Entwurfsbeschreibung

Allgemeines

Die zu entwickelnde Software ermöglicht Kooperation zwischen mehreren und Projektteilnehmern. Dazu können die Projektteilnehmer Kontakte, Termininformationen und Dokumente anlegen und neue Projektteilnehmer einladen. Zur Vereinbarung gemeinsamer Termine durch Anlegen einer entsprechenden Setie vorgeschlagen werden und ausgewählte Projektteilnehmer könnten diesen Termin dann bestätigen. Ein rollenbasiertes Rechtesystem (durch Gruppen realisiert) steuert innerhalb des Projektnetzwerks die Zugriffsrechte der Projektteilnehmer auf die bereitgestellten Ressourcen.

Produktübersicht

Die zu entwickelnde Software wird als Erweiterung der Software crabgrass entwickelt. Crabgrass ist eine web-basierte eine an sozialen Netzwerken ausgerichtete Kollaborationssoftware, die hauptsächlich von dem Technologiekollektiv Riseup entwickelt wird [1]. Sie ist primär strukturell auf die Zusammenarbeit zwischen Individuen und Interessengruppen (wie z.B. NGOs [2]) ausgerichtet, die gemeinsame gesellschaftliche Ziele verfolgen und sich diesbezüglich koordinieren müssen.

Die Software ermöglicht das Anlegen von Benutzerprofilen, das Knüpfen von Kontakten zwischen Benutzern, den Nachrichtenaustausch zwischen Benutzern, und Microblogging. Benutzer können sich dabei in Gruppen organisieren, die sich wiederum mit anderen Gruppen oder Benutzern in Netzwerken organisieren können.

Jeder Benutzer, jede Gruppe und jedes Netzwerk kann für sich spezifische Seiten verschiedener Typen anlegen. Standardmäßig werden Wiki-Seiten, Gruppendiskussionen, Artikel, verschiedene Arten von Abstimmungen, (versionierte) Dateien, eingebundene Videos, Bildergallerien, Aufgabenlisten, Termine und Ankündigungen unterstützt, für die jeweils auf Gruppen und Benutzerbasis Berechtigungen vergeben werden können, unterstützt.

[1] Aus der Selbstdarstellung: „The Riseup Collective is an autonomous body based in Seattle with collective members world wide. Our purpose is to aid in the creation of a free society, a world with freedom from want and freedom of expression, a world without oppression or hierarchy, where power is shared equally. We do this by providing communication and computer resources to allies engaged in struggles against capitalism and other forms of oppression.“

[2] engl. non-governemental organisation - ein nichtstaalicher, zivilgesellschaftlicher Interessenverband, der sich üblicherweise abseits von Profitorientierung für soziale oder politische Veränderungen engagiert

Das Zugriffskontrollsystem scheint (ein formaler Beweis wäre noch zu erbringen), wenn auch terminologisch und weltanschaulich verschieden, formal mit dem in ANSI INCITS 359-2004 definierten Core RBAC Modell übereinzustimmen, mit der Beschränkung, dass keine explizite Rollenaktivierung möglich ist, sondern alle Rollen gleichzeitig aktiviert sind. Das ist mit Hinblick auf das Dreieck von Zooko recht sinnvoll ist, da zwar auf der einen Seite eine Zugriffskontrolle erfolgen und eine gewisse Privatheit geboten werden soll, aber auf der anderen Seite auch nicht technisch-versierte Personen, die zur freiwilligen Mitarbeit motiviert werden sollen und anders als vielleicht Mitarbeiter großer Firmen und Regierungen nicht extra geschult werden können.

Crabgrass bietet somit bis auf die notarielle Beurkundung, einer Bestätigungsmöglichkeit für Termine und einem zweckmäßigen Kalender alle notwendigen Funktionen, die für das Produkt gefordert werden. Bei einer fast ausschließlich serverseitigen Webanwendung ist notarielle Beurkundung nicht sinnvoll umsetzbar und sollte somit außerhalb des Systems (z.B. mit Hilfe qualifizierter elektronischer Signaturen) durchgeführt werden. [3] Als Ersatz der Notarfunktion würde es auch eher der Ideologie von crabgrass entsprechen, sinnvoller eine Abstimmungsfunktion für versionierte Seiten (Wiki-Seiten, Dateien) zu ergänzen.

