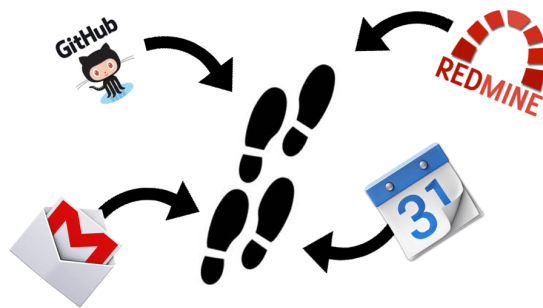


TimeTraces

Seminararbeit "Entwickeln von Anwendungen für Hand Held"

Marco Wettstein

2015-03-02



Inhaltsverzeichnis

1	Vorwort	4
2	Ist-Analyse	5
2.1	Ausgangslage	5
2.2	Rollen	5
2.3	Systeme	6
2.3.1	Controllr	6
2.3.2	Google Apps	9
2.3.3	Redmine	9
2.3.4	GitHub	9
2.3.5	Timetunnel	10
2.3.6	strms.io (vormals storyline.li)	10
2.3.7	Systemübersicht	10
3	Anforderungsanalyse	12
3.1	Resultate	12
4	Konzept	13
4.1	Grundlegende Idee	13
4.2	Begriffe	13
4.3	Vom Event zum Timeentry	13
4.3.1	Projekt und Task-zuweisung	13
4.3.2	Fehlende Start- und Endzeit	13
4.3.3	“Merging” - Kombinieren von Events	14
4.4	Mögliche Darstellungen	15
4.4.1	Kalender-Darstellung	15
4.4.2	Listen-Darstellung	15
4.5	Datenquellen für den Prototyp	16
5	Umsetzung Prototyp	17
5.1	Technologie-Wahl	17
5.2	Meteor	17
5.3	Sprache	17
5.4	Authentifizierung	17
5.5	Reactive-Programming und REST-Schnittstellen: “Reactive REST-Mapping”	18
5.5.1	Funktionsweise	18
5.6	Verwendete Event-Quellen	20

5.6.1	Google Kalender	20
5.6.2	Beispiel für <code>panter:publish-array</code> anhand von Google Kalender	20
5.6.3	Redmine	21
5.6.4	Github	22
5.7	Event-Darstellung und Zeiterfassung	22
6	Ausblick und Diskussion	24
6.1	Vorteile von "Reactive REST-Mapping"	24
6.2	Probleme und Erweiterungen zum "Reactive REST-Mapping"	24
6.3	Schwierigkeiten bei der Darstellung	24
A	Anhang	26
A.1	Quellcode	26
A.2	Quellenangaben	26

Abbildungsverzeichnis

1	Screenshot von "Controllr"	6
2	Systemübersicht	11
3	Beispiel Kalender Applikation Quelle: ("Our 10 Favorite Calendar Apps to Keep Your Schedule in Check")	15
4	Einstellungs-Bildschirm der umgesetzten Anwendung. Auf kleineren Bildschirmen werden alle Felder untereinander dargestellt.	18
5	Sequenzdiagramm der Interaktion zwischen Client, Server und Datenquellen	19
6	Kalender-Wahl-Bildschirm der umgesetzten Anwendung	21
7	Umgesetzte Darstellung der Event-Liste eines Tages	22
8	Eingabemaske für einen Zeiteintrag. Alle Felder werden vorausgefüllt	23

Listings

1	Controllr REST API für Zeiteinträge	6
2	Schema von <code>/api/entries</code>	7
3	Controllr REST API für Projekte	7
4	Schema von <code>/api/projects</code>	7
5	Controllr REST API für Tasks	8
6	Schema von <code>/api/tasks</code>	8
7	Publizieren der Google Kalender mit Hilfe von <code>'panter:publish-array'</code> in CoffeeScript	20
8	Spacebars-Template zur Abbildung 6	20
9	Routen-Konfiguration für den Einstellungsbildschirm auf Abbildung 6 in CoffeeScript	21

1 Vorwort

Sehr geehrte Leserschaft

Smartphones und andere mobilen Geräte haben nicht nur durch ihre Mobilität unseren Alltag und die Arbeitswelt erobert, sondern auch durch neue Bedienkonzepte und durch Bündelung verschiedener Datenquellen und Dienste. Kalender und GPS verbinden Ort und Zeit des Benutzers - das Gerät weiss jederzeit, was der Benutzer gerade tut oder geplant hat und wo er sich befindet und kann daraus Absichten des Benutzers vorhersehen. Konzepte wie Google Now verfolgen diesen Ansatz¹. Das Smartphone wird vermehrt zum intelligenten digitalen Assistenten.

Mit dieser Vision versuchte ich die firmeninterne Anwendung zur Zeiterfassung ("controllr") der Firma Panter AG für Smartphones neu zu konzipieren.

