Timetraces Seminararbeit Handheld

Marco Wettstein

Contents

List of Figures

Listings

1 Vorwort

2 Ist-Analyse

Ein zentraler Aspekt der Arbeit sind die erwähnten Kontext-Daten der Benutzer, doch nicht jeder Mitarbeiter hat die gleiche Art von Kontext. Ein Entwickler "dokumentiert" seine Arbeit häufig in einer Code-Versionisierungs-Software, für einen Projektleiter jedoch stehen vielleicht Meetings und entsprechende Kalender-Einträge im Vordergrund. Eine vorgängige Analyse der Rollen und verfügbaren Systemen ist daher unabdingbar.

2.1 Rollen

Nachfolgend eine Liste der Rollen, welche bei Panter vertreten sind. Manche Mitarbeiter nehmen mehrere Rollen ein. Manche Mitarbeiter arbeiten zudem häufig extern bei Kunden und nutzen teilweise andere Systeme. Software-Engineer Konzipiert, erstellt, testet und wartet Anwendungen und Systeme. Arbeitet häufig im Team in einem agilen Entwicklungsprozess (häufig Scrum).

Scrum-Master Leitet und überwacht den Scrumprozess. Er plant und moderiert häufig die Scrum-Meetings, wie Planning-Meeting und Daily Scrum-Meetings, sowie andere Scrum-Aktivitäten.

Product Owner-Assistent Der Product Owner-Assistent (PO-Assistant) wird dem häufig externen Product Owner zur Seite gestellt und unterstützt diesen beim Erstellen und Abnehmen der User Stories. Er

Sales Sales-Mitarbeiter beraten bestehende Kunden und potentielle neue Kunden über neue Projekte und nehmen an Pitches teil.

Administration Kümmert sich um Buchhaltung, Administation und Human Resources-Aufgaben.

Marketing Erstellt und setzt das Marketing-Konzept der Firma um, fördert die Sichtbarkeit der Firma und unterstützt dadurch die Bestrebungen der Sales-Mitarbeiter.

Community-Manager Kümmert sich um die Verwaltung des Cowork-Space "colab-zurich.ch". Zur Zeit (Ende 2014) eine Praktikumsstelle.

2.2 Systeme

Den Mitarbeitern stehen in der Regel folgende Systeme zur Verfügung, sofern sie nicht im externen Einsatz eingschränkt sind:

2.2.1 Controllr

Von Panter erstellte Software für das Finanz-Controlling, Zeiterfassung und Resourcenplanung. Es ist das zentrale System, welches die Anwendung anbinden soll. Zugriff erfolgt über ein Webinterface oder über eine REST-Schnittstelle.

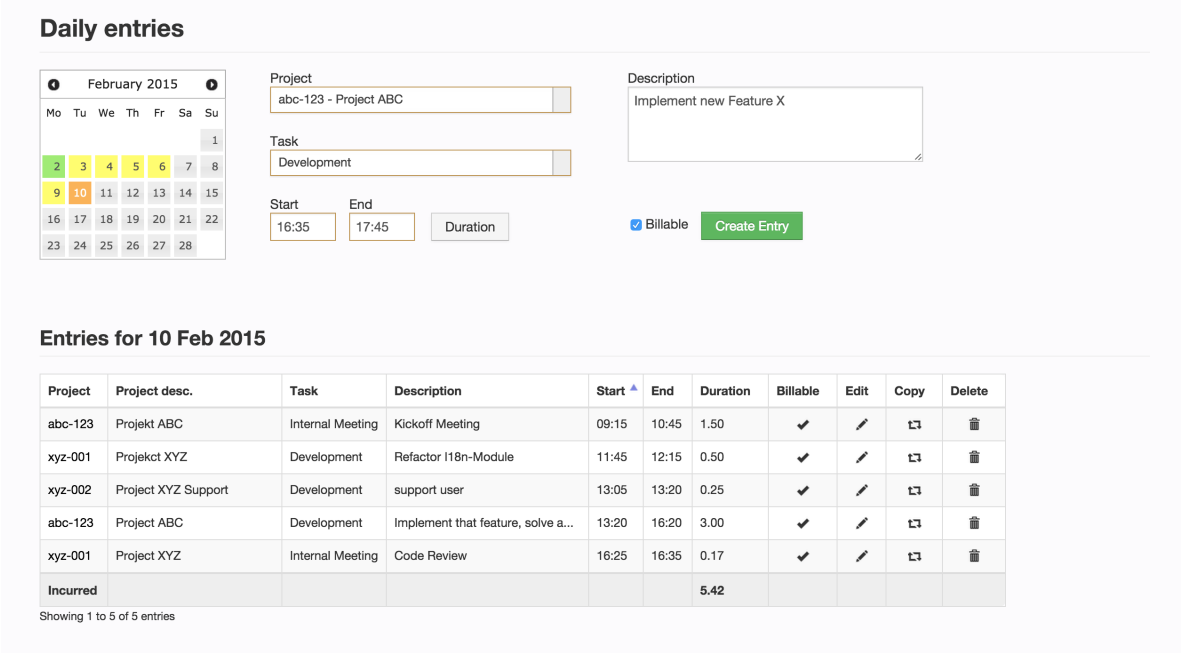

Figure 1: Screenshot von "Controllr"

2.2.1.1 Zeiteinträge Die REST-Schnittstelle von Controllr ermöglicht unter anderem das Lesen und Manipulieren von Zeiteinträgen, Projekten und Tasks. Im Listing 1 sind die Schnittstellen zu den Zeiteinträgen zu sehen. Diese Schnittstellen dienen zum Lesen (GET), Erstellen (POST), Bearbeiten (PUT / PATCH) und Löschen (DELETE) von Zeiteinträgen. Listing 2 zeigt das Schema eines solchen Zeiteintrages.

```
1  GET      /api/entries(.:format)
2  POST     /api/entries(.:format)
3  GET      /api/entries/new(.:format)
4  GET      /api/entries/:id/edit(.:format)
5  GET      /api/entries/:id(.:format)
6  PATCH    /api/entries/:id(.:format)
7  PUT      /api/entries/:id(.:format)
8  DELETE   /api/entries/:id(.:format)
```

