record Frameworks verwendet. Diese Model Klassen enthalten Methoden mit denen SQL Statements an die Datenbank geschickt werden können. Falls das Resultat der SQL Anweisung Daten enthält, so werden sie der View weitergereicht und dort gerendert. Die so erstellte Seite wird dann als Antwort auf die Anfrage, an den Client geschickt. E-Mailversand und Weiterleitungen gehören ebenfalls zu den Funktionen die in einem Controller ausgeführt werden können.

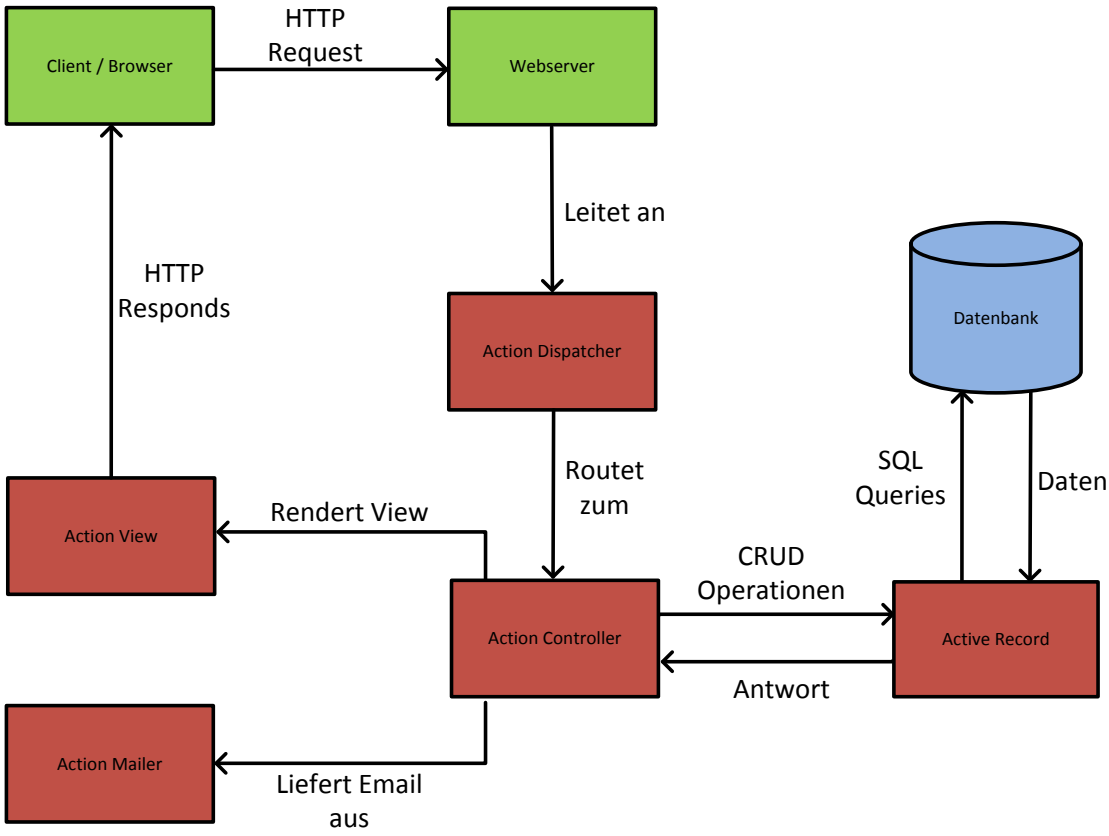

Abbildung 4.1: Architekturdiagramm einer Rails Applikation

4.3.1 Ruby on Rails Prinzipien

Hier werden die von Rails verfolgten Prinzipien `don't repeat yourself` und `convention over configuration` vorgestellt, da diese einen erheblichen Einfluss auf die spätere Realisierung des Prozess Repositories haben.

Convention Over Configuration

Frameworks für die Entwicklung von Applikationen, die ohne dem Prinzip der Konventionen arbeiten, verwenden Konfigurationen um alle Einstellungen in der Applikation vorzunehmen. Konfigurationen steuern dabei den Ablauf der Arbeitsschritte die in ei-

ner Applikation stattfinden, legen Sicherheitsmechanismen fest oder geben an wie Verbindungen zu einer Datenbank aufgenommen werden soll.

Rails folgt dem Prinzip der Konvention vor Konfiguration [MO12]. Erklären kann man die Funktionsweise dieses Prinzips damit das sinnvolle Standardkonfigurationen existieren die das Zusammenspiel der Rails Komponenten steuern. Ein Beispiel für eine Konvention ist, dass die View Dateien immer so heißen wie ihre zugehörige Methode im Controller. Zu der Methode `new` gehört damit die View `new.html.erb`. Ein weiteres Beispiel für Rails Konventionen ist, das Model Dateien immer gleich benannt werden wie die zugehörige Datenbanktabelle. Während der Name der Datenbanktabelle im Plural angegeben ist wird die Model Klasse jedoch im Singular angegeben [MO12]. Zu der Datenbanktabelle `users` gehört demnach die Model Klasse `User.rb`.

Das Prinzip `convention over configuration` hat den großen Vorteil das die Konfigurationsdateien der Applikationen kleiner werden, da sich die Menge der Konfigurationen die vorgenommen werden müssen reduziert.

Don’t Repeat Yourself

Das Prinzip `don’t repeat yourself` steht für die Vermeidung von Redundanzen im Programm-Code [MO12]. Programm-Code der öfter vorkommt ist schwieriger zu Warten und nicht selten eine Fehlerquelle wenn später Modifikationen am Code vorgenommen wurden. Das Rails Framework stellt mehrere Mechanismen bereit um Redundanzen im Programm-Code zu vermeiden. Für den Programm-Code der in Controllern ausgeführt wird stellt Rails Hilfs- Methoden bereit in die redundanter Code ausgelagert werden kann. Bei den Views werden sogenannte Partials zur Verfügung gestellt, die die gleiche Funktion wie die Hilfs- Methoden erfüllen nur das sie normalerweise HTML Code zum Ergebnis haben. Zusätzlich wird das oft gleichbleibende Webseitenlayout in Layout Dateien ausgelagert und nur die sich ändernden Elemente dynamisch angepasst [MO12].

4.3.2 Ausgewählte Rails Module

Rails beinhaltet mehrere Module in die jeweils bestimmte Funktionalitäten des Frameworks ausgelagert sind. In diesem Abschnitt werden die Module Action Dispatcher, Active Record, Action Controller und Action View vorgestellt und ihre Funktionen erklärt.

Action Dispatcher

Dieses Modul ist für das Routing verantwortlich [MO12]. Es leitet eine HTTP Anfrage an den zugehörigen Controller und die zugeordnete Methode weiter. Ein Beispiel kann folgendermaßen Aussehen: ein Request mit der Url `repository.de/user/login` wird an den Controller `Users` und dort an die enthaltene Methode `login` weitergeleitet. Alle Routingregeln werden in der Datei `routes.rb` verwaltet [MO12].

Active Record

Active Record ist ein Persistenz Framework welches in der Lage ist eine Verbindung zwischen dem objektorientierten Klassenmodell von Rails und den Daten aus einer Datenbank herzustellen [MO12]. Eine Datenbanktabelle wird hierbei als eine Klasse abgebildet, eine Zeile in der Datenbanktabelle als ein Objekt und eine Spalte als Attribut des Objekts. Das Modul Active Record stellt somit das Model im Model-View-Controller Konzept dar und ist für die Beschaffung der Daten und ihre Validierung verantwortlich [MO12].

Action Controller

Dieses Modul ist das Bindeglied zwischen Active Record und der View Komponente im MVC-Konzept [MO12]. Der Controller nutzt die Objekte des Models, um die benötigten Daten zu erhalten, die er dann der View zur Darstellung übergibt. Dies geschieht in Rails indem die Daten in Instanzvariablen gehalten werden, auf die die Views ebenfalls Zugriff haben. Im Controller ist die gesamte Programmlogik zu finden.

Action View

Die Daten die der Controller übergibt werden in diesem Modul dargestellt und danach in das gewählte Layout eingebettet [MO12]. Die View kann die Daten in mehreren Formaten ausgeben, der Standard ist HTML, jedoch ist auch XML oder JSON möglich [MO12].

5

Implementierung

In diesem Kapitel wird auf die Implementierung des Prozess Repository eingegangen. Zuerst wird die zugrundeliegende Systemarchitektur vorgestellt und veranschaulicht. Anschließend wird der Aufbau der internen Datenbankarchitektur anhand von ER- Diagrammen dargestellt und textuell beschrieben. Die Funktionsweise und Verwendung der Controller, in denen die Programmlogik implementiert ist, wird erläutert und anhand von Beispielen veranschaulicht. Zusätzlich führt dieses Kapitel einige Mockups ein, auf deren Basis die Benutzerschnittstellen implementiert sind und geht auf ihre Implementierung ein.

5.1 Systemarchitektur

In diesem Abschnitt wird die dem Prozess Repository zugrunde liegende Systemarchitektur vorgestellt und veranschaulicht. Es wird auch auf das verwendete MVC-Entwurfsmuster eingegangen und erklärt.

Architektur

Das System wird nach dem Model-View-Controller Entwurfsmuster (“MVC“) in einer Client/Server-Architektur implementiert. Durch das MVC Entwurfsmuster wird dabei der Programm-Code in Verantwortlichkeitsbereiche eingeteilt, wodurch beispielsweise eine bessere Wartbarkeit erreicht werden kann.

Das *Model*, im MVC Entwurfsmuster, regelt die Verwaltung der Daten. Es ist verantwortlich für die Datenspeicherung in einer Relationalen Datenbank, für die Datenbeschaffung und für die Implementierung der Änderungsoperationen, die auf den Daten operieren. Dies geschieht in Model Klassen, deren Objekte die jeweiligen Daten einer Zeile in einer Datenbanktabelle repräsentieren.

Die *View* ist die Benutzerschnittstelle des System und ist verantwortlich für die Darstellung der ihr übergebenen Daten. Durch sie können die Benutzer die im System gehaltenen Prozessdokumentationen suchen, ansehen oder herunterladen. Sie ist ebenfalls dafür zuständig den Benutzern auf mögliche Fehler und auf mangelnde Autorisierung hinzuweisen. Alle Interaktionen zwischen einem Benutzer und dem System laufen schlussendlich über die View.

Der *Controller* ist das Bindeglied zwischen Model und View. In ihm steckt die eigentliche Programmierlogik des Systems. Der Controller ist Verantwortlich für die Beschaffung von Daten, die für das System relevant sind, um sie dann der View zur Darstellung zu übergeben. Eine weitere Funktion besteht darin Benutzer, durch das Versenden von E-Mails, mit Instruktionen oder Informationen zu versorgen.

