

Standort-basierte Daten via MQTT

Seminararbeit "Webservice Security"

Marco Wettstein

2015-05-14

Inhaltsverzeichnis

1	Einleitung	3
2	Ausgangslage	4
2.1	Timetraces	4
2.1.1	Standortverlauf als weitere Event-Quelle	4
2.1.2	No-Data-Konzept	5
2.2	Standort-Dienste und MQTT	8
3	Aufgabenstellung	9
3.1	Zielsetzung	9
4	Recherche	9
4.1	Standard-Dienste	9
4.1.1	Google Latitude	9
4.1.2	Google+	9
4.1.3	OwnTracks	11
4.2	MQTT	11
4.2.1	Netzwerklayer und Sicherheit	11
4.2.2	Anwendungen	11
4.2.3	Topics und Publish-Subscribe	11
4.2.4	Quality of Service (QoS)	12
4.2.5	Broker	12
4.3	Verfügbare MQTT-Broker	12
4.3.1	Mosquitto	12
4.3.2	HiveMQ	13
4.3.3	Moquette	13
4.3.4	Mosca	13
4.3.5	GnatMQ	13
4.3.6	Gehostete MQTT-Broker	13
4.3.7	Wahl	14
5	Konzept & Design	15
5.1	Sicherheitsaspekte	15
5.1.1	Location Privacy	15
5.1.2	Verschlüsselung	15
5.1.3	Authentifizierung	15
5.2	Architektur	16

5.3 Standort-Service als Brücke	16
5.4 Konzeption eines Standort-Services	16
5.4.1 Rollen	16
5.4.2 User-Stories	16
5.4.3 System	17
5.4.4 Sicherheitsaspekte	19
6 Umsetzung	20
6.1 Integration von TimeTraces	20
6.2 Screenshots	20
7 Diskussion	21
8 Ausblick	22
8.1 Indoor-Standorte mittels Beacons	22
8.2 Generischer Event-Service	22
A Annotierter Quelltext - Location-Service	23
A.1 Simple location service with Meteor - app.coffee.md	23
A.1.1 Collections	23
A.1.2 DDP-API	23
A.1.3 REST-API	24
A.1.4 MQTT-Bridge	24
A.1.5 Client Views	25
A.2 File app.jade	26
A.3 File app.styl	27
B Integration des Location-Service in timeTraces	27
B.1 LocationService.coffe.md	27
Quellenangaben	28

1 Einleitung

2 Ausgangslage

2.1 Timetraces

Im Rahmen einer Seminararbeit wurde für die Controlling- und Zeiterfassungs-Applikation “controllr” (siehe Abbildung 1) eine neue Client-Anwendung gebaut, welche durch die Integration verschiedener Dienste wie Github, Redmine und Google Calendar eine Art Protokoll der geleisteten Arbeit erstellt. Aus den Einträgen dieses Protokoll können in der Anwendung direkt Zeiteinträge in “controllr” erstellt werden. Abbildungen 2 zeigt das Arbeitsprotokoll von *TimeTraces*. Durch Anwählen eines Eintrages wird eine vor-ausgefüllte Eingabemaske angezeigt, welche den Zeiteintrag über eine REST-Schnittstelle an “controllr” sendet (Abbildung 3).

The screenshot displays the 'controllr' application interface. At the top, there's a 'Daily entries' section with a calendar for February 2015. The calendar shows the 10th of February as the selected date. To the right of the calendar is a form for creating a new entry. This form includes fields for 'Project' (filled with 'abc-123 - Project ABC'), 'Task' (filled with 'Development'), 'Description' (filled with 'Implement new Feature X'), 'Start' time (16:35), 'End' time (17:45), and a 'Duration' field. There is a 'Billable' checkbox which is checked, and a green 'Create Entry' button. Below the form, there's a section titled 'Entries for 10 Feb 2015' which contains a table of entries.

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” (Quelle (Wettstein, 6))

TimeTraces wurde als “Meteor”-Anwendung gebaut (siehe dazu Abschnitt ??) und ist eine Client-Server-Anwendung, welche externe Dienste integriert. Die Anwendung speichert dabei ausser den Benutzer-Logins und den Einstellungen der Benutzer keine weiteren Daten (Siehe Abschnitt 2.1.2). Sämtliche Daten werden dabei vom Serverteil der Anwendung aggregiert und an den Client gesendet. Die Daten werden vom Server dabei über REST-Schnittstellen in einem Polling-Verfahren abgerufen. Das Polling wird gestartet, sobald der clientseitige Teil der Anwendung die Daten über eine DDP-Subscription abonniert und beendet, sobald der Client die Subscription beendet.

Abbildung 4 zeigt den Ablauf einer Subscription eines Clients.