Listing 1: Controllr REST API für Zeiteinträge

```
{
    "id": 12513,
    "created_at": "2014-09-15T14:25:07.000Z",
    "updated_at": "2014-11-13T14:45:31.000Z",
    "deleted_at": null,
    "day": "2014-09-15",
    "start": "2000-01-01T09:50:00Z",
    "end": "2000-01-01T10:00:00Z",
    "duration": 10,
    "state": "invoiced",
    "description": "Standup Meeting",
    "billable": true,
    "invoice_id": 123,
    "project_id": 1100,
    "task_id": 10042,
    "user_id": 12,
    "project_shortname": "abc-001",
    "task_name": "Internal Meeting",
    "user_username": "maw"
}
```

Listing 2: Schema von /api/entries

Auffallend ist "start" und "end", bei denen das Datum offenbar aus Formatgründen angefügt wird und ohne Relevanz ist. Lediglich die Zeit ist relevant. Für das Datum des Zeiteintrages ist "day" relevant. Dies bedeutet auch, dass jeder Zeiteintrag einem Tag zugeordnet ist, es kann keine einzelnzen Zeiteinträge geben, die über mehrere Tage gehen (z.b. über Mitternacht).

Manche relationale Daten sind zudem Denormalisiert (project_shortname und task_name).[1]

Der Lesezugriff (GET) auf diese Daten liefert jeweils ein Array dieser Einträge. Mit GET-Parameter können die Resultate gefiltert werden:

date_to, date_from Liefert die Einträge mit "day" zwischen date_from und date_to (inklusive)
employee_usernames Liefert nur die Einträge eines bestimmten Users
project_shortnames Liefert nur die Einträge eines bestimmten Projektes
states Liefert nur Einträge mit einem bestimmten state

2.2.1.2 Projekte

Projekte können über die Schnittstellen von Listing 3 abgerufen werden und liefern ein Array von Projekten wie im Schema 4

```
GET      /api/projects(.:format)
GET      /api/projects/:id(.:format)
```

Listing 3: Controllr REST API für Projekte

```
{
    "id": 1234,
    "shortname": "abc-001",
    "description": "Beispiel Projekt",
```

[1] Als Denormalisierung bezeichnet man das bewusste Einfügen redundanter Informationen einer relationalen Datenbank zu Gunsten eines besseren Laufzeitverhaltens oder einfacherem Zugriff. Im obigen Beispiel, wird neben der project_id auch der project_shortname mitgegeben, welcher direkt abhängig von der project_id ist. Das Denormalisieren entspricht der Umkehrung der Normalisierung.

```
5       "start": "2014-03-21",
6       "end": "2014-11-13",
7       "created_at": "2014-04-07T10:15:11.000Z",
8       "updated_at": "2015-02-06T17:50:39.000Z",
9       "project_state_id": 6,
10      "probability": "1.0",
11      "deleted_at": null,
12      "external": true,
13      "note": "Dies ist ein Beispielprojekt",
14      "worktime_budget": "1302.0",
15      "cached_expected_profitability": 0.581247,
16      "cached_expected_return": 685.5,
17      "company_id": 111,
18      "active": true,
19      "leader_id": 62,
20      "business_unit_id": 1,
21      "cached_budget": "112236.0",
22      "budget_notification_sent": false,
23      "mwst": true,
24      "daily_rate": "1200.0",
25      "hours_per_day": "8.0",
26      "google_id": "Gyqrt64o8AOywyeBGyqrt64o8AOywyeB",
27      "planning_note": "",
28      "time_billable": "1255:43",
29      "burned_time": "1301:47"
30  }
```

Listing 4: Schema von /api/projects

2.2.1.3 Tasks Die Tasks-Schnittstelle wird in Listings 5 und 6 beschrieben.

Es existieren noch weitere Schnittstellen, welche aber für die Arbeit weniger relevant sind.

```
1  GET      /api/tasks(.:format)
2  GET      /api/tasks/:id(.:format)
```

Listing 5: Controllr REST API für Tasks

```
1   {
2       "id": 6711,
3       "name": "Freelance vor Ort",
4       "project_id": 3330,
5       "created_at": "2011-11-24T19:02:08.000Z",
6       "updated_at": "2011-11-24T19:07:23.000Z",
7       "active": true,
8       "deleted_at": null,
9       "billable_by_default": true,
10      "worktime_budget": null,
11      "global_task_id": null,
12      "daily_rate": null
13  }
```

Listing 6: Schema von /api/tasks

2.2.1.4 Authentifizierung Für die Authenfizierung des Users muss ein Token (user_token) an die Schnittstellen mitgegeben werden. Der Token kann auf der Profil-Seite im Controllr abgefragt werden, nachdem man sich eingeloggt hat.

2.2.2 Google Apps

Für Email, Kalender und Dateien wird Googles Business Angebot "Google Apps" verwendet. Der Service kann über verschiedene REST-APIs abgefragt und bedient werden. Da Implementationen dieser Schnitstellen in vielen Sprachen bereits existieren, bietet es sich an, manche dieser Schnittstellen zu verwenden.

2.2.2.1 Email Business-Variante von Googles Mail-Lösung GMail. Die Anwendung wird im Browser bedient und kann zudem über eine REST-API abgefragt werden. Es können insbesondere Nachrichten, Threads (Zusammengehörende Nachrichten) und Labels abgefragt werden. Siehe ("Google Apps Email API").