Die bestehende Anwendung funktioniert prinzipiell auch auf mobilen Endgeräten mit kleinen Bildschirmen mittels einfachen Verfahren des Responsive Webdesign², doch reicht das reine Umordnen der Elemente in diesem Fall nicht, um den Prozess der Zeiterfassung sinnvoll auf ein Smartphone zu übertragen. Denn nicht nur die Grösse des Bildschirms hat einfluss auf die Bedienbarkeit, sondern auch die Eingabemöglichkeiten des Gerätes. Schaltflächen sind mit einem berührungssensitiven Bildschirms schwerer zu treffen und müssen entsprechend gestaltet werden, ebenso ist die Eingabe von Text auf einem Smartphone aufwendiger und daher langsamer im Vergleich zu einem klassischen Computer mit Maus und Tastatur.

Um dieses Problem anzugehen überlegte ich mir daher zwei Stossrichtungen:

- Bedienelemente der Zeiterfassung für Smartphones optimieren
- Die Menge der Bedienelemente und Eingaben reduzieren

Während die erste Option in der Regel einfacher umzusetzen ist und auch häufig gemacht wird, empfinde ich doch die zweite Variante als weitaus spannender und zeitgemässer.

Mit der Eingangs erwähnten Vision im Hinterkopf überlegte ich mir daher, wie ich die Anzahl Benutzerinteraktionen für eine Zeiterfassung minimieren kann, indem ich verschiedene Datenquellen eines Mitarbeiters miteinander verbinde. Im Idealfall würde ein Benutzer die Einträge lediglich noch bestätigen müssen.

Die Applikation würde eine Art "Spurensuche" machen, nach den Spuren, die wir bei der täglichen Arbeit hinterlassen. Ausgehend aus dieser Idee entstand auch der Projekt-Name: "TimeTraces".

¹Siehe Quelle ("Google Now")

²Prinzipiell werden beim Responsive Webdesign Elemente auf einer Webseite so angeordnet und allenfalls in ihrer Grösse angepasst, sodass sie auf verschiedenen Bildschirmgrössen und -formate sinnvoll Platz haben. Quelle ("Responsive Webdesign - Wikipedia")

2 Ist-Analyse

2.1 Ausgangslage

Jeder Mitarbeiter der Firma Panter AG trägt regelmässig seine gearbeitete Zeit in der Anwendung "Controllr" ein. Dabei wird unter anderem die gearbeitete Zeit, das zugehörige Projekt, ein Task-Typ und eine Beschreibung angegeben. Diese Tasks müssen am Ende eines Monats bestätigt werden, damit eine Auswertung stattfinden kann.

Die Anwendung ist als Webbasierte Lösung implementiert und ist für die Benutzung am Computer ausgerichtet, funktioniert prinzipiell aber auch auf kleineren Smartphones und Tablet-Computer. Dabei wurden die Elemente bei wenig Platz untereinander angeordnet. Die Eingabe-Elemente bleiben unangetastet.

Die Erfassung eines Zeiteintrages erfordert folgende Interaktionen:

- Auswahl eines Tages durch klick auf einen kleinen Kalender.
- Auswahl eines Projektes durch ein *Select*-Element
- Auswahl eines Tasks durch ein *Select*-Element
- Eingabe der Startzeit im Format hh:mm
- Eingabe der Endzeit im Format hh:mm
- Eingabe eines Beschreibungstextes
- Bestätigung durch Klick auf "Create Entry"

Weitere Einzelheiten sind unter Abschnitt 2.3.1 zu finden.

Jedem Mitarbeiter stehen weitere Systeme zur Verfügung, welche er in seiner täglichen Arbeit benutzen kann (Siehe Abschnitt 2.3). Vieler dieser Systeme können über Schnittstellen ausgewertet werden um beispielsweise die Aktivität eines Benutzers auszuwerten. Dadurch entsteht eine Art "Protokoll" des Benutzers und seiner Arbeit.

2.2 Rollen

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

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

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

Administration Kümmt sich um Buchhaltung, Administration 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 Kümmt sich um die Verwaltung des Cowork-Space “colab-zurich.ch”. Zur Zeit (Ende 2014) eine Praktikumsstelle.

2.3 Systeme

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

2.3.1 Controllr

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

The screenshot displays the 'Daily entries' form and a table of 'Entries for 10 Feb 2015'.