Es wird angestrebt, die entstandenen Erweiterungen und Änderungen in das Projekt zurückfließen zu lassen und diese zu einem solchen Reifegrad zu bringen, dass sie produktiv eingesetzt werden können, denn gerade vor dem Hintergrund jüngster politischer Entwicklungen und Aufstände, scheint eine freie (im Sinne von Freiheit) Kommunikationsplattform zur politischen Organisation dringend gebraucht[4].

Grundsätzliche Struktur und Entwurfsprinzipien des Gesamtsystems

Crabgrass wurde mit Ruby on Rails, einem dem Model-View-Controller-Prinzip folgenden Webframework für die Programmiersprache Ruby, entwickelt und nutzt die Datenbank MySQL. (Die Datenbankabstraktion [5] von Ruby on Rails unterstützt zwar auch andere Datenbanken, sofern ein Adapter für `ActiveRecord` existiert bzw. eine gewisse Untermenge von SQL implementiert ist, aber crabgrass nutzt MySQL-spezifische Funktionen).

Wie bei den meisten Webanwendungen beschränkt sich die Aufgabe von crabgrass auf das Auslesen, Schreiben, Suchen und Anzeigen von der Daten der Datenbank. Für Erweiterungen der Software bedeutet das, mit Hinblick auf das Model-View-Controller-Prinzip, dass Anfragen an eine bestimme URI im Controller entgegen genommen werden, ausgewertet und in entsprechende Datenbankanfragen überführt werden. Die Ergebnisse der Datenbankabfragen werden dann mittels Templates und Hilfsfunktionen, die HTML-Bereiche oder Javascript-

[3] Zur Umsetzung gesetzlicher Vorgaben, einschließlich entsprechender Zertifizierung ist crabgrass schlecht geeignet. Viele Entscheidungen beim Entwurf und bei der Umsetzung der Software gingen von der Annahme aus, dass sich Nutzer innerhalb derselben Gruppe gegenseitig trauen.

[4] Auf lange Sicht ist jedoch höchst zweifelhaft, ob zentralisierte Dienste und Webandwendungen dafür die adäquate Lösung sind.

[5] Abstraktion bedeutet hier im Sinne des Worts auch Verlust und Beschränkung von Funktionalität, denn die Abstraktion stellt nur eine gemeinsame Untermenge dessen, was sie abstrahiert, zur Verfügung, so dass viele Datenbank-spezifische Funktionen nicht oder nur durch Umgehung der Abstraktion genutzt werden können.

Abschnitte erzeugen, in eine Darstellung für den Nutzer umgewandelt. Die Erweiterungen im Rahmen des Praktikums betreffen hauptsächlich Views und Controller. Die Datenmodelle für die benötigten Funktionen existieren bereits und müssen nur geringfügig (durch Erweiterung um wenige Attribute) angepasst werden.

Üblicherweise verläuft die Verarbeitung einer Anfrage an eine Rails-Applikation wie folgt: Die URL der Anfrage wird den Routing-Routinen der Komponente `ActionController` übergeben, die nach vorgegeben Regeln die Anfrage einen Controller und einer Aktion dieses Controllers zuordnet. Ergänzende Angaben in der URL werden als Paramter in den Assoziativspeicher `params` abgelegt. In dem Kontroller wird dann die zugeordnete Aktion ausgeführt. Dabei wird häufig mit einem oder mehreren zugehörigen Datenmodellen (im Folgenden um der Kürze Willen Modell genannt) interagiert, um Datensätze zu modifizieren und Informationen für die nächste Anzeige für den Nutzer bereit zu stellen. Diese Modelle erscheinen durch objektrelationales Mapping gegenüber der Anwendungslogik als Ruby-Klassen, werden aber durch die die Komponente `ActiveRecord` als Datensätze in einem relationalen Datenbanksystem [6] persistiert. Nachdem die mit der Aktion verbundene Geschäftslogik abgewickelt ist, wird aus dem Controller heraus die Komponente `ActionView` angewiesen, eine passende Anzeige zum Resultat der Aktion zu generieren. Hierzu sucht `ActionView` passende Templates, die das Layout der zurückzugebenden Seite beschreiben, fügt an passenden Stellen die sich dynamisch ändernden Inhalte ein und verbindet diese Teile im Prozess des Rendering zur einer neuen Seite, die als Antwort auf die Anfrage zurückgegeben wird.