Möglich wäre beispielsweise, Nachrichten nach Projekt-Namen aus "Controllr" zu durchsuchen und diese als Quelle zu verwenden.

2.2.2.2 Kalender Kalender-Anwendung von Google. Wird in der Firma häufig verwendet und kann ebenfalls über eine REST-API abgerufen weren. Kalendereinträge bieten sich insbesondere an, da diese bereits über ein ähnliches Format verfügen wie die Zeiteinträge; sie haben u.a. eine Start- und Endzeit, sowie eine Beschreibung. ("Google Kalender API").

2.2.2.3 Authentifizierung Die Authentifizierung wird OAuth 2.0 verwendet. Es existieren zahlreiche Implementierungen dieses Standards, was die Verwendung dieser Schnittstellen vereinfacht. ("Google OAuth2")

2.2.3 Redmine

Projektverwaltungs-Anwendung, welche von der Firma für viele Projekte verwendet wird. Die Projekte werden stets in einem agilen Prozess entwickelt, welcher meistens SCRUM ist. In Redmine befinden sich daher Stories und zugehörige Tasks, sowie deren Stand. Mitarbeiter, welche an externen Projekten beim Kunden arbeiten, verwenden allerdings häufig nicht Redmine, sondern jeweilige Firmen-Interne Anwendungen.

Redmine bildet nicht direkt typische SCRUM-Artefakte wie Stories und Tasks ab, sondern es werden üblicherweise "Issues" erfasst. Über Erweiterungen können aber Stories und Tasks ebenfalls erfasst werden, diese werden dann als unterschiedliche "Issue"-Typen erfasst.

Redmine bietet ebenfalls eine REST-API, welche es u.a. erlaubt, Issues und Projekte abzufragen. ("Redmine REST API")

2.2.3.1 Authentifizierung Für die Authentifizierung muss ein fester Token mitgegeben werden, welcher User-spezifisch ist und auf der Profil-Seite von Redmine abgefragt werden kann.

2.2.4 Github

Verwaltungsoberfläche und Hosting-Dienst für Software-Projekte, welche die namensgebende Quellcode-Versionsverwaltungs-Software git verwendet. Nicht alle Projekte verwenden Github für die Quellcode-Versionisierung. Insbesondere externe Projekte beim Kunden verfügen über eigene Versionisierungstools.

Github verfügt ebenfalls über reichhaltige REST-APIs; die Beschreibung dieser APIs kann in der Quellenangabe eingesehen werden. Es bietet sich an, die Schnittstelle "Events" zu verwenden, welche

beispielsweise Aktivitäten eines Benutzers aufzeigt. Damit kann die Tätigkeit eines Users auf Github an einem Tag abgefragt werden. Die Art der Aktivität und das Repository sind dabei zweitrangig. (https://developer.github.com/v3/)

Ein solches Event verfügt über einen Typ, eine Beschreibung, eine Identifizerung des Repositories und einen Zeitpunkt, an dem dieses Ereignis oder Aktivität stattgefunden hat.

2.2.4.1 Authentifizierung Github unterstützt verschiedene Authentifizierungsverfahren: Basic Authentication mit Username / Password, OAuth2 mit Token oder OAuth2 mit Key/Secret , siehe ("Github Authentifizierung").

2.2.5 Timetunnel

Der Timetunnel ist eine von Panter entwickelte Software, welche es ermöglicht, Zeiteinträge direkt über den Kalender zu tätigen. Die Software besteht im wesentlichen aus einem Script. welches Kalendereinträge mit definierten Stichworten im Titel sucht und als Zeiteinträge für die Zeiterfassung in Controllr erfasst. Das bedeutet, Zeiteinträge können direkt im Kalender erfasst, statt im Web-GUI. Die Anwendung wird nach erster Abklärung kaum genutzt, allerdings können Ideen davon, wie z.b. das Tagging der Einträge wiederverwendet werden.

2.2.6 strms.io (vormals storyline.li)

Sich in Entwicklung befindende Software, welche von Panter entwickelt wird. Die Anwendung konsolidiert verschiedene Google-Dienste wie Email, Kalender und Google Drive und stellt die Aktivitäten eines Benutzers auf diesen Diensten chronologisch dar. Ziel ist es, Emails, Kalendereinträge und Dateien in einen Zusammenhang zu bringen. Die Anwendung bietet sich als Datenquelle an, weil sie bereits mehrere Quellen vorkonsolidiert und chronologisch ordnet. Das Projekt befindet sich zum Zeitpunkt dieser Arbeit in Entwicklung. Eine Zusammenarbeit, sowie Wissensaustausch ist möglich. Es existieren allerdings zum Zeitpunkt der Entwicklung dieser Arbeit keine Schnittstellen, welche einen programmatischen Zugriff erlauben.

2.2.7 Systemübersicht

Abbildung 2 zeigt eine Übersicht der Syteme und eine mögliche Integration der zu erstellenden Applikation.