Daily entries form:

- Calendar:** February 2015. The 10th is highlighted.
- Project:** abc-123 - Project ABC
- Task:** Development
- Description:** Implement new Feature X
- Start:** 16:35, **End:** 17:45, **Duration:** (auto-calculated)
- Billable:** ☒ Billable
- Create Entry:** (button)

Entries for 10 Feb 2015 table:

Project	Project desc.	Task	Description	Start	End	Duration	Billable	Edit	Copy	Delete
abc-123	Projekt ABC	Internal Meeting	Kickoff Meeting	09:15	10:45	1.50	✓			
xyz-001	Projekt XYZ	Development	Refactor I18n-Module	11:45	12:15	0.50	✓			
xyz-002	Project XYZ Support	Development	support user	13:05	13:20	0.25	✓			
abc-123	Project ABC	Development	Implement that feature, solve a...	13:20	16:20	3.00	✓			
xyz-001	Project XYZ	Internal Meeting	Code Review	16:25	16:35	0.17	✓			
Incurred						5.42				

Showing 1 to 5 of 5 entries

Abbildung 1: Screenshot von “Controllr”

2.3.1.1 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

```

1 {
2   "id": 12513,
3   "created_at": "2014-09-15T14:25:07.000Z",
4   "updated_at": "2014-11-13T14:45:31.000Z",
5   "deleted_at": null,
6   "day": "2014-09-15",
7   "start": "2000-01-01T09:50:00Z",
8   "end": "2000-01-01T10:00:00Z",
9   "duration": 10,
10  "state": "invoiced",
11  "description": "Standup Meeting",
12  "billable": true,
13  "invoice_id": 123,
14  "project_id": 1100,
15  "task_id": 10042,
16  "user_id": 12,
17  "project_shortcode": "abc-001",
18  "task_name": "Internal Meeting",
19  "user_username": "maw"
20 }

```

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 einzelnen Zeiteinträge geben, die über mehrere Tage gehen (z.b. über Mitternacht).

Manche relationale Daten sind zudem Denormalisiert (project_shortcode und task_name).³

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

2.3.1.2 Projekte Projekte können über die Schnittstellen von Listing 3 abgerufen werden und liefern ein Array von Projekten wie im Schema 4 beschrieben. Relevant sind id eines Projektes, das Kürzel (shortcode), sowie der Beschreibungstext description.

```

1 GET      /api/projects(.:format)
2 GET      /api/projects/:id(.:format)

```

Listing 3: Controllr REST API für Projekte

```

1 {
2   "id": 1234,
3   "shortcode": "abc-001",
4   "description": "Beispiel Projekt",
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",

```

³Als 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_shortcode mitgegeben, welcher direkt abhängig von der project_id ist. Das Denormalisieren entspricht der Umkehrung der Normalisierung.

```

9     "project_state_id": 6,
10    "probability": "1.0",
11    "deleted_at": null,
12    "external": true,
13    "note": "Dies ist ein Beispielpjekt",
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": "Gyqrt64o8A0ywywBGyqrt64o8A0ywywB",
27    "planning_note": "",
28    "time_billable": "1255:43",
29    "burned_time": "1301:47"
30 }

```

Listing 4: Schema von /api/projects

2.3.1.3 Tasks Die Tasks-Schnittstelle wird in Listings 5 und 6 beschrieben. Relevant sind hier die Felder `id`, `name`, `project_id`, welches auf ein zugehöriges Projekt verweist, sowie `billable_by_default`, welches angibt, ob ein Task standardmässig verrechenbar ist oder nicht.

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

2.3.1.4 Authentifizierung Für die Authentifizierung 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.

2.3.2 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 Schnittstellen in vielen Sprachen bereits existieren, bietet es sich an, manche dieser Schnittstellen zu verwenden.

2.3.2.1 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.⁴

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

2.3.2.2 Kalender Kalender-Anwendung von Google. Wird in der Firma Panter häufig verwendet und kann ebenfalls über eine REST-API abgerufen werden. 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.⁵

2.3.2.3 Authentifizierung Die Authentifizierung wird OAuth 2.0 verwendet. Es existieren zahlreiche Implementierungen dieses Standards, was die Verwendung dieser Schnittstellen vereinfacht.⁶

2.3.3 Redmine

Projektverwaltungs-Anwendung, welche von der Firma Panter 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 Firmeninterne 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.⁷

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

2.3.4 GitHub

Verwaltungsoberfläche und Hosting-Dienst für Software-Projekte, welche die namensgebende Quellcode-Versionenverwaltungs-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.⁸

⁴Quelle ("Google Apps Email API").

⁵Quelle ("Google Kalender API")

⁶Quelle ("Google OAuth2")

⁷Quelle ("Redmine REST API")

⁸Quelle ("GitHub API")

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

2.3.4.1 Authentifizierung GitHub unterstützt verschiedene Authentifizierungsverfahren: Basic Authentication mit Username / Password, OAuth2 mit Token oder OAuth2 mit Key/Secret.⁹

2.3.5 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.

2.3.6 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.

2.3.7 Systemübersicht

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

⁹Siehe ("Github Authentifizierung").