2.1.1 Standortverlauf als weitere Event-Quelle

TimeTraces nutzt bisher *Github*, *Google Calendar* und *Redmine* als Event-Quellen. Als weitere Event-Quelle soll nun der Standort-Verlauf des Benutzers genutzt werden. Diese Daten sollen dem Benutzer helfen, die Zeiteinträge genauer zu erfassen. Der Benutzer sieht somit nicht nur, was wann gearbeitet wurde, sondern auch wo. Es löst zudem das Problem, dass es oft schwierig ist, sich an den Startzeitpunkt einer Arbeit zu erfassen: es kann z.b. festgestellt werden, wann das Büro betreten wurde.

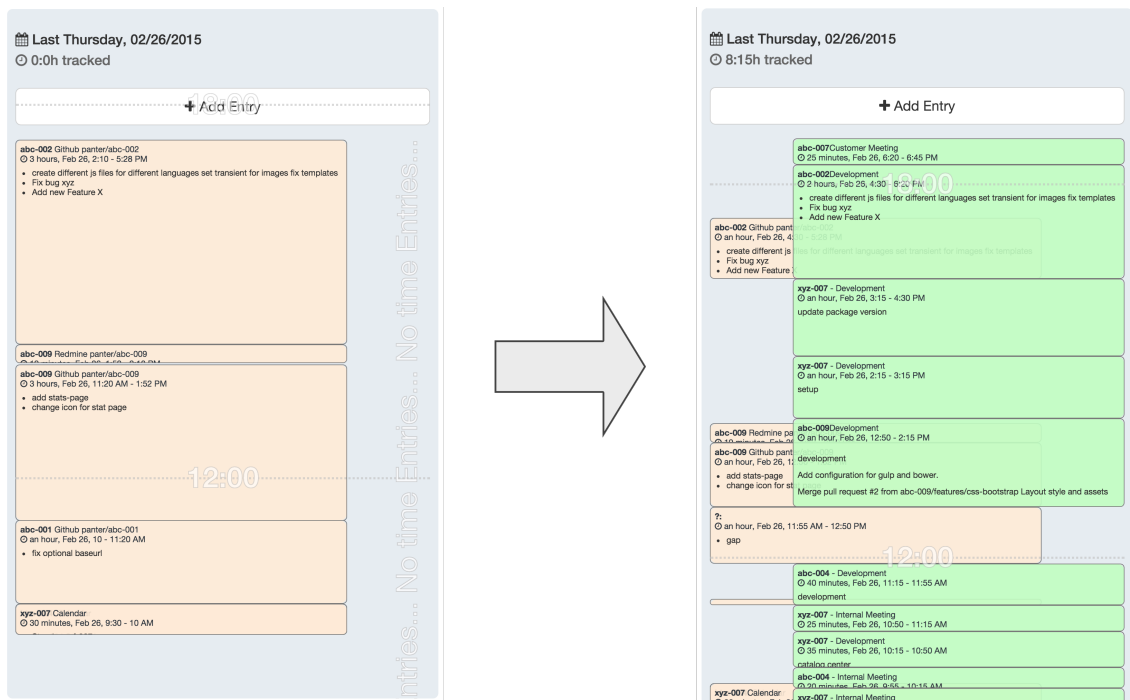


Abbildung 2: Darstellung der Event-Liste eines Tages in *TimeTraces*. Quelle (Wettstein, 22)

2.1.2 No-Data-Konzept

Im Zeitalter von Cloud-Computing ist es schwierig geworden, genau zu bestimmen, wo die Daten eines Benutzers überall sind und wer die Controller über diese Daten hat.

Beim Design von *TimeTraces* wurde dies berücksichtigt, indem nur die nötigsten Daten, wie Einstellungen von Benutzer gespeichert werden. Die verwendeten Daten der Event-Quellen, wie Kalender und Github werden dabei nicht dupliziert und gespeichert, sondern lediglich an die Client-Anwendung weitergeleitet.

new Time Entry [X]

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

Close [Insert]

Abbildung 3: Eingabemaske für einen Zeiteintrag in *TimeTraces*. Alle Felder werden vorausgefüllt. Quelle (Wettstein, 23)