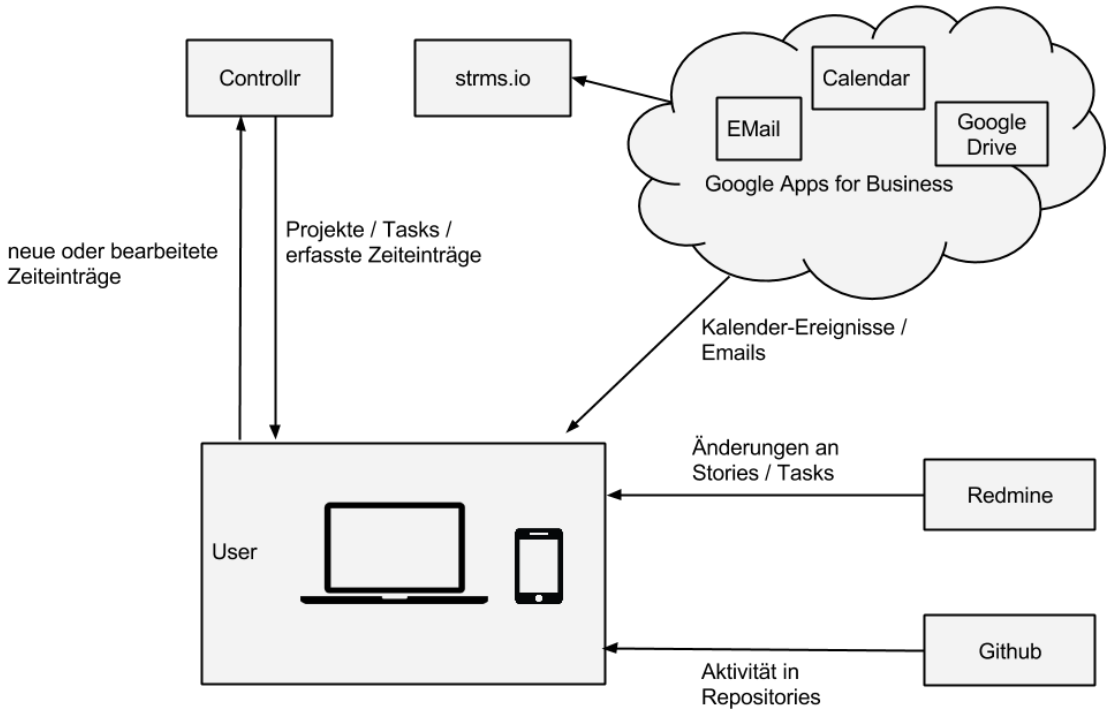

Figure 2: Systemübersicht

3 Anforderungsanalyse

Um das Nutzungsverhalten der verschiedenen Systeme zu erruieren, wurde eine Umfrage unter den Mitarbeitern gemacht.

Es ging darum, einen Einblick in folgende Fragestellungen zu erhalten:

- Welche Systeme werden wie oft verwendet?
- Welche Systeme protokollieren die Arbeit der Mitarbeiter am besten?
- Gibt es Unterschiede zwischen verschiedenen Rollen oder Mitarbeiter im internen und Mitarbeiter im externen Einsatz?
- Wird die Zeiterfassung regelmässig gemacht oder unregelmässig?
- Wo liegen die Probleme der bestehenden Lösung?
- Welche Smartphone-Betriebsysteme werden verwendet?
- Würde eine Smartphone-App auf Anklang stossen?

3.1 Resultate

Auf eine detailierte Auswertung wurde verzichtet, allerdings wurden folgende Beobachtungen gemacht:

- Google Kalender und Email wird von jedem Mitarbeiter häufig verwendet, egal ob im externen Einsatz oder nicht.
- Github und Redmine werden ebenfalls sehr häufig verwendet, einen signifikanten Unterschied zwischen internen oder externen Einsatz wurde nicht festgestellt, allerdings waren nur sehr wenige Beantwortungen von externen Mitarbeitern eingegangen.
- Die Mitarbeiter schätzen, dass Redmine, Github, Email und Kalender ihre tägliche Arbeit am besten protokollieren, wobei der Kalender leicht vorne liegt.
- Die Zeiterfassung wird häufig am Abend nach einem Arbeitstag gemacht.
- Die Zeiteinträge werden generell nicht allzu detailiert kommentiert.
- Viele Mitarbeiter arbeiten an mehreren Projekten täglich, manche aber auch nur an einem pro Tag. Bei dieser Frage, die mit einer Bewertung von 1-5 beantwortet werden musste, gab es zwei Häufungen mit wenig Beantwortungen im Mittelfeld. Offenbar gibt es zwei Gruppen von Mitarbeiter: Mitarbeiter die häufig nur an einem Projekt arbeiten und Mitarbeiter, welche häufig die Projekte wechseln.
- Der Frage, ob eine Mobile-App von den Mitarbeitern genutzt werden würde, wurde von einer Mehrheit mit Ja beantwortet, allerdings haben immerhin 30% der Mitarbeitern die Frage verneint. In den Kommentaren hat sich dann gezeigt, dass diese eine Lösung bevorzugen würden, die auch am Computer funktioniert und nicht an ein Smartphone gebunden ist.
- Android und IOS wird von ähnlich vielen Mitarbeitern genutzt, wobei Android etwas vorne liegt. Andere Systeme kommen nicht zum Einsatz

4 Konzept

4.1 Begriffe

Event Als "Event" wird in der Arbeit ein Ereignis mit einem bestimmten Zeitpunkt, sowie Beschreibung und anderen Daten angesehen. Die "Events" der zu verwendenen Quellen, haben in der Regel keine Zeitdauer und damit keine Start- und Endzeit, mit Ausnahme der Daten aus dem Kalender. Ein Event kann im einfachen Fall ein Kalendereintrag sein. Es kann aber auch ein Commit auf Github sein, ein abgeschlossener Task auf Redmine oder ähnliches. Ein Event kann man damit auch als "Spur" (engl. "Trace") verstehen, die die tägliche Arbeit hinterlässt.

Timeentry Unter "Timeentry"[2] wird ein zu erstellender Zeiteintrag auf der Zeiterfassungsapplikation (Controllr) verstanden. Im Gegensatz zu einem "Event", verfügt ein "Timeentry" immer über eine Start- und eine Endzeit. Zudem wird einem "Timeentry" einem Projekt, einem Task und einem Datum zugewiesen und mit Beschreibung und weiteren Metadaten versehen. Ein "Timeentry" entspricht also dem Modell eines geleisteten Stücks Arbeit, welches es zu protokollieren gilt.