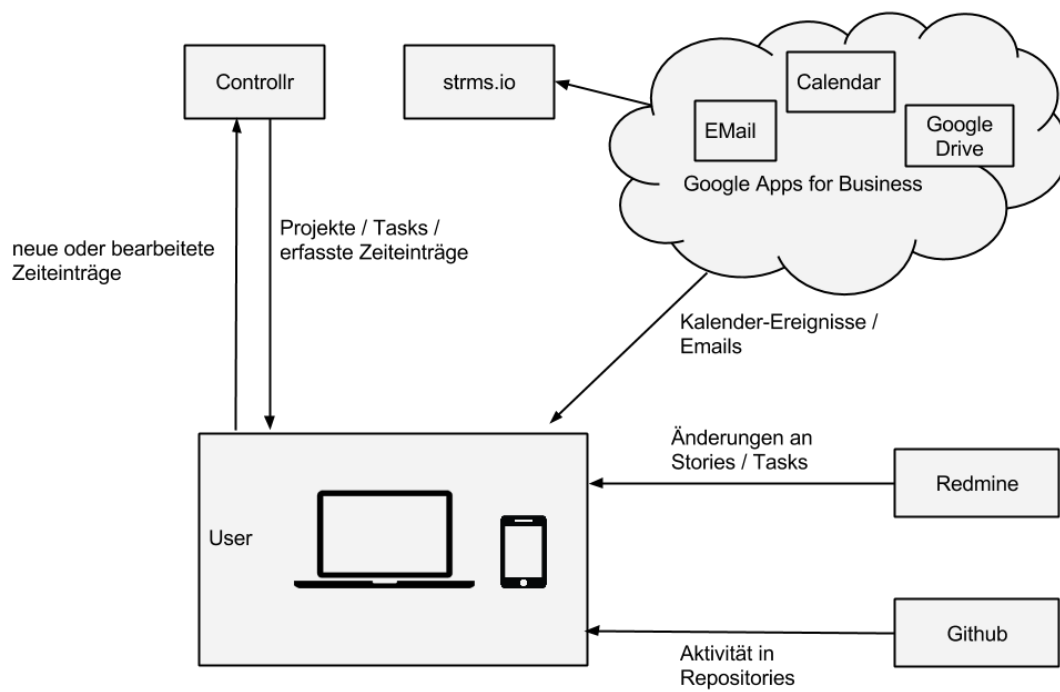


Abbildung 2: Systemübersicht

3 Anforderungsanalyse

Um das Nutzungsverhalten der verschiedenen Systeme zu eruieren, 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-Betriebssysteme werden verwendet?
- Würde eine Smartphone-App auf Anklang stossen?

3.1 Resultate

In der Auswertung der Umfrage 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 detailliert 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 Grundlegende Idee

Ausgehend von verschiedene Datenquellen soll eine Art "Protokoll" über die Aktivität eines Users erstellt werden. Ein einzelner Eintrag dieses Protokoll (ein "Event", wie im Abschnitt 4.2 erwähnt) soll dabei einem zu erfassenden Zeiteintrag entsprechen.

Der Benutzer kann einen solchen Eintrag auswählen um aus diesem einen Zeiteintrag zu erstellen, dabei sollen möglichst viele der Felder, wie Start-, Endzeit, Projekt, Task und Beschreibung vorausgefüllt werden.

4.2 Begriffe

Event Als "Event" wird in der Arbeit ein Ereignis mit einem bestimmten Zeitpunkt, sowie Beschreibung und anderen Daten angesehen. Die "Events" der zu verwendeten 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"¹⁰ wird ein zu erstellender Zeiteintrag auf der Zeiterfassungssaplikation (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.

4.3 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:

4.3.1 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.

4.3.2 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:

¹⁰Einfache transkription aus dem Englischen von "Zeiteintrag" in Anlehnung an "Entry", dem Datenmodel, welches in der Zeiterfassungssaplikation "Controllr" verwendet wird

- 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.
 - 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.

4.3.2.1 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.

4.3.3 "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)

4.4 Mögliche Darstellungen

4.4.1 Kalender-Darstellung

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



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

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
- 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

4.4.2 Listen-Darstellung

Viele Kalender bieten als alternative Darstellung eine Listenansicht an. In dieser Ansicht werden Einträge untereinander aufgelistet ohne dass die Zeitdauer berücksichtigt wird.

Vorteile:

- Jeder Eintrag kann gleich gross dargestellt werden und somit steht für Titel oder Beschreibung des Eintrages genug Platz zur Verfügung.
- Die Ansicht kann sehr kompakt gemacht werden, insbesondere, wenn an einem Tag nur wenige Ereignisse eingetragen sind
- Einfach zu implementieren

Nachteile:

- Für den Betrachter ist nicht direkt ersichtlich, wie lange ein einzelnes Ereignis dauert

4.5 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”-Webframework¹¹ in Javascript. Meteor basiert auf einem reaktiven Programmierparadigma und löst Client-Server-Kommunikation und Data-Binding¹² 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 Konzeptarbeit 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.¹³

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”).¹⁴

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

¹¹“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.