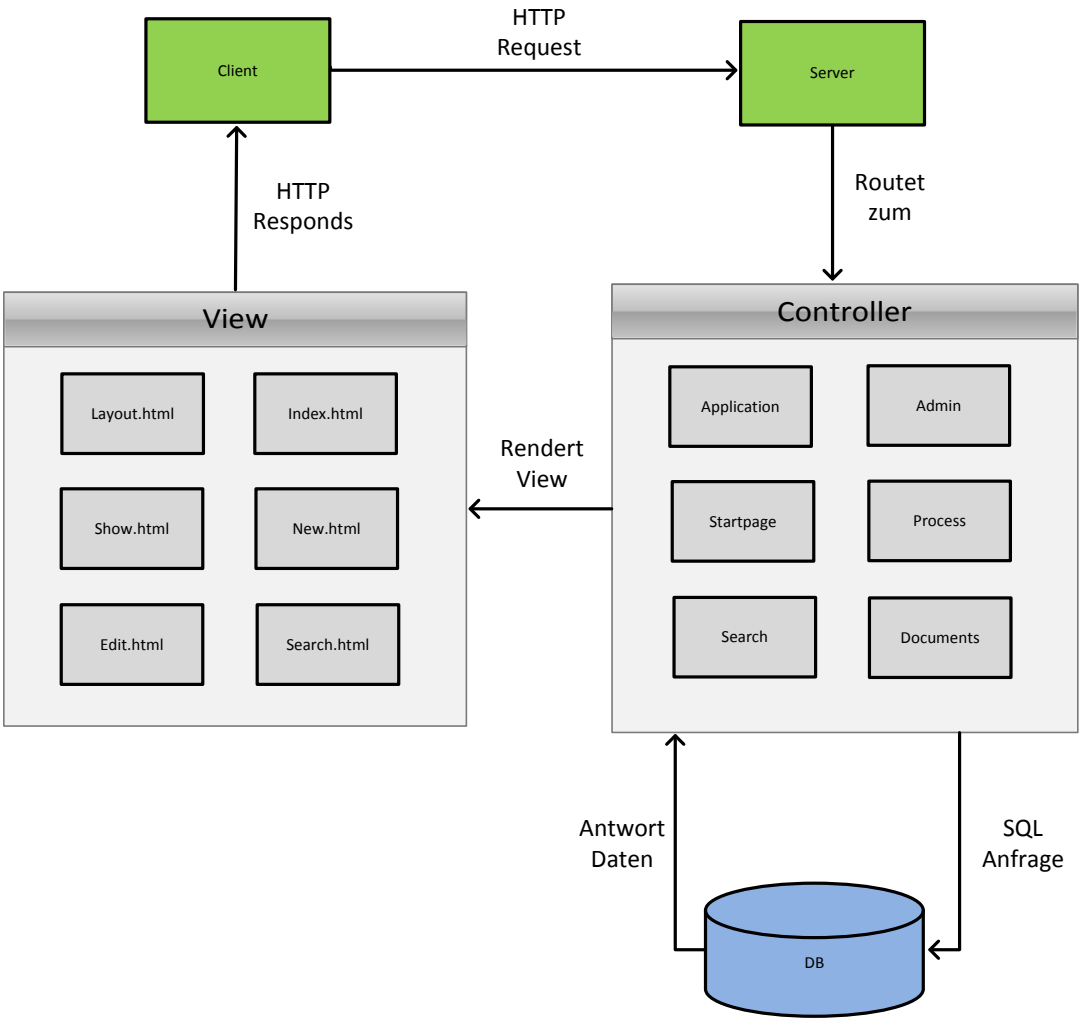

Abbildung 5.1: Vorläufiges Architekturdiagramm des Prozess Repositories

Architekturdiagramm

Abbildung 5.1 zeigt die Architektur auf, die dem System zugrunde liegt. Das System basiert auf einer klassischen Client/Server Architektur, auf der die meisten Webseiten aufbauen. Der Client schickt dabei, über das HTTP- Protokoll, eine Anfrage an den Server, um auf die gewünschte Seite Zugriff zu erhalten. Handelt es sich bei der angefragten Webseite um eine dynamisch generierte Seite, so ruft der Server einen Controller auf der eine SQL Anfrage an die Datenbank schickt. Bekommt der Controller die angefragten Daten von der Datenbank, so übergibt er sie der View in der sie dann dargestellt werden. Die Ergebnisseite wird anschließend an den Client übertragen, wo sie anzeigt und nach belieben neue anfragt werden kann.

Die Klassen der Controller und der Views, welche in Abbildung 5.1 dargestellt sind, werden in späteren Kapiteln genauer betrachtet.

5.2 Model

Das Model ist für die persistente Datenhaltung zuständig. Es ist verantwortlich für die Speicherung und Wiederbeschaffung der Daten.

5.2.1 Datenbankentwurf

Im Datenbankentwurf wird die Architektur der Datenbank festgelegt in der die Daten des Prozess Repository schlussendlich gespeichert werden. Für die Darstellung der Datenbankarchitektur werden Entity-Relationship-Diagramme [Che76] verwendet.

Im Entity-Relationship-Diagramm [Che76] (ER-Diagramm) wird die Datenbankarchitektur mit Hilfe der Entitäten und ihrer Beziehungen zueinander definiert und dargestellt. Eine Entität symbolisiert dabei ein Element, das in der Datenbank gespeichert wird. Dies dient zum einen der effizienteren Modellierung und zum anderen der besseren Nachvollziehbarkeit der Datenbankarchitektur.

In den folgenden Abschnitten werden die ER-Diagramme des Prozess Repository vorgestellt. Die Attribute der zugehörigen Entitäten werden aufgrund ihrer Anzahl nicht im ER-Diagramm aufgeführt. Eine detaillierte Liste der Attribute folgt in der anschließenden Beschreibung der Datenbank.

ER-Diagramm des Systems

Die Grobstruktur der Datenbankarchitektur ist in Abbildung 5.2 veranschaulicht. Jede Entität stellt dabei eine Tabelle in der Datenbank dar, in der die Daten persistent gehalten werden. Textuell beschrieben zeigt das Diagramm folgendes.

Die Tabelle Benutzer verwaltet die im System registrierten Benutzer. Benutzer gehören zu genau einer Gruppe. Ferner können Benutzer eine oder mehrere Prozesse erstellen. Dabei können es einzelne Prozesse oder Prozesskollektionen sein die erstellt werden, beides wird in der Datenbanktabelle Prozesse abgebildet. Die Tabelle Prozesse verwaltet hierbei die Prozessbeschreibung der einzelnen Prozesse.
Ein Benutzer kann Ansprechpartner für viele Prozesse sein und ein Prozess kann mehrere Ansprechpartner haben. Die Tabelle Ansprechpartner verwaltet dabei die Zuordnungen zwischen Prozessen und Ansprechpartnern.
Einem Benutzer können für verschiedene Prozesse zusätzliche Schreib- und Leserechte gewährt werden und ein Prozess kann viele Benutzer besitzen die zusätzliche Schreib- und Leserechte haben. In der Tabelle Zusätzliche Rechte wird diese dynamisch Rechtezuweisung verwaltet.
Zu einem Prozess können mehrere Dateien gehören und jede Datei kann mehrere Versionen besitzen. In der Tabelle Dateien werden dabei die Informationen über die Dateien gespeichert und die eigentliche Dateien in der Versionen Tabelle hinterlegt. Jede Datei gehört zu genau einer Software mit der man die zugehörige Datei öffnen kann. Die Tabelle Software verwaltet die Namen der Software die zum öffnen von Dateien benötigt wird.

Die Tabelle “Prozesse“ ist der Kern der Datenbankarchitektur, sie besitzt noch weitere Assoziationen die in Abbildung 5.3 dargestellt sind.

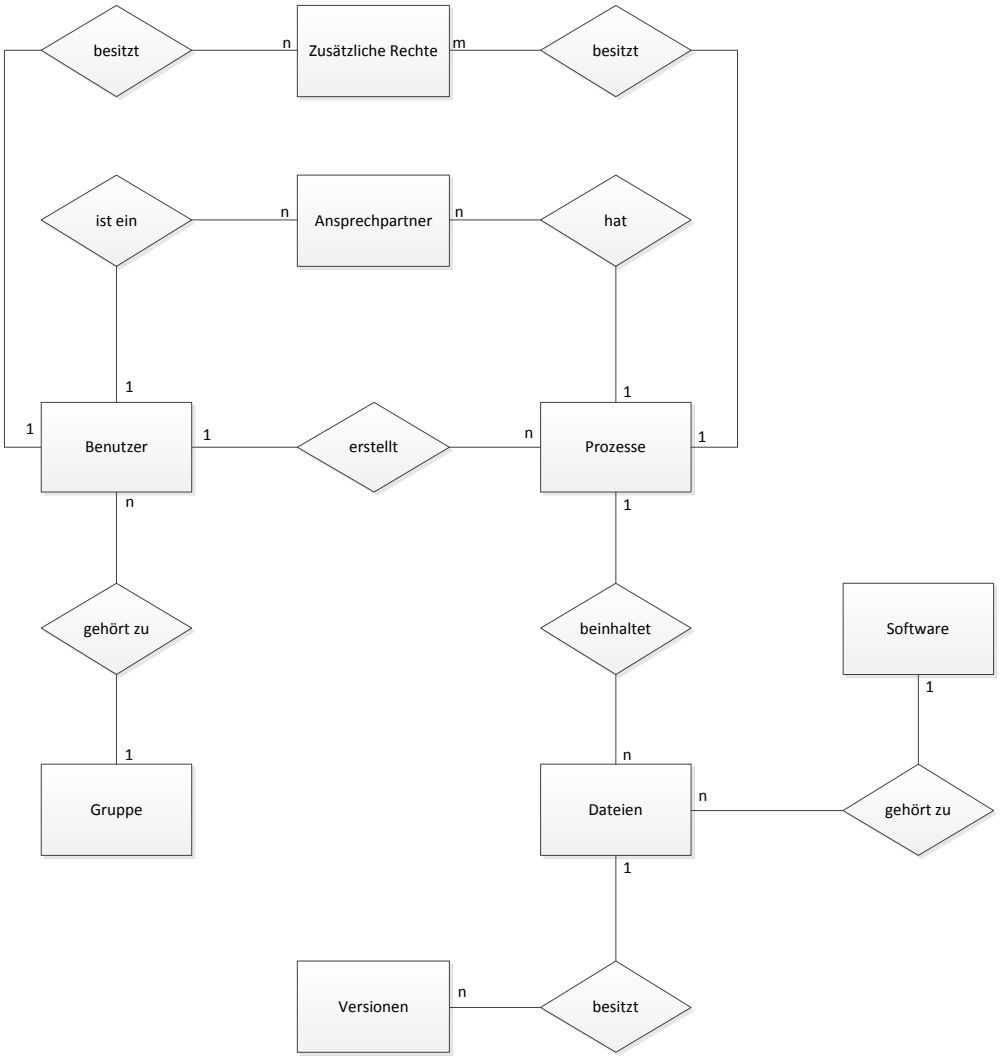

Abbildung 5.2: ER-Diagramm des Systems

ER-Diagramm der Prozesse

Im Diagramm in Abbildung 5.3 werden weitere Assoziationen mit der Prozess Tabelle dargestellt. Folgende Assoziationen werden beschrieben.

