Entwurfsbeschreibung Vorprojekt

Inhalt

1. Allgemeines 1
2. Produktübersicht 1
3. Grundsätzliche Struktur- und Entwurfsprinzipien 1
4. Struktur und Entwurfsprinzipien der einzelnen Pakete 2
   4.1 Model 2
   4.2 Controller 3
   4.3 View 3
5. Datenmodell 4
6. Test 5

1. Allgemeines

Im Rahmen dieser Softwarestudie soll ein Prototyp der Online-Plattform zur Terminkoordination entstehen. Mit diesem Prototyp soll die Durchführbarkeit des Gesamtprojektes abgeschätzt und erste Funktionen des Gesamtprojektes umgesetzt werden. Zu den ersten Funktionen werden insbesondere der Login des Benutzers, die Ansicht des Kalenders inklusive Events, das Erstellen und Bearbeiten von Events gehören.

2. Produktübersicht

Die Softwarestudie wird in einem Webbrowser zu betrachten sein. Nach dem erfolgreichen Einloggen auf der Login-Seite kann der Nutzer seinen Terminkalender betrachten.

Mittels Klicken auf einen ausgewählten Kalendertag gelangt der Nutzer zu der Seite, über welche ein Event erstellt werden kann. Hierbei kann er alle notwendigen Informationen zu dem Event eingeben und es dann erstellen. Das Event erscheint danach im Terminkalender.

Durch Anklicken des erstellten Events gelangt der Nutzer dann zur Anzeige des Events mit allen Details, wobei er durch Klicken auf einen Button das bestehende Event bearbeiten kann.

3. Grundsätzliche Struktur- und Entwurfsprinzipien

Da das Projekt als Webapplikation realisiert werden soll setzen wir auf das verbreitete Framework Ruby on Rails. Grundprinzip ist dabei das Model-View-Controller Design, kurz MVC. Somit teilt sich die Software in 3 grobe Bereiche: Das Model ist für die Speicherung und Verarbeitung der Daten zuständig. Es kümmert sich sowohl um die Anbindung an die Datenbank, als auch um die Verarbeitung und Validierung der Daten. Aufgabe der Views ist es, die fertig verarbeiten Daten grafisch Darzustellen. In unserem Projekt wird es speziell darum gehen Html dynamisch mit Daten zu füllen und zu rendern. Das letztliche Verbindungsstück zwischen allen Komponenten und den Nutzern ist der Controller. Es nimmt die vom Nutzer angeforderten oder gesendeten Daten entgegen und entscheidet wie diese verarbeitet werden. Es informiert gegebenen Falls ein Model oder ein View und liefert die Antwort am Ende an den Nutzer aus.

Den Prinzipien von Rails folgend werden wir die Anwendung möglichst ressourcenorientiert anzulegen. Das bedeutet im Allgemeinen eine klare Aufteilung des Programmcodes in reale bzw. abstrakte Objekte. So beinhaltet unser Applikation verschiedene mehr oder weniger abstrakte Objekte wie Nutzer, Events sowie die Involvierung der Nutzer in ein Objekt. Diese logische Aufteilung bestimmt also letztlich auch die praktische Aufteilung der Codefragmente. Jedes Objekt, welches also in der Anwendung benötigt wird, erhält daher ein eigenes Model, welches in unserem Fall gleichzeitig immer an eine eigene Datenbanktabelle angebunden ist. So stellt also jeder Eintrag in der Tabelle Users eine Instanz der Klasse User dar.

Ähnlich verhält es sich mit Controllern. Wir erstellen für jede Ressource die später über unsere Webapplikation abrufbar und editierbar sein soll einen eigenen Controller. Dabei orientieren wir uns am Prinzip des „Single point of responsibility“. Dabei ist jede Klasse für genau eine Aufgabe zuständig.

Dieses Vorgehen ist insbesondere im Blick auf die bevorstehende Implementierung einer REST Schnittstelle von Vorteil, da so später nur noch wenig Funktionalität verändert oder hinzugefügt werden muss.

Zusätzlich setzen wir viele Bibliotheken (in Ruby Gem genannt) ein, welche bereits viele Funktionen implementieren.