¹²Data-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

¹³Siehe (“Building IOS and Android Mobile Apps with PhoneGap”)

¹⁴Siehe (“Meteor Coffeescript”), sowie (“CoffeeScript (Offizielle Seite)“)

Settings

General Settings

Start of Day
09:30

Minimum Merge Time (minutes)
15

Number of Days to show
5

Event List Mode
Calendar

Redmine Settings

RedmineProjects

- ☐ pan-001 Panter Projekt 1
- ☐ abc-001 Projekt abc

redmine URL
https://pm.panter.ch

redmineApiKey

Calendar Settings

Calendars

- ☐ Panter Meetings
- ☐ scf@panter.ch
- ☐ ZHAW
- ☒ maw@panter.ch
- ☐ frei@panter.ch
- ☐ ise@panter.ch
- ☐ seb@panter.ch
- ☐ Martin Minkler

Controllr Settings

Controllr Api Key

Controllr User ID
84

Controllr User Name
maw

Github Settings

Github Access Token

Github Username
macrozone

Abbildung 4: Einstellungs-Bildschirm der umgesetzten Anwendung. Auf kleineren Bildschirmen werden alle Felder untereinander dargestellt.

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.¹⁵

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” (Abonnements) gearbeitet und dadurch das “Reactive Programming”-Modell von Meteor beibehalten werden kann. Dadurch entstand das “Smart-Package”¹⁶ `panter:publish-array`, welches in einer initialen Version auf den Meteor-Paket-Manager gestellt wurde.¹⁷

Dieses Verfahren wird nachfolgend “Reactive REST-Mapping” genannt.

5.5.1 Funktionsweise

`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. Abbildung 5 zeigt ein Sequenzdiagramm der Interaktion zwischen Client-Teil der App, Server-Teil

¹⁵Siehe Quellen (“Reactive Programming - Wikipedia”; “Meteor Manual - Transparent Reactive Programming”)

¹⁶So werden Pakete von Meteor genannt. Pakete können von einer zentralen Registrierungsstelle referenziert werden und Abhängigkeiten werden automatisch aufgelöst.

¹⁷Source-Code in der Quellenangabe (“Panter:publish-Array Github Repository”)

der App, sowie zwei Datenquellen. Dargestellt sind zwei Poll-Intervalle, bevor der Client die Sitzung beendet. Typischerweise sind es mehr; umgesetzt wurden Intervalle von 10 Sekunden.