4.2 Vom Event zum Timeentry

Listet man alle gesammelten "Events" eines Tages nacheinander auf, erhält man ein erstes Protokoll. Um daraus konkrete Zeiteinträge zu erstellen, müssen die Events noch umgeformt und mit weiteren Daten versehen werden:

4.2.1 Projekt und Task-zuweisung

Jedem Timeentry auf dem Controllr muss ein Projekt und ein Task zugewiesen werden. Die Liste der Projekte und Tasks können von einer Schnittstelle abgerufen werden. Jedes Projekt und jeder Task verfügt über einen Namen und eine Beschreibung. Um für ein Event ein Task und ein Projekt zu bestimmen, ergeben sich folgende Möglichkeiten:

- Der Beschreibungstext und andere Metadaten des Events werden nach Projektnamen durchsucht und das entsprechende Projekt bei einem Treffer dem Event zugewiesen
- Github-Events entsprechen i.d.R. einem "Development"-Task
- Kalendereinträge entsprechen einem "Meeting"-Task
- Der Enduser kann selbst Regeln bestimmen, beispielsweise indem er Schlüsselwörter definiert, welche ein bestimmtes Projekt oder einen bestimmten Task forcieren.

4.2.2 Fehlende Start- und Endzeit

Die Events aus den meisten Quellen besitzen aber nur einen Zeitpunkt, keinen Start- und Endzeitpunkt. Es müssen also gewisse Annahmen getroffen werden, um Start- und Endpunkt zu bestimmen. Folgende Annahmen ergeben sich:

- Der Zeitpunkt jedes Events entspricht dem Zeitpunkt, an dem eine Arbeit **beendet wurde**.
  - In Hinblick auf Github-Commits trifft dies zu, da man i.d.R. nach einer Programmieraufgabe diese Änderungen in die Versionskontrolle speichert.

[2] Einfache transkription aus dem Englischen von "Zeiteintrag" in Anlehnung an "Entry", dem Datenmodel, welches in der Zeiterfassungsapplikation "Controllr" verwendet wird

  - Bei Redmine wird man häufig nach getaner Arbeit einen Task als Erledigt markieren oder den Progress des Tasks ändern

- Nach einem abgeschlossenen Task wendet man sich dem nächsten Task zu. Die **Startzeit eines Events** entspricht daher der **Endzeit des letzten Events**
- Die **Startzeit des ersten Events an einem Tag** ist der übliche Arbeitsbeginn eines Mitarbeiters.

4.2.2.1 Probleme: Kalendereinträge verfügen immer über eine Startzeit, wenn es sich nicht um Tagesevents handelt. Sie können sich daher mit Einzel-Events aus anderen Quellen überlappen. Ebenfalls ist es möglich in Kalendern überlappende Einträge zu erstellen.

Mehrere Events können sehr nahe aneinander liegen. Ein mögliches Szenario wäre, wenn ein Benutzer einen Github-Commit erstellt und danach den entsprechenden Task in Redmine aktualisiert. In diesem Falle würden beide Events sogar die gleiche Arbeit protokollieren.

Um diese Probleme zu umgehen, können mehrere Events miteinander kombiniert ("Merging") werden.

4.2.3 "Merging" - Kombinieren von Events

Nah aneinanderliegende Events oder überlappende Events können miteinander kombiniert werden und als ein Event gezählt werden. Dies vereinfacht die Handhabung und die Darstellung von Events. Events dürfen aber nur kombiniert werden, wenn sie auch zum gleichen Projekt gehören. Es ergeben sich daher folgende mögliche Regeln:

- Zwei Events können kombiniert werden, wenn sie mutasslich zum gleichen Projekt gehören.
- Das Ende des zweiten Events (d.h. mit späterer End-Zeit) liegt nahe am Ende des ersten Events (Es ist ein Grenzwert zu definieren)
- Die Gesamtlänge der kombinierten Events sollte einen weiteren Grenzwert nicht überschreiten
- Die Startzeit des zweiten Events liegt vor der Endzeit des ersten Events (Überlappung)

4.3 Mögliche Darstellungen

4.3.1 Kalender-Darstellung

Abbildung 3 zeigt eine übliche Darstellung einer Kalender-Applikationen. Dabei wird jeder Tag als Spalte dargestellt mit dem Begin des Tages oben und das Ende unten.

Diese Darstellung eignet sich auch für die geplante Zeiterfassungsanwendung. "Events" eines Tages können wie die Ereignisse in einem normalen Kalender dargestellt werden.

Vorteile:

- Sie zeigt die relativen Unterschiede der Zeiten durch proportional unterschiedliche Höhen der einzelnen Blöcke
- Sie kann je nach Spaltenbreite mehrere Tage gleichzeitig darstellen
- Sie ermöglicht dem Betrachter qualitativ festzustellen, wieviel Aktivität an einem Tag stattfand

Nachteile:

- Kurze Events können sehr klein werden - und dadurch auch schwer klickbar

Figure 3: Beispiel Kalender Applikation("Our 10 Favorite Calendar Apps to Keep Your Schedule in Check")

- Je nach Breite ist nur sehr wenig Platz vorhanden für Beschriftungen oder Details auf den einzelnen Einträgen, bei kurzen Events ist dies noch akuter.
- Die Umsetzung einer solchen Darstellung ist vergleichsweise aufwendig

4.3.2 Listen-Darstellung

(. . .)

4.4 Datenquellen für den Prototyp

Für den Prototyp wurde folgende Systeme zur Anbindung gewählt:

Google Kalender, Redmine (Issues), Github (Events). Obwohl nach der Umfrage Emails ebenfalls sehr häufig genutzt werden, wurde auf die Abfrage von Emails verzichtet, da diese schwieriger auszuwerten sind.