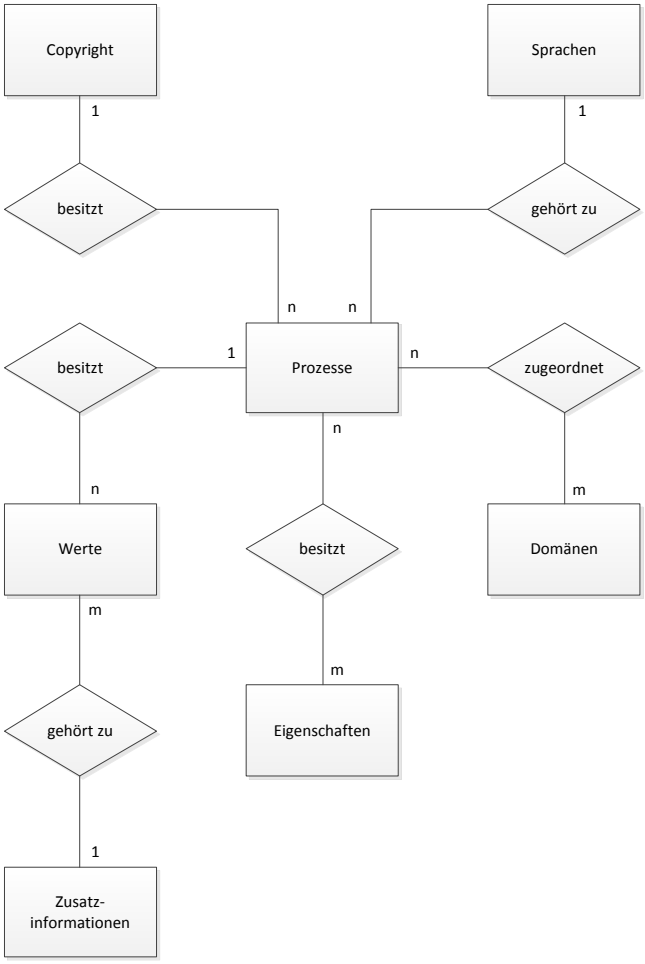

Abbildung 5.3: Weiter Assoziationen mit der Prozesstabelle

Prozessen ist eine Sprache zugeordnet und sie unterliegen einem bestimmten Urheberrecht (Copyright). Die Tabelle Sprache ist hierbei für die Verwaltung der Sprachen

zuständig und die Tabelle Copyright für die Verwaltung von Urheberrechtsinformationen. Prozesse können mehreren Domänen zugeordnet seien und eine Domänen kann viele Prozesse enthalten. In der Tabelle Domänen werden die Namen der Domänen hinterlegt. Prozesse können viele Eigenschaften und Zusatzinformationen, die in der Prozessdokumentation beschrieben sind, besitzen und eine Eigenschaft beziehungsweise Zusatzinformation kann zu mehreren Prozessen gehören. Die Tabellen Eigenschaften und Zusatzinformationen verwalten hierbei Informationen über die Prozesse.
Einem Prozess und einer Zusatzinformation können mehrere Werte zugeordnet sein. In der Tabelle Werte werden die Werte zu den zugehörigen Zusatzinformationen hinterlegt und die Zuordnung zu den Prozessen vorgenommen.

5.2.2 Beschreibung der Datenbank

In diesem Abschnitt werden die einzelnen Datenbanktabellen vorgestellt und ihr Verwendungszweck beschrieben. Dabei wird auf die einzelnen Attribute der Entitäten (d.h. die Tabellenspalten) eingegangen.

Benutzer

In dieser Tabelle werden die registrierten Benutzer des Systems gespeichert (siehe Tabelle 5.1). Ein Benutzer besitzt eine E-Mailadresse und ein Passwort mit denen er sich am System anmeldet. Zusätzlich wird hier Gruppen des Benutzers und der Status des Kontos gespeichert.

Gruppe

In Tabelle 5.2 werden die möglichen Benutzergruppen aufgelistet. Eine Gruppe wird hierbei eindeutig durch eine ID repräsentiert und mit einem Namen versehen. Insgesamt existieren die folgenden sechs Benutzergruppen: Administrator, Editor, Benutzer, Gast, Anonym und Ansprechpartner.

Spalte	Feldtyp	Beschreibung
Id	Integer	Primärer Schlüssel dieser Tabelle.
Name	Varchar(255)	Name des Benutzers.
Email	Varchar(255)	Email des Benutzers, wird zum Anmelden verwendet.
Passwort	Varchar(30)	Passwort des Benutzers (als Hash-Wert).
Gruppe	Integer	Gruppe des Benutzers (Rechteverteilung / Fremdschlüssel).
Aktiv	Boolean	Benutzer als gelöscht markiert oder nicht.

Tabelle 5.1: Benutzer

Spalte	Feldtyp	Beschreibung
Id	Integer	Gruppennummer.
Name	Varchar(13)	Name der Gruppe.

Tabelle 5.2: Gruppe

Registrierung

Diese Tabelle wird benötigt um die Registrierung im System sicherer zu gestalten (siehe Tabelle 5.3). Da die Registrierung nur per Einladung erfolgen soll, darf sie für alle anderen nicht zugänglich sein. Bei der Registrierung wird dem Benutzer daher eine URL mit angehängtem eindeutigen Hash-Wert zugesandt über die dieser sich registrieren kann. Vor dem Zugriff auf die Registrierung wird geprüft ob ein Hash-Wert in der URL vorhanden ist und in dieser Tabelle existiert. Falls dies zutrifft wird die Registrierungsseite angezeigt.

Ansprechpartner

Diese Tabelle (siehe Tabelle 5.4) ist eine Hilfstabelle um die Ansprechpartner mit den Prozessen zu verbinden. Dabei kann ein Ansprechpartner für mehrere Prozesse zuständig sein und ein Prozess mehrere Ansprechpartner haben.

Spalte	Feldtyp	Beschreibung
Gruppe	Integer	Fremdschlüssel der Gruppe.
Hash	Varchar(255)	Ein zufällig generierter in der Tabelle eindeutiger Hash-Wert (Primärer Schlüssel dieser Tabelle).

Tabelle 5.3: Registrierung

Spalte	Feldtyp	Beschreibung
Prozess_id	Integer	ID eines Prozesses (Fremdschlüssel).
Benutzer_id	Integer	ID eines Benutzers (Fremdschlüssel).

Tabelle 5.4: Ansprechpartner

Prozesse

In der Tabelle 5.5 werden die Allgemeininformationen “Titel“ und “Beschreibung“ einer Prozessdokumentation oder -kollektion gespeichert, zusätzlich wird unterschieden ob der Prozess frei gegeben wurde oder nicht. Die restlichen Einträge sind Referenzen auf andere Tabellen in denen die zusätzlichen Informationen des zugehörigen Prozesses gespeichert sind. Sollte ein Prozess in einer Prozesskollektion enthalten sein, so wird ihre ID entsprechend angegeben.

Domänen

In Tabelle 5.6 werden die Domänen gespeichert zu denen die einzelnen Prozessdokumentationen und Prozesskollektionen zugeordnet sind. Eine Domäne wird intern repräsentiert durch eine ID und einen für den Benutzer lesbaren Namen.

Spalte	Feldtyp	Beschreibung
Id	Integer	Primärer Schlüssel dieser Tabelle.
Kollektion_id	Integer	ID der Kollektion.
Benutzer_id	Integer	ID des Prozesserstellers, Fremdschlüssel der Benutzertabelle.
Sprache_id	Integer	Sprache in der der Prozess beschrieben ist, Fremdschlüssel zur Sprachentabelle.
Copyright_id	Integer	Urheberrecht dem der Prozess unterliegt, Fremdschlüssel zur Copyright Tabelle.
LeseRecht_id	Integer	Leseberechtigung für eine Gruppe, Fremdschlüssel zur Gruppentabelle.
Titel	Varchar(255)	Titel des Prozesses.
Beschreibung	Text	Beschreibung des Prozesses.
Bearbeitet	Boolean	Gibt an ob Prozess fertig bearbeitet ist.
Erstellt_am	Timestamp	Erstellungsdatum.
Geändert_am	Timestamp	Änderungsdatum.

Tabelle 5.5: Prozesse

Prozessdomänen

Die Tabelle 5.7 ist eine Hilfstabelle um den Prozessen eine oder mehrere Domänen zuordnen zu können. Eine Domäne kann mehrere Prozesse besitzen und ein Prozess kann zu mehreren Domänen gehören.

Dateien

In Tabelle 5.8 werden Informationen über die Dateien einer Prozessdokumentation gespeichert. Eine Datei besitzt einen Namen und einen MIME-Typ. Zusätzlich werden in dieser Tabelle die Referenzen auf die zugehörige Prozessdokumentation und Software gespeichert. Die eigentliche Datei wird in binärem Format in der Tabelle “Versionen“ archiviert.

Spalte	Feldtyp	Beschreibung
Id	Integer	Primärer Schlüssel dieser Tabelle.
Name	Varchar(255)	Name der Domäne.

Tabelle 5.6: Domänen

Spalte	Feldtyp	Beschreibung
Prozess_id	Integer	ID eines Prozesses (Fremdschlüssel).
Domäne_id	Integer	ID einer Domäne (Fremdschlüssel).

Tabelle 5.7: Prozessdomänen

Software

In Tabelle 5.9 werden die Namen der Programme gespeichert, die benötigt werden um die Dateien der Prozesse zu öffnen. Sie wird verwendet um Redundanzen in der Dateitabelle zu vermeiden.

Spalte	Feldtyp	Beschreibung
Id	Integer	Primärer Schlüssel dieser Tabelle.
Prozess_id	Integer	ID eines Prozesses (Fremdschlüssel).
Software_id	Integer	ID eines Softwareprogramms (Fremdschlüssel).
Name	Varchar(255)	Name der Datei.
Mime_type	Varchar(20)	MIME-Typ der Datei.
Erstellt_am	Timestamp	Erstellungsdatum.

Tabelle 5.8: Dateien

Spalte	Feldtyp	Beschreibung
Id	Integer	Primärer Schlüssel dieser Tabelle.
Name	Varchar(255)	Name der Software.

Tabelle 5.9: Software

Spalte	Feldtyp	Beschreibung
Id	Integer	Primärer Schlüssel dieser Tabelle.
Sprache	Varchar(255)	Sprache der Prozessdokumentation.

Tabelle 5.10: Sprachen

Sprachen

In Tabelle 5.10 werden die Namen der Sprachen gespeichert in denen die Prozesse beschrieben sind. Sie wird verwendet um Redundanzen in der Prozesstabelle zu vermeiden.

Copyright

In Tabelle 5.11 werden die Urheberrechtsinformationen der einzelnen Prozesse gespeichert. Sie wird verwendet um Redundanzen in der Prozesstabelle zu vermeiden.