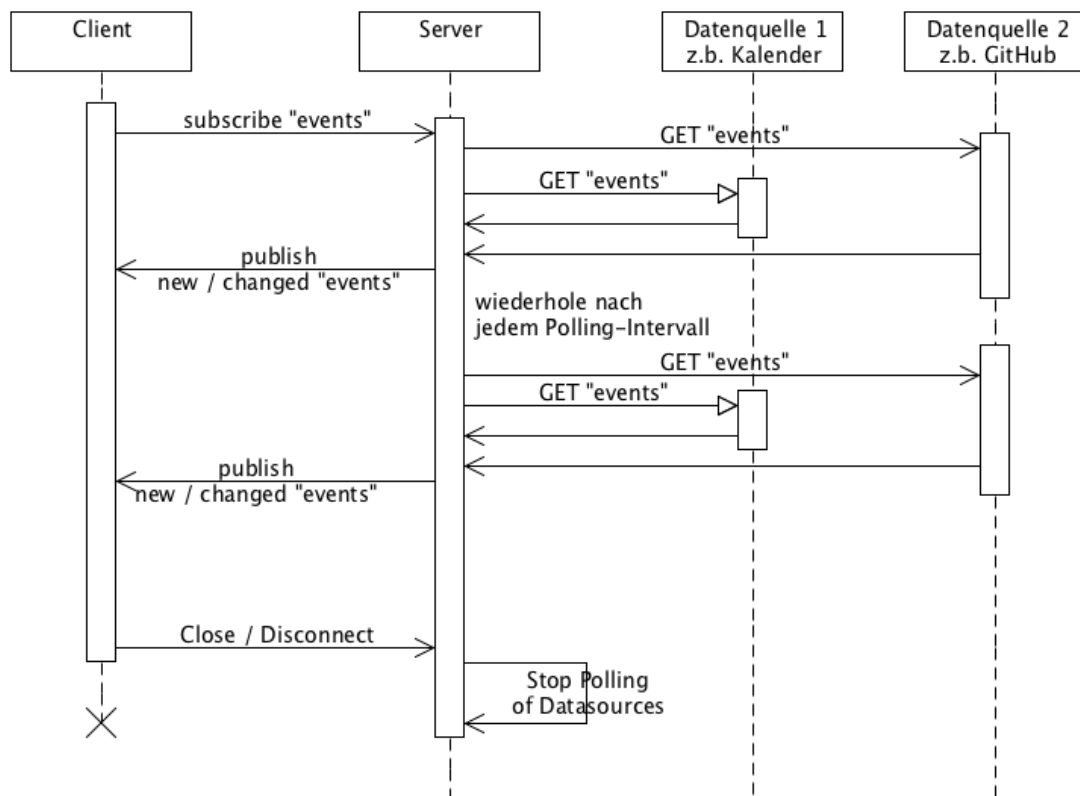


Abbildung 4: Ablauf einer Subscription von *TimeTraces* zwischen Client - Server und externen Quellen.
Quelle (Wettstein, 19)

2.2 Standort-Dienste und MQTT

Die Fähigkeiten zur Standort-Bestimmung (GPS, o.ä.) auf Smartphones ermöglicht viele Anwendungen, wie Navigation, Standort-bezogene Informationen oder Protokollierung für Sportler, Wanderer, o.ä. Es existieren viele Dienste zum diesem Thema, unter anderem von Google, wie *Google+* oder den eingestellten Dienst *Google Latitude*. Im mobilen Betriebssystem *Android* ist die Standort-Bestimmung zudem seit den ersten Versionen ein wichtiges Feature.

Neben solchen Cloud-Diensten existieren auch Systeme, welche auf eigener Infrastruktur - und damit unabhängig von Drittanbieter betrieben werden können.

Ein solches System ist die Anwendung *OwnTracks*, welche als Ersatz für den eingestellten Dienst *Google Latitude* entwickelt wurde. Sie kann auf IOS- und Android-Geräten installiert werden und kommuniziert über einen *Broker* über das Protokoll *MQTT*, welches als *Protokoll des Internets der Dinge* standardisiert werden.

3 Aufgabenstellung

Auf einem mobilen Endgerät des Benutzers soll eine Anwendung laufen, welcher seinen Standort periodisch an eine zentrale Stelle übermittelt.

Die Anwendung *TimeTraces* wird um eine Funktion erweitert, welche diese Daten abfragt und auswertet. Der Benutzer soll dabei sehen, zu welchem Zeitpunkt er sich an welchem Standort aufgehalten hat. Die Daten werden also nicht in Echtzeit benötigt (wie es beispielsweise häufig bei Standort-basierter Werbung der Fall ist), sondern mit Verzögerung (einige Tage bis wenige Wochen).

Als Übertragungsprotokoll soll dabei *MQTT* verwendet werden

3.1 Zielsetzung

Die Ziele dieser Arbeit sind

- Vertiefung in das Thema MQTT
- Betrachtung sicherheitsrelevanter Aspekte von MQTT,
- Betrachtung generell sensibler User-Daten, wie Standortverlauf
- Setup einer geeigneten MQTT-Broker-Lösung mit geeigneten Sicherheitseinstellungen
- Verbinden von *OwnTracks* oder einer ähnlichen Anwendung und *TimeTraces* via MQTT und dem gewählten Broker

4 Recherche

4.1 Standard-Dienste

4.1.1 Google Latitude

Google Latitude war ein Dienst, welche es einem Benutzer ermöglichte, seinen Standort an seine Freunde zu senden und zu sehen, wo sich seine Freunde aufhalten.