Das Ruby on Rails gibt starken Anreiz, sich an Konvetiontionen bezüglich der Speicherorte, Dateinamen und Bezichner vieler Programmbestandteile zu halten, da beim Einhalten dieser Konventionen der Konfigurationsaufwand sinkt und häufig benötigte Initialisierungen und Importvorgänge implizit von Ruby on Rails übernommen werden [7]. Erhält eine Rails Applikation zum Beispiel die Anfrage „/event/edit/42“, wird diese konventionsgemäß einem `EventController` zugeordnet, in dem die Methode `edit` gesucht wird. Dieser Kontroller erhält automatisch Zugriff auf Hilfsmethode einer Klasse `EventHelper` und einem Datenmodell namens `Event` (welches in der Datenbank in der Tabelle `events` zu finden sein wird) und so weiter. Da die Menge dieser Konventionen umfangreich und nicht nur spezifisch für dieses Projekt sind, sei für sie auf externe Literatur verwiesen [8]

In Crabgrass wurde die vorgegebene Verzeichnisstruktur nach Rails-Konvention um die Verzeichnisse features, mods, tools und app\permissions erweitert. Die Verzeichnisse mods und tools sind Erweiterungspunkte für die Basisanwedung (genauer beschrieben im kommenden Absatz). In features werden die Beschreibungen für Szenarien und erwartetes Programmverhalten für das Cucumber Test Framework abgelegt. Methoden und Funktionen, die innerhalb des Programms über Zugriffsberechtigungen von Benutzern auf Seiten und Aktionen einer Crabgrass-Instanz entscheiden, werden in seperate Module unter app\permissions ausgelagert.

Neben der Erweiterung von crabgrass durch Modifikation der eigentlichen Anwendung lässt sich crabgrass durch Mods und Tools erweitern. Tools fügen neue Seiten-Typen hinzu und

[6] Es ist auch möglich, mit `Active Record` über nicht relationale Datenbanksysteme zu abstrahieren, allerdings wird von dieser Möglichkeit selten gebrauch gemacht, da Vorteile und Besonderheiten alternativer Datenbankarchitekturen schwer durch die API von `Active Record` zu erreichen sind.

[7] das Prinzip von 'Convention over Configuration'

[8] etwa auf Hansson, Ruby, Thomas: Agile Web Development with Rails, 3. Ausgabe, The Pragmatic Bookshelf, Raleigh, USA. Seite 275ff. und Seite 268ff.

durch Mods werden sonstige Erweiterungen und Veränderungen abgedeckt. Tools definieren für ihre Views Templates, die Anzeige der Tool-spezifischen Inhalte übernehmen (partial templates, oder kurz parials). Diese partials werden durch andere partials für ständig oder häufig darzustellende Seiteninhalte (Navigationsleise, Sidebar und Ähnliches) eingebettet, die zentral in der Hauptapplikation definiert sind. Dadurch ist einheitliche Gestaltung und Handhabung generischer Seitenbestandteile sichergestellt.

Da Ruby on Rails ein synchrones Webframework ist und im wesentlichen keine Hintergrundaufgaben unterstützt [9], ist crabgrass an sich nochmals in die mit Ruby On Rails entwickelte Webschnittstelle und Hintergrunddienste (Indizierung mit Sphinx, Hintgrundaufgaben mit BackgroundDRb, Cron-Jobs) aufgeteilt, die vor allem der Volltextsuche und rechenintensiveren Verarbeitung von Bildern, Videos und Dokumenten dient.

Grundsätzliche Struktur und Entwurfsprinzipien der Pakete

Der Kalender wurde basierend auf einem externen Rails-Plugin zur Kalenderdarstellung entwickelt. Zur Einbindung des Kalenders in das Projekt wurden jeweils für Nutzer und Gruppen passende Controller angelegt und zu ihrer Darstellung partials erstellt und eingebunden. Dafür sind bezüglich des Datenflusses nur ohnehin im System vorhandene Daten mittels einer bereits für Crabgrass entwickelten internen Such-Engine zu suchen und zusammenzutragen.

Der Kaleder stellt Termine monatsweise dar und enthält neben den Verweisen auf die Seiten der einzelnen Termine auch Verweise auf den vorhergehenden Monat und den Folgemonat.

Das Datenmodell der Termine war um ein Datumsattribut für die Bestätigungsfrist zu erweitern. Zudem wich die Gestaltung und technische Umsetzung zur Darstellung der Termininformationen stark von den Prinzipen ab, die mehrheitlich bei anderen Seitentypen des crabgrass-Projektes befolgt wurden. Für ein besseres Erscheinungsbild und einheitlichere (und somit zugänglichere) Benutzerführung wurden die Views für die Aktionen Termin anlegen, Termin anzeigen und Termin bearbeiten grundlegend überarbeitet, so dass sie nun einheitlich denselben Formulargenerator nutzen, der auch bei vielen anderen Formulardarstellungen in Crabgrass Anwendung findet. Zudem wurde die Darstellung der bestätigenden bzw. (noch) nicht bestätigenden Nutzer verbessert: Statt der ursprünglich vorhandenen Auflistung der Nutzernamen als Text wurde eine Aufreihung der Icons der entsprechenden Nutzer (mit entsprechenden Verweisen) aus der Sidebar übernommen.