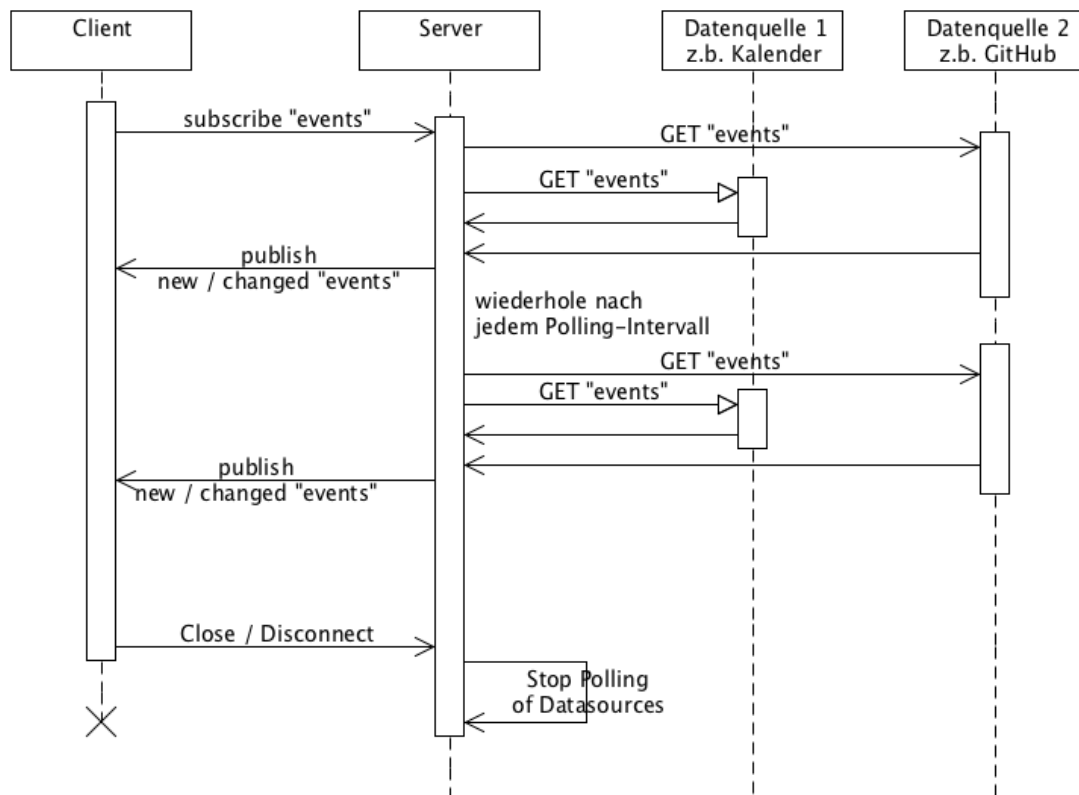


Abbildung 5: Sequenzdiagramm der Interaktion zwischen Client, Server und Datenquellen

Durch dieses Verfahren können die Daten aus den verschiedenen Quellen auf dem Server der Anwendung vorkonsolidiert und vorbearbeitet werden. Die Daten der verschiedenen "Event"-Quellen werden 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 Kalender

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.¹⁸

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.¹⁹

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 6 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 Spacebars²⁰-Template des Einstellung-Bildschirms. Dabei wurde das Paket `aldeed:autoform` verwendet, welches es ermöglicht ausgehend von Schema-Definitionen automatisch Formulare zu erzeugen.²¹

```

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

```

Listing 8: Spacebars-Template zur Abbildung 6

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.

¹⁸Siehe ("Meteor Accounts")

¹⁹Github-Repository unter Quelle ("Meteor Google API")

²⁰Standard-Template-Sprache von Meteor, angelehnt an "Handlebars" (<http://handlebarsjs.com/>)

²¹Github-Repository ist unter Quelle ("AutoForm Github") zu finden.

Calendar Settings

Calendars

- ☐ Panter Meetings
- ☐ scf@panter.ch
- ☐ ZHAW
- ☒ maw@panter.ch
- ☐ frei@panter.ch
- ☐ ise@panter.ch
- ☐ seb@panter.ch
- ☐ Martin Mächler
- ☐ hai@panter.ch
- ☐ adi@panter.ch
- ☐ mak@panter.ch
- ☐ Georgios Kontoleon
- ☐ Privat
- ☐ Geburtstage
- ☐ Feiertage in der Schweiz
- ☐ Week Numbers