Spalte	Feldtyp	Beschreibung
Id	Integer	Primärer Schlüssel dieser Tabelle.
Copyright	Varchar(255)	Urheberrecht des Prozesses.

Tabelle 5.11: Copyright

Spalte	Feldtyp	Beschreibung
Id	Integer	Primärer Schlüssel dieser Tabelle.
Dokument_id	Integer	ID einer Datei (Fremdschlüssel).
Data	Blob	Binäre Repräsentation der Datei.
Kommentar	Text	Kommentar für die Datei.
Aktuelle_Version	Boolean	Gibt ob die Version aktuell ist.
Hauptversion	Integer	Hauptversionsnummer.
Nebenversion	Integer	Nebenversionsnummer.
Erstellt_am	Timestamp	Erstellungsdatum.

Tabelle 5.12: Versionen

Versionen

In Tabelle 5.12 werden die Dateien der Prozessdokumentationen in Binärem Format gespeichert. Es wird zu jeder Version einer Datei ein Kommentar gespeichert und die Angabe ob es sich um die aktuellste Version handelt oder nicht. Die Versionsnummer wird aufgeteilt und in zwei separaten Spalten gespeichert. Dies vereinfacht die Logik der Versionierung und die Darstellung in der Benutzeroberfläche.

Zusatzinformationen

In Tabelle 5.13 werden die möglichen Zusatzinformationen der Prozesse gespeichert. Die Zusatzinformationen bestehen aus einem (Name, Wert)-Tupel, die die Funktionen eines Prozesses näher beschreiben. Ein Beispiel für eine Zusatzinformation wäre (Anzahl Arbeitsschritte, 15). In dieser Tabelle wird jedoch nur der Name der Zusatzinformation und der Datentyp des Wertes gespeichert. Der zugehörige Wert für einen bestimmten Prozess wird in der Tabelle “Werte“ hinterlegt.

Spalte	Feldtyp	Beschreibung
Id	Integer	Primärer Schlüssel dieser Tabelle
Name	Varchar(255)	Name der Zusatzinformation.
Datentyp	Varchar(10)	Datentyp der Zusatzinformation.

Tabelle 5.13: Zusatzinformationen

Werte

In Tabelle 5.14 werden die Wertdaten der Zusatzinformationen für die einzelnen Prozesse gespeichert. Diese Tabelle stellt eine (n:m)- Beziehung mit einem zusätzlichen Attribut zwischen der Zusatzinformations- und der Prozesstabelle dar.

Spalte	Feldtyp	Beschreibung
Id	Integer	Primärer Schlüssel dieser Tabelle
Prozess_id	Integer	Id eines Prozesses (Fremdschlüssel).
Zusatzinformation_id	Integer	Id der Zusatzinformation (Fremdschlüssel).
Wert	Varchar(255)	Wert der Zusatzinformation.

Tabelle 5.14: Werte

Eigenschaften

In Tabelle 5.15 werden die verschiedenen Eigenschaften, die die Prozesse besitzen können gespeichert. Die Eigenschaften werden durch ihren Namen beschrieben.

Spalte	Feldtyp	Beschreibung
Id	Integer	Primärer Schlüssel dieser Tabelle.
Eigenschaft	Varchar(255)	Name der Eigenschaft.

Tabelle 5.15: Eigenschaften

Prozess Eigenschaften

Tabelle 5.16 stellt eine (n:m)- Beziehung zwischen Eigenschaften und Prozessen dar. Sie ist eine Hilfstabelle um mehrere Eigenschaften mit mehreren Prozessen zu verbinden.

Spalte	Feldtyp	Beschreibung
Prozess_id	Integer	ID eines Prozesses (Fremdschlüssel).
Eigenschaft_id	Integer	ID einer Eigenschaft (Fremdschlüssel).

Tabelle 5.16: Prozess Eigenschaften

Zusätzliche Zugriffsrechte

Tabelle 5.17 stellt eine (n:m)- Beziehung zwischen der Benutzer- und Prozesstabelle dar. Sie speichert zusätzliche Zugriffsrechte für Prozesse, die dynamisch einzelnen Benutzern erteilt werden können.

Spalte	Feldtyp	Beschreibung
Id	Integer	Primärer Schlüssel dieser Tabelle.
Prozess_id	Integer	ID eines Prozesses (Fremdschlüssel).
Benutzer_id	Integer	ID eines Benutzers der Zugriffsrechte erhalten soll (Fremdschlüssel).
Recht	Boolean	Zusätzliches Zugriffsrechtes auf einen Prozess FALSE = Lesen; TRUE = Schreiben.

Tabelle 5.17: Zusätzliche Zugriffsrechte

5.2.3 Model Klassendiagramme

In diesem Abschnitt wird die objektrelationale Abbildung der Datenbanktabellen vorgestellt und auf ihre Beziehungen eingegangen. Im Klassendiagramm werden aus

Übersichtsgründen keine Methoden angegeben, es existieren jedoch `getter`- und `setter`- Methoden zu jedem Attribut.

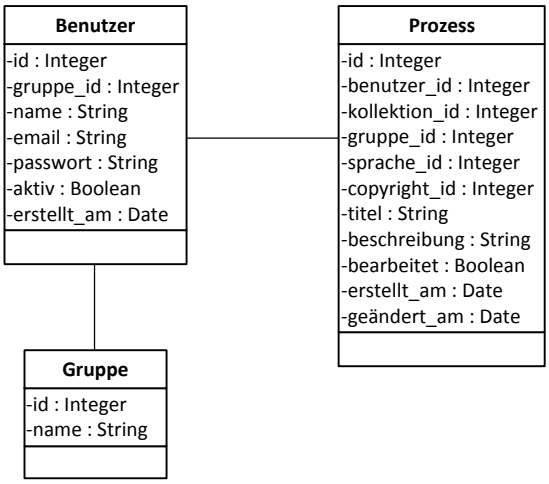

Abbildung 5.4: Einfache Assoziation zwischen Benutzer und Prozessen

In der Abbildung 5.4 wird die Klassenabhängigkeit zwischen Benutzer, Prozess und Gruppe beschrieben. Es besteht eine einfache Assoziation zwischen diesen Klassen, da sie nicht von einander abhängig sind.

Abbildung 5.5 zeigt weitere einfache Assoziationen der Klasse Prozesse. Eigenschaften, Zusatzinformationen, Sprache, Copyright und Domänen stehen in Beziehung zu einander, ihre Existenz ist jedoch nicht von einander abhängig.

In dem Diagramm in Abbildung 5.6 werden die Abhängigkeiten die im System existieren dargestellt.

Ein Prozess als ganzes beinhaltet Dateien, Eigenschaften und Zusatzinformationen. Abhängig von der Prozessklasse ist die Datei- und Wertklasse. Abhängig von der Dateiklasse ist die Klasse der Versionen. Sollte eine Datei gelöscht werden, so werden

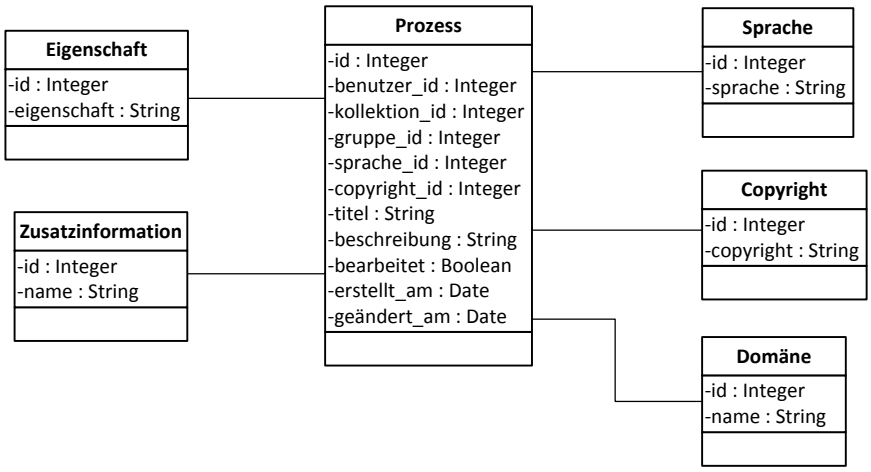

Abbildung 5.5: Weitere Assoziationen der Prozessklasse

auch alle Versionen dieser Datei gelöscht. Sollte eine Prozessdokumentation gelöscht werden, so werden alle zugehörigen Dateien und Werte gelöscht.

5.2.4 Implementierung des Models

Dieser Abschnitt geht auf die Implementierung des Models ein. Die Umsetzung wird anhand von Programm-Code aufgezeigt und erläutert.

Das Model wird in Ruby on Rails mit Hilfe des *ActiveRecord* Moduls umgesetzt [MO12]. Jede Model Klasse erbt von diesem Modul und bekommt den Namen der zugehörigen Datenbanktabelle im Singular. Das Mapping zwischen der Datenbanktabelle und der Model Klasse regelt Rails daraufhin selbst. Die Attribute der Datenbanktabelle müssen in den Model Klassen nicht direkt angegeben werden, da sie von Rails aus der Definition der Tabelle abgeleitet werden können. Jegliche Änderungsoperationen auf den Attributen werden in der Datenbank ausgeführt und die Änderung in dem jeweiligen Klassenobjekt unmittelbar aktualisiert. Die Methoden zum speichern, löschen oder ändern eines Objektes müssen dabei ebenfalls nicht selbst implementiert werden da sie von *ActiveRecord* bereitgestellt werden.

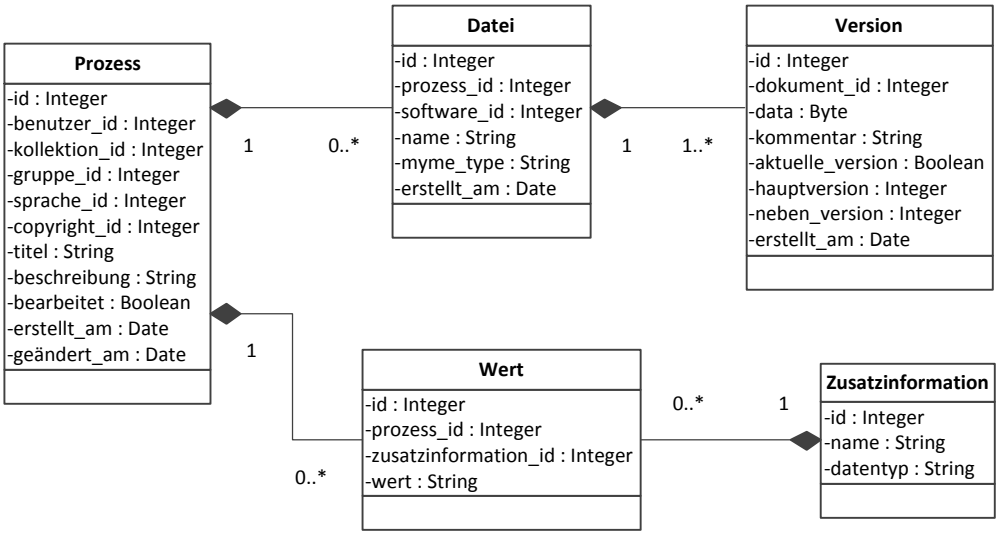

Abbildung 5.6: Abhängigkeitsbeziehungen der Prozessklasse

Da viele Methoden und Konventionen in Rails an die englische Sprache angelehnt sind, wird der Programm-Code sowie die Datenbanktabellen bei der Implementierung in die englische Sprache übersetzt.
Der Programm-Code aus Abbildung 5.7 zeigt beispielhaft die Implementierung der *Sprachen* Tabelle in der Datei `Language.rb`.