Termine

In der Entwicklerversion von Crabgrass, auf die dieses Projekt aufbaut, war bereits ein `event_tool` vorhanden, dass das Anlegen von Terminen ermöglichte und berechtigte Nutzer konnten Rückmeldungen (im Sinne von Ich nehme Teil oder ich nehme nicht Teil) abgeben konnten. Ein Frist für solche Rückmeldungen war allerdings nicht vorgesehen und das Layout der zugehörigen `EventPage` sehr minimalistisch und fügte sich nicht in das Gestaltungsgefüge ein, dass durch die meisten anderen Crabgrass-Seiten vorgegeben war.

[9] Die meisten Erweiterungen lagern diese in Hintegrundprozesse aus, weil MRI zur Zeit noch einen Global-Intpreter-Lock hat, so dass effektiv bei rechenintensiven Aufgaben nur ein Thread gleichzeitig ausgeführt wird.

Damit die gewünschten Fristen für Rückmeldungen für Termin berücksichtigt werden können, wurde ein neues Attribut `deadline` für die Frist zum Modell `Event` hinzugefügt. Rückmeldungen für Termine werden nun allgemeiner als Bestätigung (bzw. das Fehlen einer Bestätiung) umgedeutet, da es auch Termine geben kann, die Abstimmung bedürfen, ohne dass sie mit einer expliziten Teilnahme verbunden sind (z.B. das Einreichen eines Dokuments). Zudem wurden zum `Event`-Modell Validierungen hinzugefügt, die die Plausibilität der eingegeben Zeitpunkte eines Termins beim Erstellen und bei Änderung überprüfen. Es wird etwa geprüft, ob der Zeitpunkt für das Ende eines Termins nach dessen Beginn liegt, ob die Frist für Bestäigung (sofern vorhanden) vor dem Startzeitpunkt des Termins abläuft und ob nicht bereits einer dieser Zeipunkte verstrichen ist. Das Attribut `deadline` wird nullwertig interpretiert: Der Wert `nil` zeigt an, dass es keine Frist für Bestätigung vorhanden ist.

Während das Modell Termine (die Klasse `Event`) in der Hauptapplikation zu finden ist, wurden fast alle Views sowie Controller und Hilfsmethoden für die Darstellung von und Interaktion mit Terminen in die Erweiterungen `tool\event_tool` aus gelagert. In dieser Erweiterung wird das Modell einer `EventPage` angelegt, dessen Datenmodell dem einer generischen Crabgrass-Seite entspricht. Jeder Termin ist mit `EventPage` assoziiert und alle Nutzer, die zumindet Schreibrechte auf eine entsprechende `EventPage` haben, können den Termin bestätigen. Ob ein Termin von einem Nutzer bestätigt wurde, wird durch den Wahrheitswert des Attributs `attend` des Modells `UserParticipation` signalisiert.

Das Modell `UserParticipation`, ein Entität, die `User` und `Page` in einer 1:1-Beziehung assoziiert, enthält auch das Attribut `access`, dass die Zugriffsberechtigungen des entsprechenden Nutzers auf die Seite festlegt. Abfragen über die Zugriffsberechtigungen erfolgen allerdings durch die Prädikatsfunktion `may?` aus `PermissionsHelper` bzw. über davon abgeleitete Prädikatsfunktionen der Form `may_[Aktion]?` die aus den Modulen `BasePagePermissions` und `EventPagePermissions` in den `EventController` einbunden [10] sind. Um nun alle Nutzer aufzulisten, die einen Termin nicht bestätigt haben, wird mittels der `may?`-Funktion aus `permissions` nach allen Nutzern gesucht, die zumindest Schreibrecht auf eine `EventPage` zu einem Termin besitzen (denn das ist gerade die Menge der Nutzer, die den Termin bestätigen könnten). Zieht man von dieser Menge jene Nutzer ab, bei denen das oben genannte `attend` Attribut wahr ist, erhält man die Menge nicht-bestätigenden Nutzer.