Der Dienst wurde eingestellt.¹

4.1.2 Google+

Im Sozialen Netzwerk *Google+* ist es möglich, seinen Standort an Freunde zu teilen. Der Dienst nutzt dabei das mobile Endgerät des Benutzers für die Standort-Protokollierung, sofern man die entsprechende App installiert hat.

Über eine *API* lassen sich Daten des Sozialen Netzwerkes programmatisch abfragen. Über

GET <https://www.googleapis.com/plus/v1/people/userId>

lassen sich Profil-Daten eines Benutzers abfragen. Wählt man `me` als `userId`, so kann das eigene Profil abgefragt werden. Die Dokumentation der Schnittstelle² erwähnt das Feld `currentLocation` als den aktuellen Standort der Person. Eine Test der Schnittstelle hat aber ergeben, dass das Feld nicht existiert. Auch eine gezielte Abfrage wie in Abbildung 5 gezeigt, ergab, dass das Feld nicht existiert.

Es handelt es sich hier offenbar um einen Fehler. Im *issue-tracker* der *google-plus-platform* ist ein entsprechender Eintrag³ verfasst worden mit dem Datum 24.07.2013. Bis zum Zeitpunkt des Erfassens dieser Arbeit ist keine Lösung zum Problem eingegangen.

¹Siehe ("Latitude Wurde Eingestellt")

²Siehe ("Google+ Platform - People (Personen)")

³Siehe ("Field 'CurrentLocation' Is Not Provided in API Result (or Empty), While the Field Is Clearly Filled on the Page.")

Authorize requests using OAuth 2.0: ☒ ON ⓘ

userid

me

The ID of the person to get the profile for. The special value "me" can be used to indicate the authenticated user. (string)

fields

name

Selector specifying which fields to include in a partial response.
[Use fields editor](#)

bold red = required

Execute

plus.people.get executed moments ago

time to execute: 236 ms

Request

GET https://www.googleapis.com/plus/v1/people/me?fields=name&key={YOUR_API_KEY}

Response

200 OK

- Show headers -

```
{
  "name": {
    "familyName": "Wettstein",
    "givenName": "Marco"
  }
}
```

Authorize requests using OAuth 2.0: ☒ ON ⓘ

userid

me

The ID of the person to get the profile for. The special value "me" can be used to indicate the authenticated user. (string)

fields

currentLocation

Selector specifying which fields to include in a partial response.
[Use fields editor](#)

bold red = required

Execute

plus.people.get executed moments ago

time to execute: 307 ms

Request

GET https://www.googleapis.com/plus/v1/people/me?fields=currentLocation&key={YOUR_API_KEY}

Response

200 OK

- Show headers -

```
{
}
```

Abbildung 5: Abfrage der Google+-Schnittstelle. Wird *name* abgefragt, erscheint eine Antwort, bei *currentLocation* nicht.

4.1.3 OwnTracks

Owntracks wurde als Ersatz für den eingestellten Google Standort-Dienst "Latitude" entwickelt und ursprünglich in Anlehnung an das Vorbild und dem Verwendeten Protokoll als *MQTTitude* bezeichnet.⁴

Die Anwendung zeichnet den Standort des Benutzers im Hintergrund auf und sendet die Daten an einen zu definierenden MQTT-Broker unter einem wählbaren *Topic* (Siehe Abschnitt 4.2.5). Dabei können verschiedene Einstellungen wie die Häufigkeit der Standortprotokollierung

OwnTracks ist als quelloffene Anwendung für IOS und Android erhältlich und ist unter der *Eclipse Public Licence* veröffentlicht.⁵

Die Verwendung von *OwnTracks* ist im Rahmen dieser Arbeit als Ausgangslage vordefiniert und gibt das Protokoll *MQTT* vor, es sollen aber auch Alternativen betrachtet werden⁶

4.2 MQTT

Message Queue Telemetry Transport oder kurz *MQTT* ist ein Protokoll für die Maschine-zu-Maschine-Kommunikation von Telemetrie-Daten. MQTT wurde insbesondere für Leistungsschwache Endgeräte entwickelt, sowie für Netzwerke mit hoher Verzögerung oder geringer Leistung. So wurde MQTT auch für die Kommunikation über Satelliten genutzt.⁷

MQTT wurde 1999 von Dr Andy Stanford-Clark (IBM) und Arlen Nipper (Arcom, Eurotech) entwickelt.⁸