Zur Nutzerverwaltung benutzen wir das Gem *devise*. Im Zusammenspiel mit einem weiteren Gem namens *cancan* entsteht so eine umfangreiche Nutzerkontensteuerung mit access controll lists.
Des Weiteren setzen wir auf r*ails admin*, welches die Benutzer in die Lage versetzt die meisten Daten wie: Nutzer, Rollen, Kraftwerke, materielle Ressourcen, sowie die Ressourcenkategorien grafisch anspruchsvoll und einfach zu verwalten.

4. Struktur und Entwurfsprinzipien der einzelnen Pakete

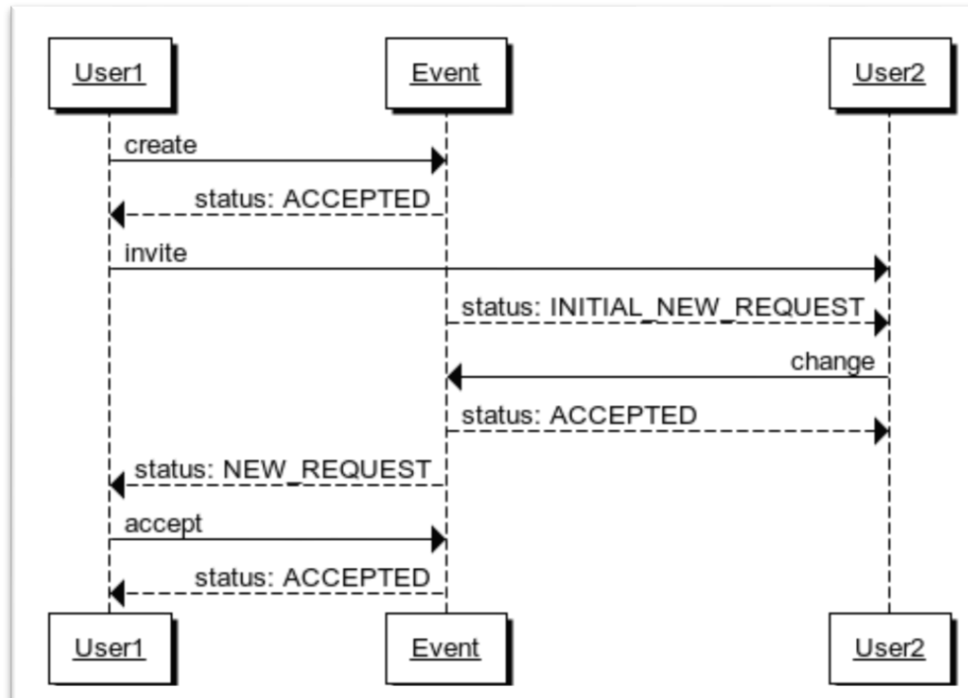
4.1 Model

Unser Datenmodell besteht aus zwei Hauptklassen, den Benutzern (user) und Terminen (events). Das User-Model kümmert sich dabei um die Verwaltung der Benutzer wodurch es die meiste Funktionalität aus der Bibliothek devise bezieht.

Da Benutzer in der Regel an mehreren Terminen teilnehmen und ein Termin mehrere Teilnehmer hat, ist es nötig beide Klassen durch ein weiteres Model, InvolvedEvent, zu verknüpfen. Diese Klassen speichert die Verbindung zwischen Nutzern und Terminen, sowie den Status des jeweiligen Nutzers im Bezug auf den Termin. Aktuell gibt es 3 verschiedene Status: INITIAL_NEW_REQUEST bedeutet, dass das Event neu erstellt wurde und der betreffende Nutzer noch nicht bestätigt hat. Bestätigt der Nutzer, wird sein Status zu ACCEPTED geändert. Wird ein Termin verändert nachdem der Termin vom Nutzer bestätigt wurde, ändert sich der Status zu NEW_REQUEST. Der Status des erstellenden bzw. bearbeitenden Nutzers ist automatisch ACCEPTED.

Um die gewünschte Funktionalität zu erhalten ist es nötig, dass Nutzer einem Termin benötigte Ressourcen zuordnen können. Die dafür benötigte Struktur ist komplex.