In der ersten Zeile von Abbildung 5.7 findet die zuvor erwähnte Vererbung statt. Die zweite Zeile definiert zwei Validierungen auf der Tabellenspalte `language`, die besagen das der Wert in dem Attribute `language` nicht leer und in der Tabelle eindeutig sein muss. Trifft eine dieser Bedingungen nicht zu so kann das Attribute nicht gespeichert werden und jeder Versuch das Objekt in der Datenbank zu speichern schlägt fehl. In der dritten Zeile wird mit Hilfe der `has_many`-Methode eine (1:n)- Beziehung zwischen der Sprach- und der Prozesstabelle definiert. Hierbei wird in der `Language` Klasse die `has_many`-Methode angegeben und in der `Processmodels` Klasse (welche die Klasse der Prozesstabelle darstellt) die `has_one`-Methode, da viele Prozesse zu einer Sprache gehören können.

Das Schlüsselwort “private“ in Zeile fünf besagt, dass alle Methoden die unterhalb dieses Schlüsselwortes stehen als Privat gelten und nur in der eigenen Klasse sichtbar sind. Die Methode in Zeile neun definiert schließlich welche Tabellenspalte bei einer Suche mit dem Plugin “Ransack“ durchsucht werden dürfen.

```
1  class Language < ActiveRecord::Base
2    validates :language, presence: true, uniqueness: true
3    has_many :processmodels
4
5    private
6
7    # Definiert welche Tabellenspalten durchsuchbar sein sollen.
8    # Hier ist die Spalte Language durchsuchbar
9    def self.ransackable_attributes(auth_object = nil)
10     %w(language) + _ransackers.keys
11   end
12 end
```

Abbildung 5.7: Model Klasse für die Sprachen Tabelle

Die Implementierung der anderen Model Klassen erfolgt analog und wird daher hier nicht nochmals wiedergegeben. Der interessierte Leser sei auf den angefügten Programm-Code der Implementierung verwiesen.

5.3 View

Im diesem Abschnitt werden einige im Rahmen der Arbeit für das System entwickelte Mockups präsentiert und auf ihre Besonderheiten eingegangen. Zusätzlich wird die Implementierung anhand von Beispielen verdeutlicht. Die hier vorgestellten Mockups dienen anschließend als Grundlage für die Implementierung der graphischen Oberfläche.

5.3.1 Mockups

In der View werden die vom Controller gelieferten Daten dargestellt und dem Benutzer zur Verfügung gestellt. Die View stellt die Schnittstelle zwischen Benutzer und System dar. Deshalb sollte sie intuitiv zu bedienen und leicht in der Handhabung sein. Im folgenden werden Mockups des Prozess Repository vorgestellt.

Startseite

Auf der Startseite (siehe Abbildung 5.8) wird eine Suchmaske für Prozessdokumentationen (ohne jegliche Leseeinschränkungen) angezeigt. Ferner existiert ein Link um auf die Seite der erweiterten Suche zu gelangen. Um Benutzern eine kleine Übersicht über den Inhalt und die Größe des Repositories zu geben wird außerdem eine Statistik über den Inhalt des Systems angezeigt. Ist ein Benutzer nicht angemeldet so ist der Login Bereich von jeder Seite aus zu erreichen.

Angemeldeter Benutzer

Meldet sich der Benutzer am System an erhält er eine spezielle Sicht (siehe Abbildung 5.9) in der ihm bis zu vier verschiedene Listen angezeigt werden. Hat der Benutzer das Recht zum Anlegen neuer Prozessdokumentationen, so werden in der ersten Liste die Prozessdokumentationen angezeigt, die noch nicht vollständig bearbeitet wurden. Solche Prozessdokumentationen sind insbesondere noch nicht durch die Suche auffindbar. Die zweite und dritte Liste beinhaltet alle Prozessdokumentationen für die der angemeldete Benutzer Lese- und Schreibrechte besitzt. Falls ein Benutzer selber bereits Prozesse im System angelegt hat, so wird zusätzlich eine Liste seiner eigenen Prozesse angezeigt, über die diese verwaltet werden können. Auch eine Suchmaske wird angeboten. Insbesondere können hierüber angemeldete Benutzer je nach Benutzerrolle auf Prozessdokumentationen mit Lesebeschränkung zugreifen können.

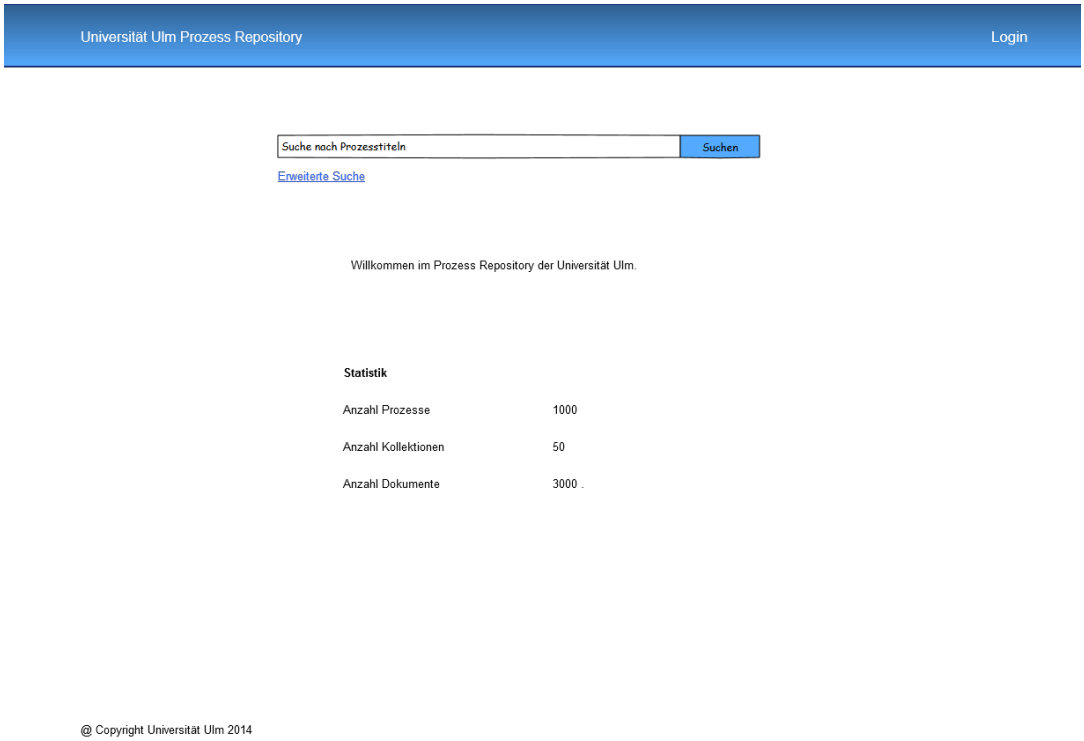

Abbildung 5.8: Startseite des Prozess Repositories

Suche

Die Suche ist das Kernstück des System. Durch sie können im System vorhandene Prozesse effizient gesucht und angezeigt werden. In der normalen Suchmaske (siehe Abbildung 5.10) wird dabei ausschließlich nach Prozesstiteln gesucht. In der ”Erweiterten Suche” kann darüber hinaus auch nach Prozessdokumentationen, deren Prozesse bestimmte Eigenschaft besitzen, gesucht werden. Je nachdem welcher Benutzerrolle der Benutzer zugeordnet ist, werden ihm nur Prozessdokumentationen angezeigt für die er eine Leseberechtigung besitzt. Die Suche nach Prozessdokumentationen wird als eine Ajax-basierte Livesuche implementiert.

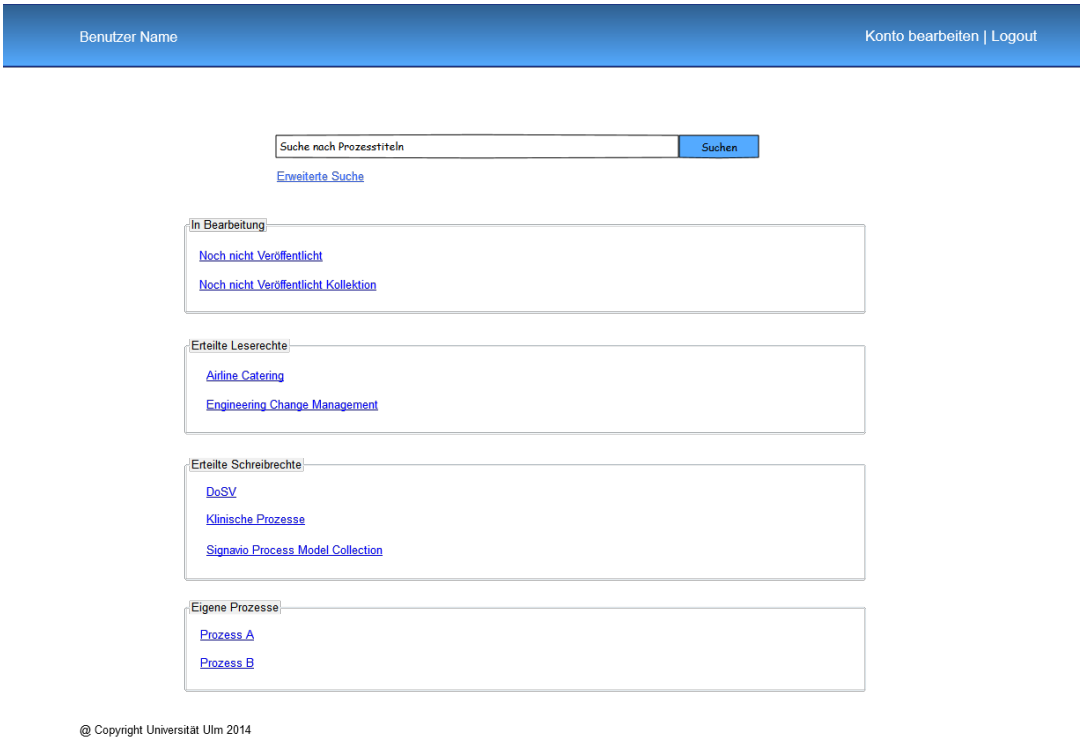

Abbildung 5.9: Sicht eines angemeldeten Benutzers

Prozessdokumentationen bearbeiten

Das Bearbeiten der Metainformationen, sowie das hinzufügen und löschen von Dateien ist integraler Bestandteil des Systems. Bearbeitungsfenster wie in Abbildung 5.11 werden für Metainformationen, Dateien der Prozessdokumentationen und Benutzerdaten benötigt.