4.2.1 Netzwerklayer und Sicherheit

MQTT ist im TCP/IP-Referenzmodell in der Anwendungs-Schicht angesiedelt und nutzt TCP als Übertragungsprotokoll. Die Übertragung kann mittels SSL/TSL verschlüsselt werden, allerdings erhöht das den Leistungsbedarf der Übertragung signifikant. MQTT sieht innerhalb des Protokolles keine Verschlüsselung vor, es ist jedoch möglich, sich mittels Benutzername und Passwort zu authentifizieren.⁹

Mit *MQTT-SN* steht eine Variante für nicht-TCP/IP-Netzwerke, wie *ZigBee* zur Verfügung.¹⁰

4.2.2 Anwendungen

Durch diese Optimierungen ist MQTT für Sensoren, wie Temperatur-, Feuchtigkeits oder Druckmesser, Lichtschalter, Bewegungsmelder und Aktoren, wie Lampen, Motoren, Relais oder ähnliches geeignet. MQTT wurde 2013 als Protokoll des *Internets der Dinge* standardisiert und bietet somit beispielsweise eine standardisierte Übertragungsmöglichkeit für die Hausautomation.¹¹

Durch die geringe Leistungsaufnahme ist MQTT ebenfalls geeignet für mobile Endgeräte, wie Smartphones, wo lange Akkulaufzeit und geringe Datenübertragung wünschenswert sind.

4.2.3 Topics und Publish-Subscribe

MQTT folgt dem Konzept einer *Message oriented Middleware* und ermöglicht das *Beobachter-Entwurfsmuster*, welches auch *publish-subscribe* genannt wird.¹² Dabei *abonniert* ("subscribe") ein

⁴Siehe Quelle ("OwnTracks")

⁵Lizenz und Quelle unter ("OwnTracks Lizenz")

⁶Owntracks nutzt MQTT als Übertragungsprotokoll. Es ist aber auch denkbar, dass anderen Anwendungen ein anderes Protokoll verwenden.

⁷Siehe Einleitung unter der Offiziellen Seite von MQTT (???).

⁸Siehe ("MQTT - Frequently Asked Questions")

⁹Siehe ("MQTT - Frequently Asked Questions")

¹⁰Siehe ("MQTT for Sensor Networks – MQTT-SN").

¹¹Quelle ("Wikipedia - MQ Telemetry Transport")

¹²Siehe Quelle ["fn_observer_pattern"].

Client ein bestimmtes *Topic*. Ein Client kann auf ein *Topic* eine Nachricht *veröffentlichen* ("publish"), welche dann alle Clients erhalten, die dieses *Topic* abonniert haben. Ein *Broker* dient dabei als Vermittler zwischen den Clients und leitet die Nachrichten an die für sie bestimmten Clients weiter. (Siehe Abschnitt 4.2.5)

4.2.4 Quality of Service (QoS)

MQTT sieht 3 Stufen für die Übertragungs-Qualität einer Nachricht vor:

QoS 0 - At most once delivery: Die Nachricht wird **höchstens einmal** zugestellt. Nachrichten mit QoS 0 können verloren gehen, wenn ein Client die Verbindung unterbricht oder ein Broker offline ist. Der Vorteil an QoS 0 liegt primär in der Performance, da Nachrichten nicht zwischengespeichert und nicht protokolliert werden muss, welcher Benutzer welche Nachricht erhalten hat.

QoS 1 - At least once delivery: Clients und Broker versuchen, die Nachrichten **mindestens einmal** zuzustellen. Es ist möglich, dass Nachrichten mehrfach zugestellt werden.¹³

QoS 2 - Exactly once delivery: Diese Stufe garantiert, dass eine Nachricht genau einmal zugestellt wird. Die Stufe stellt somit wie QoS 1 den Empfang einer Nachricht sicher und vermeidet dabei Duplikate. QoS 2 stellt somit die höchste Qualitätsstufe der Übertragung dar und erfordert damit auch mehr Komplexität und Rechenleistung in Client und Broker.

4.2.5 Broker

Broker verbinden die verschiedenen Clients und dienen als Vermittler der Nachrichten. Sie nehmen Nachrichten von Clients entgegen und senden sie an andere Clients, welche das *Topic* der Nachricht abonniert haben. Broker berücksichtigen dabei die QoS-Stufe der Nachricht und müssen bei entsprechender QoS-Stufe Nachrichten auch Zwischenspeichern.

4.3 Verfügbare MQTT-Broker

4.3.1 Mosquitto

Mosquitto ist ein quelloffener MQTT-Broker und wurde unter der BSD-Lizenz veröffentlicht. Für verschiedene Plattformen und Betriebssysteme stehen vorkompilierte Pakete als Download oder in Paketmanagern zur Verfügung.¹⁴

Der Broker kann auf einem eigenen Server installiert werden, setzt somit aber bestehende Infrastruktur voraus.