5 Umsetzung Prototyp

5.1 Technologie-Wahl

Für die Umsetzung wurde zwischen nativer Entwicklung auf einer mobilen Plattform wie IOS, Android oder Windows Phone und zwischen einer Webapplikation entschieden, welche plattformunabhängig läuft.

Da die Umfrage unter den Mitarbeitern ergeben hat, dass beide Plattformen signifikant vertreten sind, fiel die Wahl auf eine Webapplikation. Da manche Mitarbeiter eine dedizierte Smartphone-Applikation in der Umfrage abgelehnt haben, ist eine Webapplikation ebenfalls zu bevorzugen, da sie auch am Desktop-Computer, Notebook, Tablet oder ähnlichem benutzt werden kann.

5.2 Meteor

Die Webapplikation wurde unter Meteor entwickelt, einem "Full-Stack"-Webframework[3] in Javascript. Meteor basiert auf einem reaktiven Programmierparadigma und löst Client-Server-Kommunikation und Data-Binding[4] auf eine einfache Weise, was den Code sehr expressiv macht und der Fokus auf die Umsetzung des zu lösenden Problem gesetzt werden kann. Für eine Konzeptarbeitet hat dies entscheidende Vorteile.

Im September 2014 wurde das Build-Tool von Meteor erweitert, sodass auf eine einfache Weise Phonegap/Cordova-Container-Applikationen erstellt werden können. Diese Art von Applikation bündelt eine Webapplikation in einer nativen Applikation und kann in den Apple App Store, im Google Play-Store oder den Windows Store gestellt werden. Auf der jeweiligen Plattform erscheint sie als normale Anwendung. In Anbetracht des Themas "Entwicklung für Handheld" ein zusätzlicher, sinnvoller Exkurs. ("Building IOS and Android Mobile Apps with PhoneGap")

5.3 Sprache

Meteor wird in JavaScript geschrieben, unterstützt aber auch diverse JavaScript-Dialekte, wovon CoffeeScript für die Umsetzung gewählt wurde. Ausschlaggebend dafür war die schlankere Syntax, die bessere Lesbarkeit, sowie syntaktische Erweiterungen und Abkürzungen ("Syntactic sugar"). ("Meteor Coffeescript"; "CoffeeScript (Offizielle Seite)")

5.4 Authentifizierung

Meteor stellt ein einfaches Login-System über OAuth zur Verfügung, es können Google, Facebook, Github und weitere als Login-Provider definiert werden. Es wurde Google als Login-Provider gewählt, damit können auch die Firmen-Logins verwendet werden. Ist die Authentifizierung auf Google erfolgt, können verschiedene Google-APIs abgerufen werden, im Speziellen die Kalender-API, welche hier verwendet wurde.

Für die Authentifizierung gegenüber Github, Redmine und der Controllr-Applikation muss allerdings ein "Access-Token" in der jeweiligen Anwendung erstellt und einmalig in der Anwendung dieser Arbeit gespeichert werden. Für diesen Zweck und weitere Einstellungen wurde eine Einstellungs-Seite erstellt, wo der User diese Tokens und weitere Einstellungen abspeichern kann. Siehe Abbildung 4

[3] "Full-Stack"-Frameworks decken alle Schichten einer typischen Webapplikation ab, d.h. vom Client bis zum Server. Häufig können sie dadurch die Datenverbindung vom Client zum Server abstrahieren.

[4] Data-Binding bezeichnet das Verfahren, das User Interface (UI) einer Applikation derart an deren Business Logic (BL) zu koppeln, das Änderungen an der BL auf das UI reflektiert werden und umgekehrt

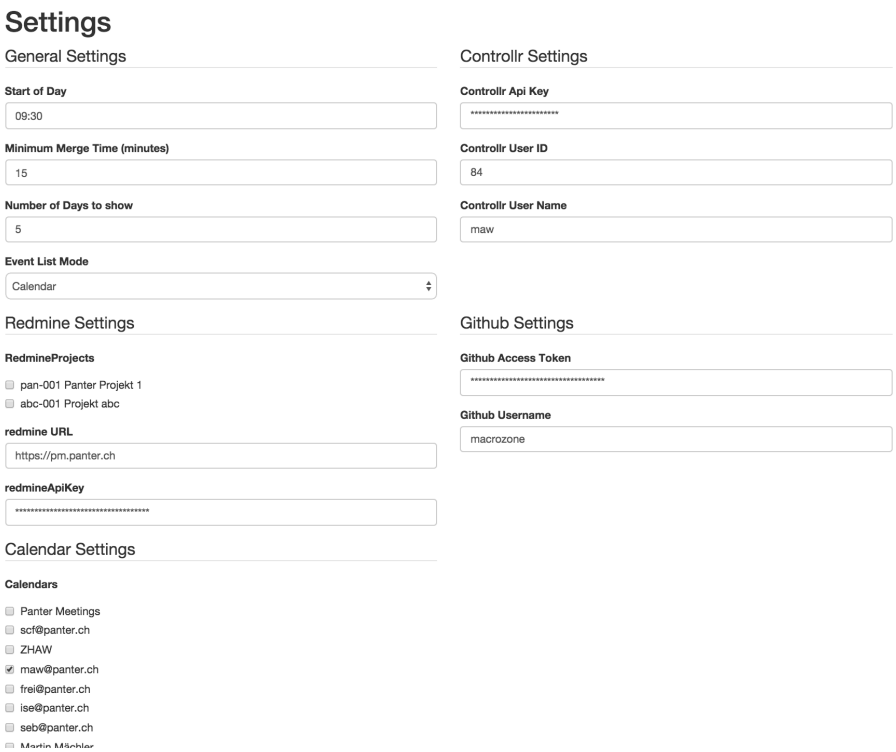

Figure 4: Einstellungs-Bildschrim der umgesetzten Anwendung

5.5 Reactive-Programming und REST-Schnittstellen: "Reactive REST-Mapping"

Meteor implementiert das Programmierparadigma der "Reaktiven Programmierung", dabei werden Änderungen der Datenquellen, welche die Applikation nutzt, automatisch propagiert und beispielsweise Darstellungen dieser Datenquellen aktualisiert. ("Reactive Programming - Wikipedia"; "Meteor Manual - Transparent Reactive Programming")