5.3.2 Implementierung der Views

Die Views werden in Ruby on Rails mit dem Modul *ActionView* umgesetzt [MO12]. *ActionView* ist für die Generierung der Antwort auf eine Anfrage eines Clients zuständig, die anschließend an diesen zurück gesendet wird. Zur Anwendung kommen dabei sogenannte Template Dateien mit der Endung `.erb`. ERB steht hierbei für “Embedded Ruby“

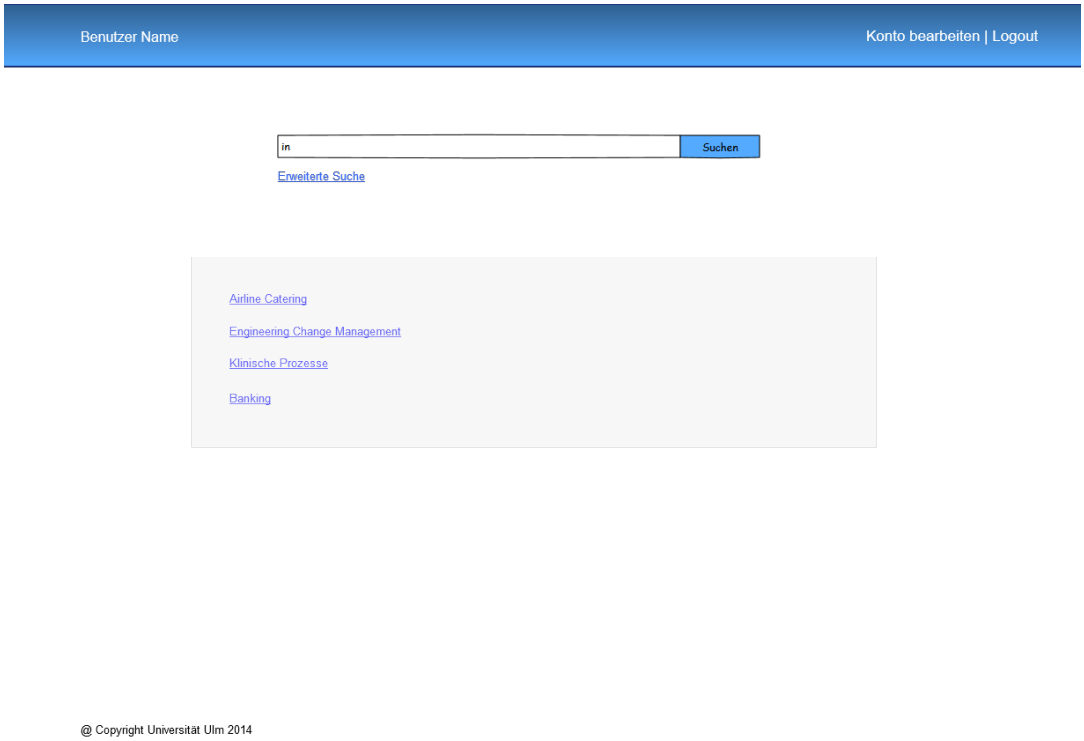

Abbildung 5.10: Suche nach Prozessen mittels ihrer Titel

und bedeutet das Ruby Code in die, meist aus HTML-Code bestehende, Template Datei eingebettet wird. Um die daraus resultierende Vermischung von HTML und Ruby Code zu reduzieren bietet Rails Hilfs- Methoden für die Generierung von HTML Elementen an. Um Redundanzen beim programmieren zu vermeiden, bietet Rails den Entwicklern drei Arten von Template-Dateien an. Die erste Art sind sogenannte Layout Dateien. Diese können dafür verwendet werden das Webseitendesign zu erstellen und stellen damit das Gerüst für die eigentliche Webseite dar. Im Body Bereich der HTML-Seite kann Rails, an vom Entwickler definierten Plätzen, weitere Templates einbinden solange das Ergebnis ein valides HTML-Dokument ergibt. In Abbildung 5.12 wird eine vereinfachte Variante der `application.html.erb` angezeigt, die das Webseitendesign für Anonyme Benutzer des Prozess Repository enthält. An die Stelle des Schlüsselwortes `yield` werden beim rendern die Templates zweiter Art eingesetzt.

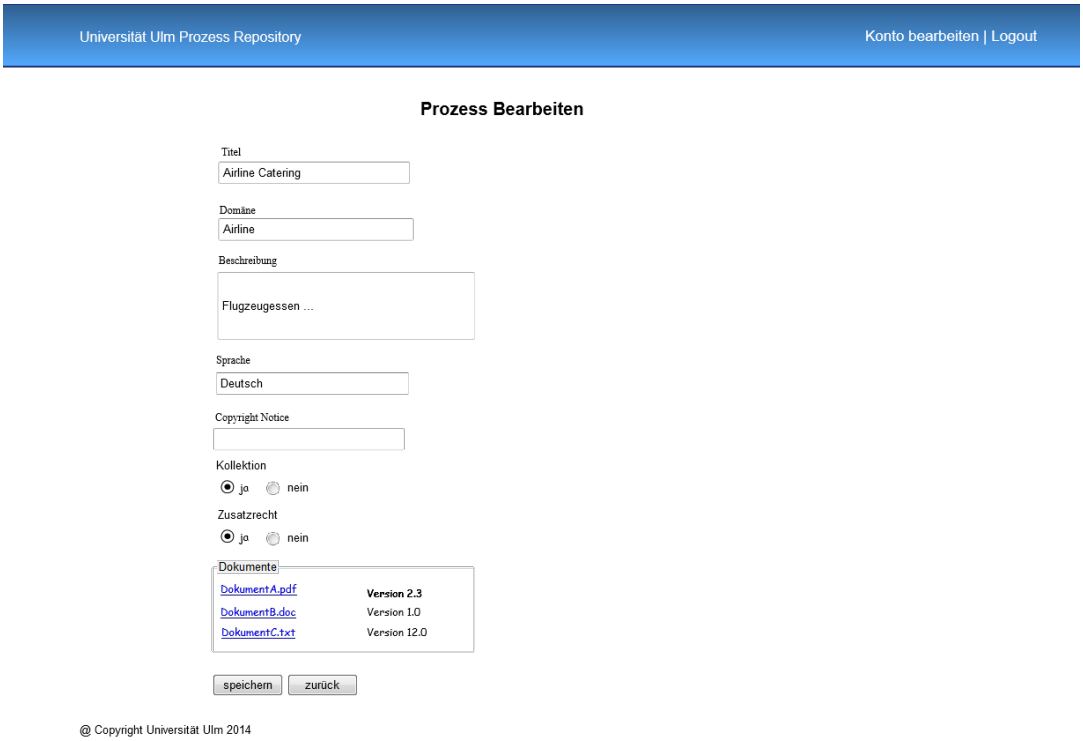

Abbildung 5.11: Bearbeitung eines Prozesses

Die zweite Art von Templates sind Dateien welche die vom Controller übergebenen Daten rendern. Diese Templates werden in die Layout Dateien eingebettet und ergeben die fertige Webseite. Abbildung 5.13 zeigt die Implementierung eines solchen Templates anhand der Startseite des Administrator-Bereichs. Hierbei werden dem Template vom Controller zwei Instanzvariablen übergeben, nämlich `@processmodels` und `@users`, die jeweils ein Array mit fünf Objekten der jeweiligen Klassen enthalten. Innerhalb der HTML Elemente `fieldset` werden diese beiden Arrays durchlaufen und in jedem Schleifendurchgang die Hilfs-Methode `link_to` aufgerufen. Dieser werden zwei Parameter übergeben, der erste ist der Name des Links der erzeugt werden soll und der zweite das aktuelle Objekt zu welchem der Link führen soll. In Abbildung 5.13 werden dazu als ersten Parameter die Namen der Prozesse beziehungsweise der Benutzer übergeben.

```
<html class="no-js" lang="de">
  <head>...</head>
  <body>
    ...
    <div class="row">
      <%= yield %>
    </div>
    ...
  </body>
</html>
```

Abbildung 5.12: Layout Template

```
# Auflistung der Neueste 5 Prozesse im System
<fieldset>
  <legend>Neueste 5 Prozesse</legend>
  <% @processmodels.each do |pro| %>
    <%= link_to pro.title, pro %><br />
  <% end %>
</fieldset>
# Auflistung der Neueste 5 Benutzer im System
<fieldset>
  <legend>Neueste 5 Benutzer</legend>
  <% @users.each do |user| %>
    <%= link_to user.name, user %><br />
  <% end %>
</fieldset>
```

Abbildung 5.13: Template der Startseite des Administrator Bereiches

Die dritte Art von Templates sind sogenannte “Partials“. Mit Partials bietet Rails die Möglichkeit öfter verwendeten HTML- oder Ruby Code auszulagern und den Rendering Prozess in kleinere Stücke zu zerteilen. Vor allem bei größeren Dateien sind Partials sehr nützlich, da sie die Übersicht erhöhen und komplexe Bereiche ausgelagert werden können. Abbildung 5.14 zeigt ein solches (vereinfachtes) Partial in das die Ausgabe der, von einer Suche gefundenen, Prozesse ausgelagert wurde. Wie bereits in Abbildung 5.13 wird auch in Abblidung 5.14 in einer Schleife die Hilfs-Methode `link_to` aufgerufen. Zusätzlich wird mit der Hilfs-Methode `content_tag` ein Tabellen Kopf Bereich erstellt in das ein Link zum sortieren der aufgelisteten Prozesse eingebettet wird.

```
<% if !process.nil? %>
  <table class="responsive">
    <%= content_tag :th, sort_link(@q, :title, "Prozess") %>
    <% process.each do |p| %>
      <tr>
        <td>
          <%= link_to p.title, p %>
        </td>
      </tr>
    <% end %>
  </table>
<% else %>
Suche war erfolglos.
<% end %>
```

Abbildung 5.14: Partial: Anzeige aller gefundener Prozesse

5.4 Controller

Dieser Abschnitt beschreibt zunächst den allgemeinen Aufbau und die Funktionsweise von Controllern in einer Ruby on Rails Applikation. Es stellt den REST Architektur-Stil vor und geht anschließend auf die Funktion ausgewählter Controller im Prozess Repository ein.