Mosquitto speichert Daten im Arbeitsspeicher und persistiert die Daten periodisch auf den Datenträger. [`fn_mosquitto_autosave_interval`]

Weiterhin kann Mosquitto sich mit weiteren Brokern via *Bridge* verbinden.¹⁵

4.3.1.1 Verschlüsselung und Authentifizierung Mosquitto erlaubt Zertifikat-basierte Verschlüsselung mittels TLS/SSL. Der Server weist dabei an den Client ein Zertifikat aus, welches der Client verifiziert. Umgekehrt besteht die Option, dass der Client sich gegenüber dem Server ebenfalls mit einem Zertifikat authentifizieren muss. Dies kann dazu genutzt werden, einen User zu identifizieren. Ohne Client-Zertifikat ist auch eine Authentifizierung mittels Benutzername und Passwort möglich. Statt eines Zertifikates kann auch ein *pre-shared-key*-Verfahren genutzt werden, wobei vorgängig ein Schlüssel ausgetauscht wird.¹⁶

¹³Die in der Quelle ("What Is MQTT and How Does It Work with WebSphere MQ?") angegebene Seite zeigt eine Übersicht über MQTT und die Verschiedenen QoS-Level.

¹⁴Siehe ("Mosquitto Homepage")

¹⁵Siehe ("Mosquitto Bridges Options")

¹⁶Siehe ("Mosquitto Authentication")

Nutzt man die Client-Authentifizierung, so ist es möglich, die Zugriffsrechte eines Clients auf einzelne Topics einzuschränken. Dabei können für ein Topic reine Lese- und Schreibrechte oder Beides vergeben werden.¹⁷ Es ist darüber hinaus auch möglich, nicht-authentifizierte Benutzer auszuschliessen.

Mosquitto unterstützt weiterhin die Kommunikation über Websockets, d.h. es kann via Websockets direkt mit dem MQTT-Broker kommuniziert werden.

4.3.2 HiveMQ

HiveMQ ist ein laut Hersteller für Unternehmen optimierter MQTT-Broker und zeichne sich durch hohe Performance, gute Skalierbarkeit (durch Clustering), hohe Sicherheit, sowie 100%iger MQTT 3.1.1 Unterstützung überzeugen.

Wie Mosquitto unterstützt HiveMQ Kommunikation über Websockets, (X509) Zertifikat-Authentifizierung, sowie Bridging.

HiveMQ ist kostenpflichtig, es stehen monatliche Lizenzen oder einmalige Lizenzen zur Verfügung. Die Kosten für die einmalige Lizenz bewegen sich zwischen 325 € und 6250 €. Eine kostenlose Testversion steht ebenfalls zur Verfügung.

4.3.3 Moquette

Moquette ist ein Quelloffener MQTT-Broker, welcher in Java implementiert ist. Er steht unter der Apache License 2.0.

Der Broker wird aktiv weiterentwickelt, unterstützt aber weniger Funktionen von MQTT als Mosquitto oder HiveMQ. So ist es u.a. noch nicht möglich, Zugriffsrechte für einzelne User zu setzen. QoS 0,1 und 2 werden unterstützt.¹⁸

4.3.4 Mosca

Dieser Broker ist als *node.js*-Applikation und über den *Node Package Manager (npm)* erhältlich. Er kann direkt als Broker genutzt werden oder in einer anderen *node.js*-Applikation als Modul genutzt werden.

Mosca ist MQTT 3.1 kompatibel, unterstützt aber nur QoS 0 und 1. Nachrichten können auf einer MongoDB oder Redis-Datenbank persistiert werden.

Der Quellcode von Mosca ist unter Github veröffentlicht, es ist allerdings keine Lizenz angegeben.¹⁹

4.3.5 GnatMQ

GnatMQ ist ein auf dem .NET-Framework von Microsoft basierender MQTT-Broker. Er unterstützt alle 3 QoS-Stufen, Authentifizierung via Benutzername/Passwort, sowie Zugriffskontrolle, jedoch noch keine SSL/TLS-Verbindung und kein *Bridge*-Modus.

4.3.6 Gehostete MQTT-Broker

Statt einer selbst verwalteten Lösung, kommen auch Cloud-Dienste in Frage (*software as a service*).

4.3.6.1 CloudMQTT *CloudMQTT* ist ein gehosteter MQTT-Broker. Es gibt ein kostenloses Abonnement und zwei kostenpflichtige (\$19, resp. \$99 pro Monat).