Abbildung 6: Kalender-Wahl-Bildschirm der umgesetzten Anwendung

```
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 6 in CoffeeScript

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 und in die Collection “Events” publiziert. Falls der Projektname (Kürzel) im Redmine mit einem Projektnamen von “controllr” übereinstimmt, so wird dieses Projekt vorausgewählt. Der Zugehörige Task ist immer “Development”

5.6.4 Github

Über die “Events”-Schnittstelle von Github wurden die Aktivitäten des Users auf Github geladen und ebenfalls als “Events” publiziert. Analog zu Redmine wird versucht, den Projektnamen zu finden und es wird der Task “Development” zugewiesen.

5.7 Event-Darstellung und Zeiterfassung

Die Ereignisse eines Tages wurden mit einer Kalender-ähnlichen Darstellung aufgelistet. Da der Fokus auf die *letzten* Aktivitäten eines Benutzer gelegt ist, beginnt die Liste mit dem letzten Tag und dem letzten Event, wobei die Ereignisse eines Tages von unten nach oben geordnet sind, im Gegensatz zu klassischen Kalender-Darstellungen, bei denen der Tag oben beginnt und unten endet.

Bereits erfasste Zeiteinträge (“TimeEntries”) wurden ebenfalls in die gleiche Darstellung übernommen. Abbildung 7 zeigt die Auflistung eines Tages, vor (links) und nach dem Erfassen der Zeiteinträge (rechts).

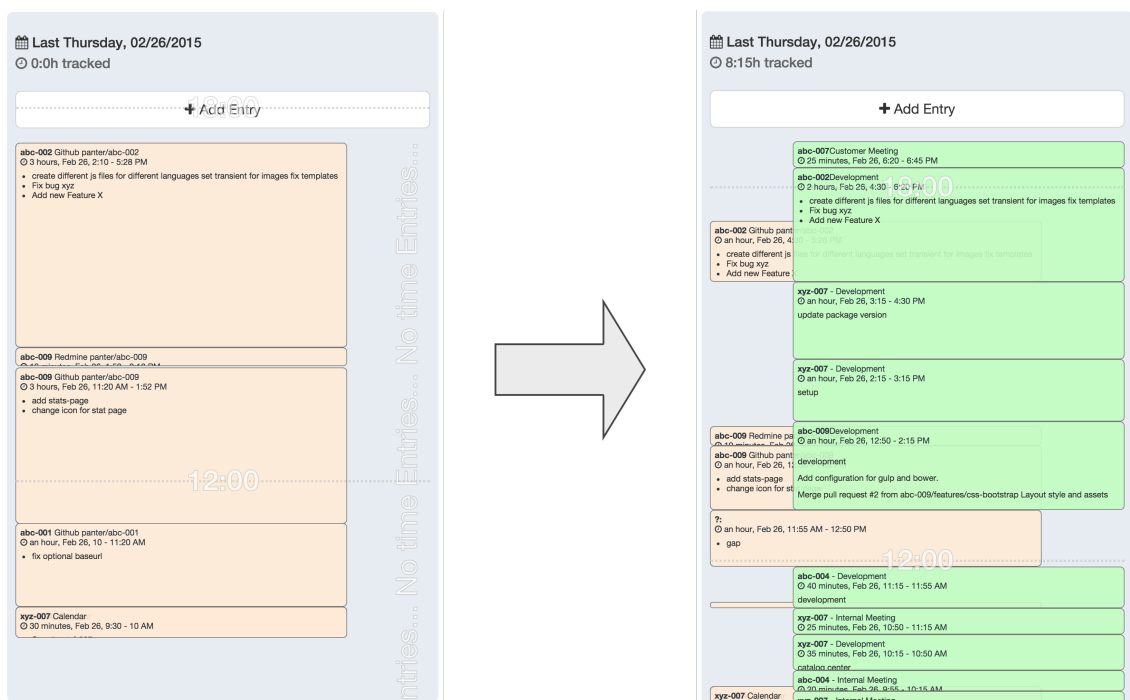


Abbildung 7: Umgesetzte Darstellung der Event-Liste eines Tages

Klickt der Benutzer auf einen roten Ereignis, so erhält er eine Eingabemaske für den Zeiteintrag (Abbildung 8). Dabei werden alle Felder vorausgefüllt:

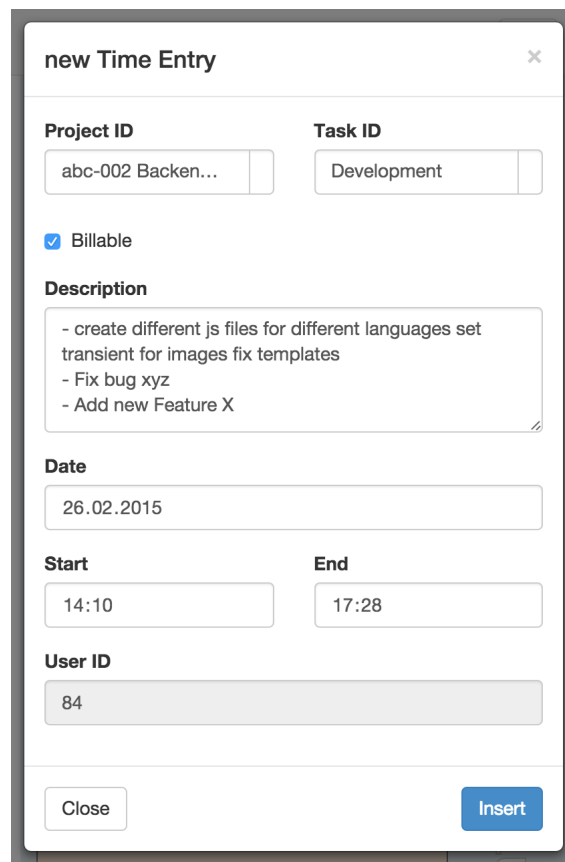
Project ID Der Beschreibungstext des Ereignisses wird nach Projekt-Kürzel durchsucht. Falls ein passendes Projekt gefunden wird, wird es ausgewählt.

Task ID Ereignisse aus Github oder Redmine werden standardmässig als “Development” gesetzt.

Billable Manche Tasks sind als “Billable” vordefiniert. Diese Information kann aus “Controllr” abgefragt werden

Date, Start, End Datum, sowie Start- und Endzeit. Die Startzeit wird, wie im Abschnitt 4.3 beschrieben, ergänzt, falls sie nicht gesetzt ist.

User ID Wird aus den Einstellungen des Benutzers übernommen.



The image shows a 'new Time Entry' form with the following fields and values:

- Project ID:** abc-002 Backen...
- Task ID:** Development
- Billable:** ☒
- Description:**
 - create different js files for different languages set transient for images fix templates
 - Fix bug xyz
 - Add new Feature X
- Date:** 26.02.2015
- Start:** 14:10
- End:** 17:28
- User ID:** 84

At the bottom, there are 'Close' and 'Insert' buttons.

Abbildung 8: Eingabemaske für einen Zeiteintrag. Alle Felder werden vorausgefüllt

6 Ausblick und Diskussion

Die Arbeit war sowohl aus konzeptioneller, wie auch aus technischer Sicht herausfordernd, zeigt aber auch eine spannende Richtung, wie Dienste verbunden werden können um Prozesse zu automatisieren.

Der umgesetzte Prototyp stiess auf Anklang, es kamen auch bereits weitere Ideen und Verbesserungen auf, die in einem neuen Rahmen zukünftig umgesetzt werden sollen. Auch die Architektur erwies sich als sinnvoll, neue Datenquellen können zukünftig mit wenig Aufwand eingebunden werden.

6.1 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”²² 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.

Durch diesen Zwischenschritt über den serverseitigen Teil der Applikation kommuniziert ein Client auch nie direkt mit einer Datenquelle. Dies kann sicherheitstechnische Vorteile haben und ermöglicht auch, häufig abgefragte Daten auf dem Server zwischenzuspeichern (siehe dazu nächster Abschnitt).