Zunächst besitzt ein Nutzer verschiedene Ressourcen. Diese werden durch die Klassen MaterialResource abgebildet. Eine solche Ressource wird wiederum einem ResourcenTyp zugeordnet, welcher die Art der Ressource definiert (zum Beispiel: Schraubenschlüssel oder Kran). Die Struktur der Typen wird dabei hierarchisch aufgebaut. Soll nun ein Termin eine bestimmte Ressource benötigen, muss dem Termin eine neue Instanz der Klasse ExpectedMaterialResource zugeordnet werden. Diese Klasse speichert den Ressourcentyp der vom Nutzer für das Event verlangt wird. Nun kann ein anderer Nutzer dieser ExpectedMaterialResource eine von ihm erstellte MaterialResource zuordnen und somit bestätigen, dass er das geforderte Material zum Termin mitbringt. Dabei muss der Typ der MaterialResource mit dem Typ der ExpectedMaterialResource übereinstimmen bzw. ein Kindelement dessen sein.



4.2 Controller

Die Aufteilung der Controlller wird von den Konventionen des Framworks Ruby on Rails bereits relativ klar vorgegeben. Dieses orientiert sich stark am RESTfull Ansatz, was im konkreten Fall bedeutend, dass für jede Ressource, welche von außen erreichbar sein soll ein Controller erstellt wird. Der jeweilige Controller besteht dabei mindestens aus einer Teilmenge aus den Operationen: index, show, new, create, edit, update und delete, wobei jede Methode genau eine Aufgabe erfüllt und fest mit einem passenden HTTP-Verb verknüpft ist (index,show,new,edit → GET, create → POST, update → PUT, delete → DELETE). Zusätzlich ist jede Methode in der Lage auf verschiedene Antworttypen zu reagieren. Es kann in jeder Methode spezifiziert werden welche Ausgaben gerendert werden können. Fordert der Client zum Beispiel eine Antwort im JSON-Format an, rendert der Controller die gewünschten Daten als JSON Objekt. Genauso können auch iCal oder HTML Dateien zurückgegeben werden. Dieses Schema erleichtert es, im Hinblick auf das Gesamtprojekt, sehr eine Restschnittstelle zu Implementieren.