Der Dienst unterstützt u.a. Authentifizierung, Zugriffskontrolle (*ACL*), *Bridge*-Modus, sowie Kommunikation über Websockets. `^fn_cloud_mqtt]`

¹⁷Siehe Option `* acl_file*` in Quelle ("Mosquitto General Options")

¹⁸Siehe ("Github Moquette")

¹⁹Siehe ("Github Mosca")

4.3.6.2 Self-Hosted gegen Cloud-Lösung Bei Standort-Daten handelt es sich um sensible Daten. Diese besitzen auch einen gewissen *Business Value*, was bei Standort-basierter Werbung zu sehen ist. Bei Cloud-Lösungen wird die Kontrolle über diese Daten an einen Fremdanbieter abgegeben. Im Rahmen dieser Arbeit ist dies zweitrangig; in einer produktiven Anwendung ist dies aber nicht zu vernachlässigen.

Cloud-Lösungen bieten aber den Vorteil des einfacheren Setup. Skalierung und Betrieb kann ebenfalls an den Anbieter abgegeben werden.

4.3.7 Wahl

Alle untersuchten Broker kämen prinzipiell in Frage (bei den Kostenpflichtigen in einer Test-Version), die Wahl fiel jedoch auf Mosquitto, da dieser sehr gut dokumentiert ist und sehr viele MQTT-Features unterstützt. Somit kann sich mit vielen Aspekten von MQTT auseinandergesetzt werden.

Für die Installation steht u.a. eine (virtuelle) Ubuntu Instanz (12.04 LTS) in einem Rechenzentrum zur Verfügung. Mosquitto kann direkt über Ubuntu/Debian's Paketmanager *apt* installiert werden.

5 Konzept & Design

5.1 Sicherheitsaspekte

5.1.1 Location Privacy

Standortdaten sind Sicherheitsrelevant. Es handelt sich um sensible Daten, die insbesondere vor Missbrauch geschützt werden müssen.

Der Schutz der *Location Privacy*, d.h. der Schutz der privaten Standortdaten ist spätestens seit dem Aufkommen GPS-fähiger Smartphones ein wichtiges Thema.

Zum Schutz der Standortdaten im Kontext der sogenannten *Vorratsdatenspeicherung* in Deutschland schrieb der *Chaos Computer Club* 2009:

“Durch die ungebremste Aufzeichnung der digitalen Spuren wird das Mobiltelefon mehr und mehr zu einer Ortungswanze, sofern dem speicherwütigen Staat nicht Einhalt geboten wird. Sollte die Vorratsdatenspeicherung vor Gericht Bestand haben, bedeutet das praktisch ein Ende der Freiheit, unbeobachtet und ungestört zu leben”

(Frank Rieger, “Chaos Computer Club Veröffentlicht Stellungnahme Zur Vorratsdatenspeicherung”)

6 Jahre später bleibt der Schutz der Standortdaten ein aktuelles Thema, die Universität Zürich hat dazu (Stand Mai 2015) eine Masterarbeit ausgeschrieben.²⁰

5.1.2 Verschlüsselung

Das Verschlüsseln der Übertragung zwischen Endgeräten dient dem Schutz der Daten vor dem Einblick oder Manipulation durch Dritte.

Eine Verschlüsselung kann *Ende-zu-Ende* über die *gesamte Übertragungsstrecke* erfolgen, d.h. beispielsweise von der Aufzeichnung der Daten auf dem Mobil-Gerät, bis zur Anzeige, resp. Auswertung der Daten auf der endgültigen Anwendung.

Die Übertragung kann auch nur Streckenweise verschlüsselt werden. *SSL/TLS* verschlüsselt beispielsweise die TCP/IP-Verbindung zwischen dem Mobil-Gerät und dem MQTT-Broker, der Broker selbst “sieht” die Daten allerdings im Klartext. MQTT sieht selbst keine Verschlüsselung innerhalb des Protokolles vor, es ist jedoch mit einem eigenen Verfahren möglich die Anwendungsdaten zu verschlüsseln. Ohne eine solche Verschlüsselung ist ein MQTT-Broker aber eine sicherheitsrelevante Stelle. Sensible Daten müssen also über einen vertraulichen Broker gesendet werden. Auch Übergänge von MQTT in ein anderes Protokoll (z.b. Websockets) sind Sicherheitsrelevant. Die entsprechenden Protokolle müssen eine gleichwertige Verschlüsselung benutzen (z.b. *HTTPS*).

5.1.3 Authentifizierung

Neben der *Vertraulichkeit* und der *Integrität* der Daten ist ferner die *Authentizität*, d.h. die Echtheit der Daten, sowie die *Verbindlichkeit* und *Zurechenbarkeit* von Bedeutung.