6.2 Probleme und Erweiterungen zum “Reactive REST-Mapping”

Meteor verwaltet für jeden aktiven Benutzer eine Verbindung und zugehörige Abonnements (“subscriptions”). Dieses Konzept wurde bei der Gestaltung von `panter:publish-array` berücksichtigt; jeder aktive Benutzer erhält eine eigene aktive “subscription”, welche auf dem Server die abonnierten Quellen regelmässig nach neuen Daten abfragt. Nachteil an dieser Lösung ist, dass bei mehreren aktiven Benutzern die Quellen sehr häufig kontaktiert werden, auch wenn die abgefragten Daten teilweise identisch sind. So ist die Projekt- und Task-Liste von “controllr” für alle Benutzer identisch. Eine Lösung wäre, diese Daten im serverseitigen Teil der Applikation zwischenzuspeichern und unter den abonnierenden Benutzer zu teilen. Diese Erweiterung könnte innerhalb von `panter:publish-array` gemacht werden.

6.3 Schwierigkeiten bei der Darstellung

Schwierigkeiten traten bei der visuellen Gestaltung auf: zwar lag der Fokus darauf, Benutzerinteraktionen zu minimieren und nicht auf der visuellen Darstellung, dennoch sind minimale Überlegungen dazu nötig.

Wie in Abbildung 7 zu sehen, können nicht immer alle Infos für ein Event angezeigt werden, wenn das Ereignis kurz ist. Eine Herausforderung war, den Skalierungsfaktor so zu wählen, das kleine Events noch sinnvoll angetippt werden können und noch die nötigsten Informationen zeigen. Ein zu grosser Skalierungsfaktor würde hingegen die Liste zu stark in der Höhe vergrössern, was die Übersicht reduziert.

Auch wurde die obige Abbildung auf einem relativ grossen Bildschirm gemacht. Auf kleinen Smartphone-Bildschirmen ist die Übersicht noch etwas eingeschränkter.

²²Die Datenquelle wird periodisch abgefragt.

Mögliche Lösungen sind:

- Infos reduzieren (z.b. Beschreibungstext entfernen)
- Nutzung von Symbolen, z.b. für Task-Typen
- Skalierung dynamisch, z.b. minimale Höhe für kürzestes Event eines Tages

Abbildung 8 zeigt zudem, dass die grundsätzlichen Eingabe-Felder der Ursprünglichen Applikation nicht direkt reduziert wurden, sie werden lediglich vor-ausgefüllt. Ein weitergehender Ansatz wäre, diese Felder auch tatsächlich auszublenden oder nur optional einzublenden, wenn der User etwas anpassen möchte. Zudem könnten die Felder, sofern der Benutzer doch ein Feld anpassen möchte, noch weiter für mobile Endgeräte zu optimieren.

Es ist dennoch möglich, dass der Benutzer lediglich den Zeiteintrag bestätigen muss, sofern alle Felder für ihn bereits stimmen. Dies kommt der Eingangs erwähnten Idealvorstellung bereits nahe.

A Anhang

A.1 Quellcode

Quellcode der Arbeit:

<https://github.com/panter/timetraces>

Das ebenfalls umgesetzte Package "panter:publish-array":

<https://github.com/panter/meteor-publish-array>

A.2 Quellenangaben

Titelbild:

Github-Icon © 2015 GitHub, Inc.

Kalender und Gmail-Icon Google Inc.

Redmine-Icon <http://de.wikipedia.org/wiki/Redmine>

"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 API." <https://developer.github.com/v3/>.

"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 Now." <https://www.google.com/landing/now/>.

"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/>.

"Panter:publish-Array Github Repository." <https://github.com/panter/meteor-publish-array>.

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

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

"Responsive Webdesign - Wikipedia." http://de.wikipedia.org/wiki/Responsive_Webdesign.