Die vom Event Tool bereitsgestellten substatiellen partials zur Eingabe oder Anzeige der Termininformationen für die Aktionen `create, show, edit` wurden mit dem Formulargenerator `Formy` erstellt. Um `Formy` auch für darstellende Seiten und nicht nur für Eingabeformulare nutzbar zu machen, wurde für den das Formularelement `Row` zu `label` und `input` das Element `content` als weiteres mögliches Unterlement ergänzt. Die Textfelder zur Eingabe von Zeitpunkten wurden um Javascript-Widgets zur Auswahl von Datum und Uhrzeit erweitert. Diese Widgets stammen von einem externen Rails-Plugin namens calendar_date_select, das bereits in Crabgrass eingebunden war, aber noch nicht benutzt wurde. Da für Termine auf ganztägige Angaben ermöglicht werden sollten, war im Formular interaktiv je nach an- oder abgewählter Checkbox für „ganztägig“ zwischen den Widgets umzuschalten, die die Auswahl von Datum und Uhrzeit oder nur die Auswahl von Datum erlauben. Da sich in den Prototype-Bindings von der Rails-Plattform für eine solche Aufgabe keine geeigneten Adapterfunktionen fanden, die ohne unnötige Ajax-Anfragen operieren, wurden für diese Aufgabe zwei kurze

[10] 'einbinden' bedeudet hier, das Übernehmen von Funktionen und Methoden durch Mixin-Compostion

Inline-Javascriptblöcke entwickelt, die an die oben genannten Widgets und der besagten Checkbox für die Eventlistener `before_show` bzw. `on_click` registriert.

Ein neuer Termin(`Event`) entsteht implizit beim Anlegen einer neuen `EventPage`. Beim Anlegen der Seite wird die Methode `build_page_data` aufgerufen, die aus den Parameter der Aktion zum Erstellen der Seite die Attribute für das zugehörige `Event`-Objekt extrahiert und anschließen wird ein solches Ojekt erstellt. Das `Event`-Objekt einer `EventPage` ist dann durch ihr Attribut `data` erreichbar. Das Ändern eines Termins folgt einem sehr ähnlichen Muster: Die geänderten Angaben werden aus den Parametern, das vom Formular bei der Anfrage übergeben wird, in gleicher Art und Weise mittels der Controller-Methode `parse_event_params` ausgelesen und umgeformt, so dass mit ihnen direkt die Attribute des ensprechenden `Event`-Modells aktualisiert werden können. Sowohl bei Erstellen als auch beim Ändern prüfen die in `Event` deklarierten und formulierten Validierungsfunktionen die Sinnhaftigkeit der Zeitangaben.

Kalender

Zur Umsetzung der Kalenderdarstelltung wurde auf das Rails-Plugin event_calendar zurückgegriffen, das unabhängig von Crabgrass entwickelt wurde. Dieses Plugin ist als seperates Gem verfügbar und wurde eine Rails Engine mittels `Railtie` dem Projekt zugänglich gemacht. Dadurch werden die vom event_calendar bereitgestellten Hilfsfunktionen jedem Controller zugänglich gemacht und `ActiveRecord::Base` (die Basisklasse für Objekte der ORM-Abstraktion) werden um einige Klassenmethoden erweitert.

Um die Termine zu finden, die im Kalender anzuzeigen sind wurde jeweils in den Namensräumen `Groups` und `Me` ein `CalendarController` angelegt. Diese Controller verwenden die für Crabgrass entwickelte pfad-basierte Such-Engine path finder mit der Methode `Page.find_by_path`, um nach allen `EventPages` zu suchen, die dem Nutzer bzw. der Gruppe angezeigt werden können. Von jeder `EventPage` aus der Treffermenge dieser Suche wird das damit asssoziierte `Event`-Objekt abgerufen.

Um eine Darstellung der Menge der anzuzeigenden `Event`-Objekte zu erhalten, werden diese Methode `create_event_strips` des event_tool Plugins übergeben, die diesen Objekten die Informationen extrahiert, die für Kalendereintäge benötigt werden, wobei zu diesen Termineinträgen noch eine Verknüpfung zur jeweiligen `EventPage` hinzugefügt wird.

Der Kalender ist somit nur eine zur Laufzeit generierte Aggregation von Daten anderer Modelle und ist im Sinne des ER-Modells als attributlose Entität zur verstehen, die in N:M-Beziehung zu `EventPages` steht. Daher existiert für den Kalender auch kein eigenständiges Datenmodell.