Bei der Umsetzung mussten diverse REST-Apis angesprochen werden. Es entstand das Bedürfnis, die Zugriffe auf diese REST-APIs derart zu abstrahieren, dass auf dem Client mit normalen Meteor-Collections und -Subscriptions gearbeitet und dadurch das "Reactive Programming"-Modell von Meteor beibehalten werden kann. Dadurch entstand das "Smart-Package"[5] `panter:publish-array`, welches in einer initialen Version auf den Meteor-Paket-Manager gestellt wurde. (TODO: add github source)

Dieses Verfahren wird nachfolgend "Reactive REST-Mapping" genannt.

`panter:publish-array` "publiziert" ein beliebiges Javascript-Array aus Objekten auf dem Server als Meteor-Collection auf dem Client. Dabei muss eine Funktion angegeben werden, welches dieses Array zurückgibt. In dieser Funktion kann beispielsweise eine REST-API aufgerufen werden, welche dieses Array zurückgibt. Es kann eine Intervall-Zeit angegeben werden, wodurch die definierte Funktion periodisch aufgerufen wird, solange der Client sich auf dieses Publikation eingeschrieben ("subscribed") hat.

Durch dieses Verfahren können die Daten aus den verschiedenen Quellen auf dem Server der Anwendung

[5] So werden Pakete von Meteor genannt. Pakete können von einer zentralen Registrierungsstelle referenziert werden und Abhängkeiten werden automatisch aufgelöst.

vorkonsolidiert und vorbearbeitet werden. Die Daten der verschiedenen "Event"-Quellen werden beispielsweise in eine Collection "Events" auf dem Client publiziert. Auf dem Client kann so das normale Data-Binding von Meteor verwendet werden, die Quellen der Events sind dabei bereits abstrahiert und transparent. Neue Quellen können somit einfach implementiert werden, der Code für das Client-UI muss nicht angepasst werden.

5.6 Verwendete Event-Quellen

5.6.1 Google Kalendar

Meteor verfügt über ein Login-System, welche es ermöglicht, sich gegenüber Google, Facebook oder weiteren Login-Providern zu authentifizieren. Wählt man Google als Login-Provider, können auch deren Schnittstellen abgefragt werden, sofern der Benutzer sein Einverständnis gibt. ("Meteor Accounts")

Für Meteor existiert ein Paket `percolate:google-api`, welches den Zugriff auf die REST-APIs von Google erleichtert. Mit Hilfe dieses Paketes wurden einerseits die Liste der abonnierten Kalender abgefragt, sowie die Events der gewählten Kalender. ("Meteor Google API")

5.6.2 Beispiel für `panter:publish-array` anhand von Google Kalender

Listing 7 zeigt ein Beispiel, wie mit dem erstellen Paket `panter:publish-array`, sowie `percolate:google-api` die Liste der abonnierten Kalender abgefragt und an den Client publiziert wird. (`handleIds` benennt die Feldnamen der IDs um, sodass Meteor-kompatible ID-Felder zurückgegeben werden (`_id`))

```
1  Meteor.publishArray
2      name: "calendarList"
3      collection: "Calendars"
4      refreshTime: 10000
5      data: (params) ->
6          user =  Meteor.users.findOne _id: @userId
7          if user?
8              url = "calendar/v3/users/me/calendarList"
9              result = GoogleApi.get url, user: user
10             return handleIds result.items
```

Listing 7: Publizieren der Google Kalender mit Hilfe von 'panter:publish-array' in CoffeeScript

Abonniert der Client nun dieses Topic `calendarList` mit `Meteor.subscribe("calendarList")`, so steht dem Client eine Collection `Calendars` zur Verfügung, wo alle vom User abonierten Kalender enthalten sind. Die Collection aktualisiert sich alle 10 Sekunden, solange der User dieses Topic abonniert hat. Abbildung 5 zeigt ein Beispiel, wie es nun möglich, diese Collection an ein Template zu binden um eine Liste aller Kalender anzuzeigen. Diese Ansicht wird genutzt, um die Kalender auszuwählen, welche als Datenquellen verwendet werden sollen.

Listing 8 zeigt das Spacebars[6]-Template des Einstellung-Bildschirms. Dabei wurde das Paket `aldeed:autoform` verwendet, welches es ermöglicht ausgehend von Schema-Definitionen automatisch Formulare zu erzeugen. ("AutoForm Github")

```
1  {{#autoForm collection="UserSettingsStore" doc=settings id="settingForm" type="update" autosave=tr
2    (...)
3    {{> afQuickField name='calendars' options=calendars noselect=true}}
4    (...)
```

[6]Standard-Template-Sprache von Meteor, angelehnt an "Handlebars" (http://handlebarsjs.com/)