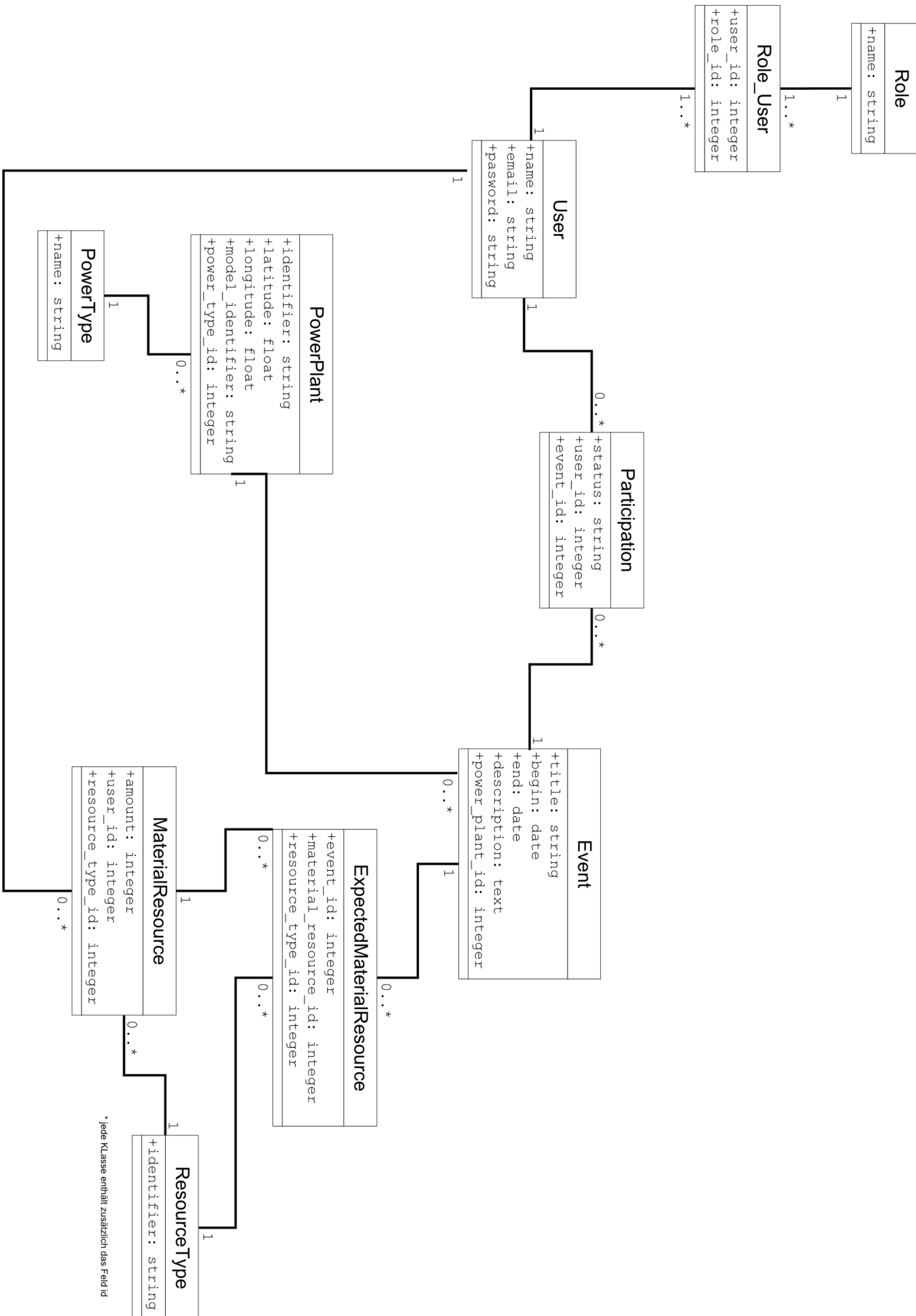
Unser Vorprojekt besteht dabei aus 4 Controllern. Der ApplicationController ist der Standartcontroller von dem alle anderen Controller erben. Hier werden Aktionen definiert, welche später für jeden Controller gelten sollen. Der HomeController ist für die Darstellung des Dashboard zuständig. Er zeig dem Nutzer die wichtigsten Informationen und dient als zentraler Anlaufpunkt. Im UserController werden alle wichtigen Funktionen für die Verwaltung der Nutzer definiert. Die meisten Funktionen werden bereits durch das Gem devise bereit gestellt. Schließlich kümmert sich der EventsController um die Verwaltung der Events. Er erstellt und bearbeitet die Events, sowie deren Abhängigkeiten.

4.3 View

Die Views stellen alle benötigten Templatedateien zur Verfügung. Sie werden vom Controller aktiviert, falls dieser das bereitgestellte Format rendern möchte. Sie erhalten dabei im Normalfall einen Satz Daten, welche sie dynamisch in das Template einbauen und übergeben die fertig gerenderte Datei an den Controller, welcher sie ausliefert. Views können dabei für alle vom Projekt benötigten Formate erstellt werden.

5. Datenmodell

Für unser Vorprojekt verwenden wir folgendes Datenmodell:



6. Test

Da es schwer ist in einem wachsenden Projekt die korrekte Funktion aller Teile einer Funktion zu überwachen benutzen wir ein in Ruby integriertes Testframework namens *TestUnit*. Dieses Framework stellt eine Reihe von Funktionen bereit, die es uns erlauben sowohl alle Klassen, als auch Funktionsabläufe zu testen. Dafür werden kleine Funktionen erstellt, welche eine bestimmte Teilkomponente oder Aufgabe testen, bei der das Ergebnis bekannt ist. Das Ergebnis der Funktion wird anschließend mit dem erwarteten Wert verglichen. Stimmen beide Werte nicht überein schlägt der Test fehl.

TestUnit gliedert sich in 3 Arten von Tests. Die Unit – Tests werden für das Model benötigt, die Functional – Tests für das Testen von Controller und View und die Integration – Tests für die Gesamtfunktionalität der Applikation.

Beim Erzeugen eines neuen Rails-Projekts werden automatisch die Ordner „functional“ , „unit“ und „integration“ erstellt. In diesen Ordnern wird der Testcode für die einzelnen Bestandteile der Applikation (Model, Controller, View) geschrieben. Außerdem wird noch der Ordner „fixtures“ automatisch generiert, in dem sich die Testdaten befinden.