5.4.1 Funktionsweise

Wie zuvor erwähnt liegt im Controller die Programmlogik einer Applikation. Er steuert das Verhalten des Systems bei einer Interaktion eines Benutzers. Bei einer Rails Applikation hat der Controller folgende Aufgaben zu erfüllen:

- Datenbankabfragen über Model Objekte ausführen
- Daten aus einer HTTP Anfrage auslesen
- Cookies lesen und setzen
- Daten aus Sessions lesen und setzen
- Senden von Daten und Dateien an den Client

- Aufrufen der Views
- Benutzer Authentifizierung
- Weiterleitungen
- Setzen von Flash Nachrichten

Flash Nachrichten sind hierbei normale Strings die mit der Antwort des Server mitgeschickt werden können um sie den Benutzern des Systems anzuzeigen. Sie werden häufig für Rückmeldungen verwendet um Benutzer auf Fehler oder Erfolg seiner Aktion hinzuweisen.

In Rails Applikationen werden Controller durch Klassen repräsentiert die alle von der Klasse ApplicationController erben. Nach der Grundidee soll ein Controller jeweils für eine Datenbanktabelle zuständig sein und Methoden für das Auslesen und Manipulieren der Daten dieser Tabelle anbieten. Es können jedoch weitere Methoden hinzugefügt werden um Programm-Code auszulagern, Funktionen auszuführen oder sogar auf andere Datenbanktabellen manipulierend zuzugreifen. Der Zugriff auf die Methoden geschieht durch eine RESTful API, die im folgenden Abschnitt erläutert wird.

RESTful API

REST steht für “Representational State Transfer“ [Fie00]. Es ist ein Architektur-Stil für Web Anwendungen und beschreibt wie auf Ressourcen zugegriffen werden kann. Ressourcen werden dabei eindeutig über eine URI repräsentiert die mit verschiedenen Daten oder Dateien assoziiert werden können. Diese Daten können wiederum verschiedene Repräsentationsformen besitzen, wie beispielsweise XML und JSON.

Für den Zugriff auf die Methoden in den Controller Klassen verwendet REST das HTTP-Protokoll und die HTTP-Methoden GET, PUT, POST und DELETE. Dadurch ist man in der Lage über eine einzige URL verschieden Operationen auf einer Ressource auszuführen. Die HTTP-Methoden haben dabei folgende Bedeutung.

Methode	Beschreibung
GET	Ressource auslesen und zuschicken
PUT	Ressource ändern oder neue anlegen
POST	Ressource ändern
DELETE	Ressource löschen

Tabelle 5.18: HTTP-Methoden

In Rails 4.0 wird die HTTP-Methode PUT durch PATCH ersetzt und wird dazu verwendet Ressourcen zu ändern anstatt sie anzulegen.

Methoden

Eine Controller Klasse die nur Methoden bereit stellt die für eine bestimmte Datenbanktabelle zuständig sind, enthält die Methoden *index*, *show*, *new*, *create*, *edit*, *update* und *destroy*. Die Methoden *index*, *show*, *new* und *edit* rufen nach ihrer Abarbeitung die gleichnamigen View Komponenten auf in denen die jeweiligen Daten dargestellt werden. Die daraufhin erzeugte HTML-Seite wird in die Layout HTML-Datei eingebettet und das Ergebnis an den Client geschickt. Die restlichen Methoden *create*, *update* und *destroy* leiten nach ihrer Abarbeitung den Request an eine weitere Methode oder HTML Seite weiter.

Im folgenden wird die Funktion der Methoden *index*, *show*, *new*, *create* und *edit* anhand der Datenbanktabelle `users` erklärt und mit Beispielen verdeutlicht.

index-Methode

In dieser Methode werden alle Daten aus der Datenbank geholt die zu einer bestimmten Ressource gehören und anschließend der `index.html.erb` Datei übergeben. Im Beispiel (siehe Abbildung 5.15) werden die Daten aller Benutzer aus der Datenbank geholt, indem die Methode *all* aus der Model Klasse `User` aufgerufen wird. All diese Benutzerdaten werden in der Instanzvariablen `@users` hinterlegt auf die in der `index.html.erb` zugegriffen werden kann.

```ruby
def index
  @users = User.all
end
```

Abbildung 5.15: Index Methode

```ruby
def show
  @user = User.find(params[:id])
end
```

Abbildung 5.16: Show Methode

show-Methode

Die Methode *show* wird verwendet um eine bestimmte Ressource anzuzeigen. Im Beispiel (Abbildung 5.16) wird dazu als erstes die in der URL angegebene User ID aus dem `params`-Array ausgelesen und der Methode *find* als Parameter übergeben. Diese Methode holt sich den Datensatz mit der entsprechenden ID und hinterlegt diese in der Instanzvariablen `@user`. Dargestellt wird der Datensatz anschließend durch die `show.html.erb` Datei.

new-Methode

In der Methode, die in Abbildung 5.17 dargestellt ist, wird das Anlegen neuer Ressourcen vorbereitet, die eigentliche Erstellung der Ressourcen geschieht in der Methode *create*. Im Beispiel wird in der Methode eine neue Instanz der Klasse `User` erstellt mit deren Hilfe in der View ein Formular aufgebaut wird. Dieses Formular wird standardmäßig in ein sogenanntes Partial ausgelagert, da es die Datei `edit.html.erb` ebenfalls verwendet.

```ruby
def new
  @user = User.new
end
```

Abbildung 5.17: New Methode

```
1  def create
2    @user = User.create(user_params)
3    redirect_to @user, notice: "Benutzer erfolgreich angelegt!"
4  end
5
6  private
7  # Parameter fuer Massenzuweisung
8  def user_params
9      params.require(:user).permit(:name, :email, :password)
10 end
```

Abbildung 5.18: Create Methode

create-Methode

Diese Methode wird zum anlegen neuer Ressourcen verwendet. In dem Beispiel in Abbildung 5.18 wird die *create*-Methode der User Klasse aufgerufen um einen neuen Benutzer in der Datenbank zu speichern. Die *user_params*-Methode filtert hierbei die Daten aus dem Request, die für den Speichervorgang benötigt werden. Somit können keine unerlaubten Zuweisungen entstehen. Nach der Speicherung findet eine Weiterleitung an die *show*-Methode statt und es wird eine Flash Nachricht gesetzt, um die erfolgte Speicherung zu bestätigen.

edit-Methode

Sind Ressourcen veränderbar, so wird die Veränderung in dieser Methode (siehe Abbildung 5.19) vorbereitet. Im Beispiel wird eine User Ressource geladen und in der Variablen `@user` hinterlegt. Das bereits von der `new.html.erb`-Datei verwendete Formular-Partial wird auch hier wieder verwendet und liest die Daten aus der Instanzvariablen `@user` aus und setzt sie in die entsprechenden Textfelder ein, um die Daten editierbar zu machen. Rails erkennt dabei ob es sich bei der Instanzvariablen `@user` um eine neue Instanz der Klasse `User` handelt oder ob diese einen bestehenden Datensatz repräsentiert und setzt die nötigen Daten und Pfadangaben automatisch. Das Partial wird anschließend in die Datei `edit.html.erb` eingebettet.

```ruby
def edit
  @user = User.find(params[:id])
end
```

Abbildung 5.19: Edit Methode

```ruby
get "users" => "user#index", as: "user_list"
get "user/:id" => "user#show", as: "user"
# Bespiel URL: example.com/user/42/new
get "user/:id/new" => "user#new", as: "new_user"
get "user/:id/edit" => "user#edit", as: "edit_user"
patch "user/:id" => "user#update", as: "update_user"
post "user/:id" => "user#create", as: "create_user"
delete "user/:id" => "user#destroy", as: "delete_user"

# Selber Effekt mit der Angabe
resources :users
```

Abbildung 5.20: Root.rb Datei

Methodenaufruf

Das Aufrufen der Methoden in einem Controller geschieht über die URL einer HTTP Anfrage. Das Standardschema einer solchen URL ist folgendermaßen aufgebaut: `example.com/Controllername/Methodenname/Parameter`. Durch den Action Dispatcher können diese URL‘s beliebige Formen annehmen. Dies muss jedoch in der `roots.rb`-Datei angegeben werden [MO12]. Die Angabe solcher URL‘s in der `roots.rb`-Datei in rails 4.0 wird in Abbildung 5.20 veranschaulicht.

Durch Angabe des Schlüsselwortes “resources“ generiert Rails die Standard Routen für eine Ressource.

5.4.2 Ausgewählte Controller Klassen

In diesem Abschnitt wird auf einige ausgewählte Controller Klassen des Prozess Repositories eingegangen und ihre Funktionen erklärt.

ApplicationController

Der `ApplicationController` ist der Haupt Controller in Ruby On Rails. Von ihm erben alle anderen Controller in einem Rails Projekt. Im `ApplicationController` können, ganz im Sinne des `don't repeat yourself` Prinzips, Methoden implementiert werden die allen anderen Controllern zur Verfügung stehen sollen. Dieser Controller wird bei der Generierung eines neuen Projektes automatisch erstellt.

ProcessController

Der `ProcessController` ist für die Änderungsoperationen auf der Prozess Tabelle der Datenbank zuständig. Da die Prozesstabelle das Zentrum der Datenbankarchitektur des Repositories darstellt, wird aus diesem Controller auch auf Daten anderer Datenbanktabellen schreibend und lesend zugegriffen.

DocumentsController

Der `DocumentsController` ist für die Änderungsoperationen auf der Datei Tabelle, in der die Informationen einer Datei gespeichert sind, zuständig. Er ist ebenfalls dafür zuständig eine Datei an den Client zu schicken wenn diese angefragt wird.

SearchController

In diesem Controller sind alle Methoden enthalten die für die Ajax basierte Livesuche erforderlich sind. Alle Suchanfragen die per Ajax an den Server gesendet werden, werden in diesem Controller verarbeitet und die Ergebnisse in Form von JSON-Daten an den Client zurück geschickt. Andere Ajax Funktionalitäten die server-seitige Funktionen aufrufen sind in den jeweiligen Controllern implementiert.