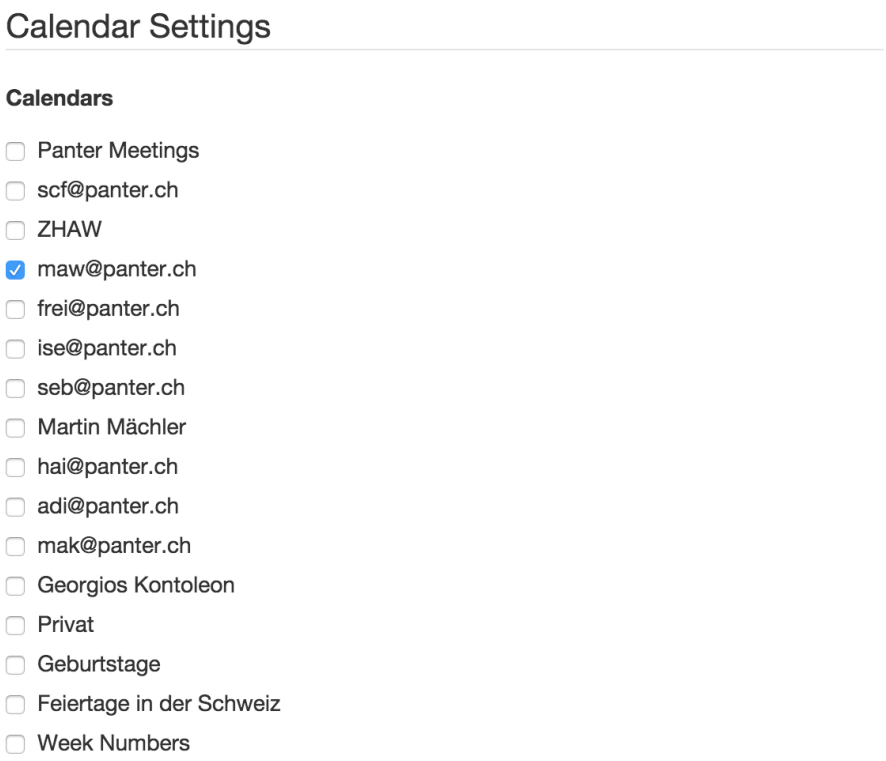

Figure 5: Kalender-Wahl-Bildschrim der umgesetzten Anwendung

```
5  {{/autoForm}}
```

Listing 8: Spacebars-Template zur Abbildung 5

In Listing 9 wird ein Teil der Routen-Konfiguration des Einstellungsbildschirms gezeigt. Die Daten-Funktion `data` gibt dabei unter anderem ein `calendars`-Feld zurück, welches im Template als Optionen für das Formularfeld verwendet wird.

```
1  Router.route 'settings',
2      waitOn: share.defaultSubscriptions
3      data: ->
4              (...)
5          calendars: -> Calendars.find().map (calendar) ->
6              label: calendar.summary
7              value: calendar._id
8          (...)
```

Listing 9: Routen-Konfiguration für den Einstellungsbildschirm auf Abbildung 5 in CoffeScript

Abbonniert nun ein Benutzer einen neuen Kalender auf Google Apps, so wird innerhalb des Aktualisierungs-Intervals von `panter:publish-array` der neue Kalender geladen und an den Client publiziert. Auf dem Client wird nun der Einstellungsbildschirm automatisch um den neuen Kalender ergänzt.

Auf eine ähnliche Art werden die Ereignisse vom Kalender abgefragt und als “Events” publiziert.

5.6.3 Redmine

Von Redmine wird die Liste der Projekte geladen, von welchen analog zur Kalenderliste bestimmte Projekte gewählt werden können, von denen die Stories und Tasks geladen werden.

Über die issues-Schnittstelle werden die Stories und Tasks geladen, an denen der Benutzer gearbeitet hat.

5.6.4 Github

Über die “Events”-Schnittstelle von Github wurden die Aktivitäten des Users auf Github geladen und als “Events” publiziert.

5.7 Vorteile von “Reactive REST-Mapping”

Das Verfahren des “Reactive REST-Mapping” wurde über die ganze Anwendung hinweg gebraucht und erleichterte den Umgang mit den REST-Schnittstellen sehr. So werden Projekte aus Redmine, “Issues” aus Redmine, “Events” aus Github, sowie “Events” aus dem Google Kalender und “TimeEntries” aus “Controllr” periodisch geladen und alle Ansichten, welche diese Daten nutzen automatisch aktualisiert.

Ein weiterer Vorteil dieses Verfahren ist, dass die Datenquelle für den Client völlig transparent ist, der Client “sieht” nur gewöhnliche Meteor-Collections. So lässt sich die Datenquelle oder die Anbindungs-Technologie einfach austauschen um beispielsweise von einem “Polling”[7] auf ein Nachrichten-basiertes System zu wechseln, wie es bereits zwischen Client und Server existiert. Damit entfällt die Verzögerung, die durch das Polling-Interval entsteht.

[7] Die Datenquelle wird periodisch abgefragt.

5.8 Event-Darstellung

5.9 Schwierigkeiten

Quellenangaben

"AutoForm Github." https://github.com/aldeed/meteor-autoform.

"Building IOS and Android Mobile Apps with PhoneGap." https://www.meteor.com/blog/2014/09/15/meteor-092-iOS-Android-mobile-apps-phonegap-cordova.

"CoffeeScript (Offizielle Seite)." http://coffeescript.org/.

"Github Authentifizierung." https://developer.github.com/v3/#authentication.

"Google Apps Email API." https://developers.google.com/gmail/api/v1/reference/.

"Google Kalender API." https://developers.google.com/google-apps/calendar/v3/reference/events#resource.

"Google OAuth2." https://developers.google.com/accounts/docs/OAuth2.

"Meteor Accounts." https://www.meteor.com/accounts.

"Meteor Coffeescript." http://docs.meteor.com/#/full/coffeescript.

"Meteor Google API." https://github.com/percolatestudio/meteor-google-api.

"Meteor Manual - Transparent Reactive Programming." http://manual.meteor.com/#deps-transparentreactiveprogramming.

"Our 10 Favorite Calendar Apps to Keep Your Schedule in Check." http://www.brit.co/10-mobile-calendar-apps/.

"Reactive Programming - Wikipedia." http://en.wikipedia.org/wiki/Reactive_programming.

"Redmine REST API." http://www.redmine.org/projects/redmine/wiki/Rest_api.