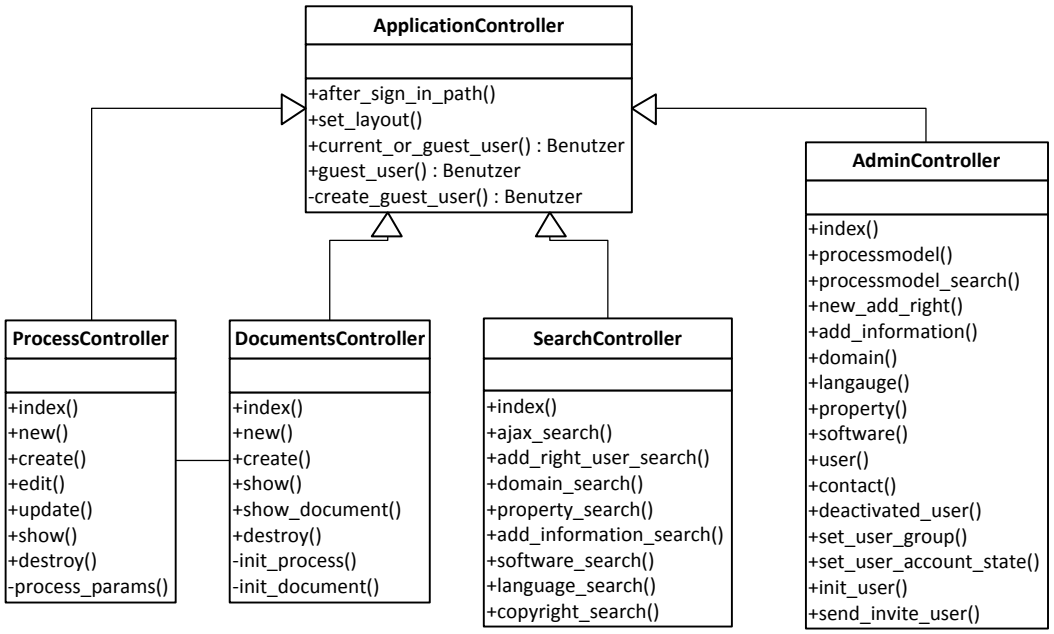

Abbildung 5.21: Klassendiagramm der Controller Klassen

5.4.3 Controller Klassendiagramm

Das Klassendiagramm in Abbildung 5.21 zeigt die von Rails vorgesehene Klassenhierarchie anhand ausgewählter Controller Klassen aus dem Prozess Repository. Ebenfalls werden die Methoden angezeigt die in den jeweiligen Controller implementiert sind.

Aus Übersichtsgründen werden dabei beim `AdminController` nicht alle implementierten Methoden aufgelistet.

5.4.4 Implementierung der Controller

In diesem Abschnitt wird die Implementierung zweier ausgewählter Methoden aus dem `ProcessController` vorgestellt und erläutert.

Abbildung 5.22 zeigt die *show*-Methode des `ProcessController`, welche eingesetzt wird um die Daten eines Objektes anzuzeigen. Hinter der Instanzvariablen `@processmodel`

befindet sich ein Objekt welches die Prozessdaten aus der aktuellen HTTP Anfrage enthält. In Zeile vier werden die Daten aller Benutzer aus der Datenbank geholt welche zu diesem Prozess gehören. In diesem Fall werden in der `@users`-Variable alle Ansprechpartner zu einem bestimmten Prozess hinterlegt. In der `@child_processmodels`-Variable werden alle Subprozesse hinterlegt, falls der aktuelle Prozess eine Prozesskollektion ist. In die Variablen `@domains` werden alle Domänen, bei `@properties` alle Eigenschaften und bei `@documents` die Informationen aller Dateien die zum aktuellen Prozess gehören, abgelegt. In `@language` und `@copyright` werden die zu dem Prozess gehörige Sprache, sowie Information über das Urheberrecht, (falls eines in der Datenbank gespeichert wurde) hinterlegt. Ist der aktuelle Prozess ein Subprozess einer Kollektion so wird das Objekt der zugehörigen Prozesskollektion in der Variable `@collection_process` hinterlegt. Auf alle in der *show*-Methode deklarierten Instanzvariablen kann daraufhin in der `show.html.erb`-Datei Bezug genommen werden, um die gewünschten Informationen darzustellen.

In Abbildung 5.23 wird die *destroy*-Methode des `ProcessController` dargestellt. Sie wird verwendet um eine Ressource, in diesem Falle einen Prozess, mit allen assoziierten Daten aus der Datenbank zu entfernen. Das zu löschende Objekt wird dazu anhand seiner ID, welche aus der HTTP-Anfrage entnommen wird, aus der Datenbank geholt. In Zeile vier wird auf diesem neu erstellten `ActiveRecord`-Objekt die *destroy*-Methode aufgerufen. Diese Methode liest aus der Deklaration der Model Klasse heraus, welche Datenbank Tabellen und Spalten mit diesem Objekt über Beziehungen verbunden sind und löscht daraufhin die Daten des Objektes und anschließend alle Beteiligten Objekte kaskadierend. In Zeile sechs bis acht wird anschließend das Format gesetzt aus dem die Antwort besteht. In diesem Fall wird `format.js` angegeben. Das bedeutet das passend zur *destroy*-Methode die Datei `destroy.js.erb` aufgerufen wird. Dies ist eine Javascript Datei in der Ruby Code eingebettet werden kann. Nach dem rendern wird der produzierte Javascript Code auf dem Client ausgeführt. Dies führt dazu, dass beim Client die Zeile in welcher der gelöschte Prozess angezeigt wurde, versteckt wird.

```
1  # Initialisiert die View mit der ein Prozess, mit all seinen
2  # Informationen, angezeigt werden kann
3  def show
4    @users = @processmodel.users
5    @child_processmodels = Processmodel
6    .where("collection_id = ?
7    AND finished = true","#{@processmodel.id}")
8    @domains = @processmodel.domains
9    @information = @processmodel.add_informations
10   @properties = @processmodel.properties
11   @documents = @processmodel.documents
12   @language = Language.find(@processmodel.language_id)
13   if !@processmodel.copyright_id.nil?
14     @copyright = Copyright.find(@processmodel.copyright_id)
15   end
16   if !@processmodel.collection_id.nil?
17     @collection_process = Processmodel.
18     find(@processmodel.collection_id)
19   end
20 end
```

Abbildung 5.22: Show-Methode des `ProcessController`

```
1 # Loescht einen Prozess und rekursiv alle
2 # Unterprozesse mit ihren Daten
3 def destroy
4   @processmodel = Processmodel.find(params[:id])
5   @processmodel.destroy
6   respond_to do |format|
7     format.js
8   end
9 end
```

Abbildung 5.23: Destroy-Methode des `ProcessController`

6

Zusammenfassung

In dieser Bachelorarbeit wurde ein web-basierendes Repository zur Verwaltung von Prozessdokumentationen und Prozesskollektionen konzipiert und entwickelt. Für diesen Zweck wurde in Kapitel 3 eine Anforderungsanalyse durchgeführt, um die Funktionalen und die Nicht-Funktionalen Anforderungen an dieses Repository zu ermitteln. Anschließend wurden in Kapitel 4 geeignete Technologien ausgewählt, mit denen das Repository implementiert wurde. Außer den Standard Web Technologien, wie zum Beispiel HTML, CSS und Javascript, wurde Ruby mit dem Ruby on Rails Framework als server-seitige Programmiersprache eingesetzt. In Kapitel 5 wurde daraufhin auf die Implementierung des Prozess Repository eingegangen. Es wurde die zu Grunde liegende Systemarchitektur erläutert, sowie ein Datenmodel konzipiert, welches textuell beschrieben und mit Hilfe von Entity-Relationship-Diagrammen veranschaulicht wurde. Zusätzlich wurde auf die Funktionen der Model, View,- sowie Controller Komponenten eingegangen.
Mit der Implementierung des Prozess Repository wurde ein Verwaltungssystem für Pro-

zessdokumentationen und Prozesskollektionen entwickelt. Das Problem der dezentralen Archivierung der Prozessdokumentationen kann durch Verwendung dieses Repositories gelöst werden. Große Prozesskollektionen die wiederum Prozesskollektionen beinhalten, können sehr komplexe Hierarchie Strukturen bilden, in denen der Überblick über die enthaltenen Prozessdokumentationen, sowie ihrer Dateien verloren gehen kann. Das Repository ist in der Lage beliebig komplexe Strukturen in der Hierarchie der Prozesskollektionen abzubilden und ihre Prozessdokumentationen übersichtlich darzustellen. Die Beschreibungsinformationen können effizient verwaltet und zugehörigen Prozess Dateien, samt all ihren Versionen, übersichtlich angezeigt werden. Die Suchmechanismen des Repositories sind darauf optimiert die Suche nach Prozessdokumentationen dem Benutzer so angenehm wie möglich zu gestalten. Für diese Aufgabe bietet das Repository eine Titel basierte Live-Suche an und eine erweitere Suche mit schrittweiser Verfeinerung der Ergebnismenge.

Abbildungsverzeichnis

Literaturverzeichnis

[Che76] CHEN, Peter P.: The entity-relationship model—toward a unified view of data. In: *ACM Transactions on Database Systems (TODS)* 1 (1976), Nr. 1, S. 9–36

[cod] *CodeIgniter Website*. https://ellislab.com/codeigniter. – Eingesehen am 20.08.2014

[cof] *Coffeescript Website*. http://coffeescript.org/. – Eingesehen am 20.08.2014

[Fie00] FIELDING, Roy T.: *Architectural styles and the design of network-based software architectures (REST)*, University of California, Irvine, Diss., 2000. – 76–105 S.

[G+05] GARRETT, Jesse J. u. a.: Ajax: A new approach to web applications. (2005). https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf

[gwt] *Google Web Toolkit Website*. http://www.gwtproject.org/overview.html. – Eingesehen am 20.08.2014

[jqu] *Jquery Website*. https://http://jquery.com/. – Eingesehen am 20.08.2014

[LRRVDA+11] LA ROSA, Marcello ; REIJERS, Hajo A. ; VAN DER AALST, Wil M. ; DIJKMAN, Remco M. ; MENDLING, Jan ; DUMAS, Marlon ; GARCÍA-BAÑUELOS, Luciano: APROMORE: An advanced process model repository. In: *Expert Systems with Applications* 38 (2011), Nr. 6, S. 7029–7040

[MO12] MORSY, Hussein ; OTTO, Tanja: *Ruby on Rails 3.1*. Galileo Press, 2012

[MWA+07] MA, Zhilei ; WETZSTEIN, Branimir ; ANICIC, Darko ; HEYMANS, Stijn ; LEYMANN, Frank u. a.: Semantic Business Process Repository. In: *International Workshop on Semantic Business Process Management (SBPM '07)* Vol. 251 CEUR Workshop Proceedings, (2007)

[nod] *Node.js Website*. `http://nodejs.org/`. – Eingesehen am 20.08.2014

[pos] *PostgrSQL Datenbank Website*. `http://www.postgresql.org/about/`. – Eingesehen am 20.08.2014

[rai] *Ruby on Rails Website*. `http://rubyonrails.org/`. – Eingesehen am 20.08.2014

[rub] *Ruby Website*. `https://www.ruby-lang.org/de/`. – Eingesehen am 20.08.2014

[sas] *SASS Webdesign Website*. `http://sass-lang.com/`. – Eingesehen am 20.08.2014

[v8e] *Google Developers, V8 engine Website*. `https://developers.google.com/v8/intro`. – Eingesehen am 10.10.2014

[YDG12] YAN, Zhiqiang ; DIJKMAN, Remco ; GREFEN, Paul: Business process model repositories–Framework and survey. In: *Information and Software Technology* 54 (2012), Nr. 4, S. 380–395

[Zil12] ZILLGENS, Christoph: *Responsive Webdesign: reaktionsfähige Websites gestalten und umsetzen*. Carl Hanser Verlag GmbH Co KG, 2012

[zur] *Zurb Foundation 5 Website*. `http://foundation.zurb.com/`. – Eingesehen am 20.08.2014

Name: Albert Bub Matrikelnummer: 752111

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Albert Bub