Jemand, der den Dienst nutzt, muss sich also ausweisen können. Dies kann über Benutzername und Passwort, über einen Schlüssel oder ein Zertifikat erfolgen.

Für die zu erstellende Anwendung soll ein Benutzername-Passwort-Verfahren zur Anwendung kommen.

²⁰Siehe (“Masterarbeit - Location Privacy - Probleme Und Lösungen”)

5.2 Architektur

Die Standort-Daten des Benutzers werden nicht in Echtzeit, sondern mit Verzögerung benötigt, um nachträglich den Standort-Verlauf des Benutzers zu rekonstruieren. Es ist also nötig, die Daten zwischenspeichern.

Da die Anwendung *TimeTraces* ein Server-Teil hat, könnte dieser für die Standortdatenspeicherung in Frage kommen.

Dagegen spricht aber, dass *TimeTraces* aus konzeptuellen Überlegungen keine Event-Daten speichert (siehe Abschnitt 2.1.2). Möchte man dieses Konzept beibehalten, so muss das Speichern der Daten auf eine andere Anwendung verlagert werden.

Die Broker-Software *Mosquitto* selbst kommt dafür ebenfalls nicht in Frage, da MQTT mit QoS 1 und 2 zwar Nachrichten persistiert, für eine ein- oder mehrwöchige Zwischenspeicherung ist ein Broker aber nicht ausgelegt.

5.3 Standort-Service als Brücke

Um dieses Problem zu lösen, kann ein Service erstellt werden, der zwischen *TimeTraces* und dem MQTT-Broker *Mosquitto* vermittelt. Da *TimeTraces* auf *Meteor* basiert, bietet es sich an, diesen Brücken-Dienst ebenfalls als Meteor-Applikation zu erstellen und zu *TimeTraces* via DDP zu kommunizieren.²¹ Prinzipiell können die Standort-Daten auch über eine REST-Schnittstelle an *TimeTraces* übermittelt werden, da sie nur in eine Richtung gesendet werden. Da es sich aber bei MQTT und DDP Message-Oriented-Middleware handelt mit dem Publish-Subscribe-Muster, bietet es sich an, DDP für die Übertragung zu verwenden.

5.4 Konzeption eines Standort-Services

5.4.1 Rollen

Der Standortservice benötigt zwei Rollen: der *Benutzer*, welche den Dienst nutzt und den *Administrator*, welcher den Dienst betreut und betreibt.

Weiterhin findet eine *Mashine-to-mashine*-Kommunikation mit dem vorgelagerten MQTT-Broker und der nachgelagerten Anwendung *TimeTraces* statt.

5.4.2 User-Stories

Die Anforderungen an diesen Standortservice werden durch folgende *User Stories* beschrieben:

Sc-01	Als Benutzer möchte ich mich für den Service registrieren können
<i>Story</i>	Neue Benutzer sollen sich registrieren können, damit sie sich später anmelden können
<i>Akzeptanzkriterien:</i>	<input type="checkbox"/> Benutzer kann Email angeben <input type="checkbox"/> Benutzer kann Passwort angeben <input type="checkbox"/> Benutzer TODO
<i>Priorität</i>	MUST

²¹Eine Evaluation möglicher Technologien für einen solchen Service ist nicht Teil dieser Arbeit. Die Wahl der Technologie basiert hier auf Vorwissen und Erfahrung des Autors.

Sc-02	Als Benutzer möchte ich mich an den Service anmelden können
<i>Story</i>	Registrierte Benutzer können sich anmelden
<i>Akzeptanzkriterien:</i>	<input type="checkbox"/> Benutzer kann Benutzername und Passwort eingeben <input type="checkbox"/> Benutzer ist danach autorisiert und kann den Dienst nutzen
<i>Priorität</i>	MUST

Sc-02	Als Benutzer möchte ich mich vom Service abmelden können
<i>Story</i>	Angemeldete Benutzer können sich vom Service abmelden. Danach müssen sie sich wieder anmelden um den Dienst zu benutzen
<i>Akzeptanzkriterien:</i>	<input type="checkbox"/> Es gibt eine Abmeldemöglichkeit (Schaltflächen, API-Befehl) <input type="checkbox"/> Nach dem Abmelden können keine Daten mehr abgefragt werden <input type="checkbox"/> Benutzer kann sich im abgemeldeten Zustand wieder anmelden oder registrieren
<i>Priorität</i>	MUST

Sc-03	Als Benutzer möchte ich meine Standort-Protokoll-Anwendung (z.b. *Owntracks*) mit dem Dienst verbinden können
<i>Story</i>	Der Benutzer kann seine Protokoll-Anwendung so konfigurieren, dass sie den Standort an den Dienst sendet
<i>Akzeptanzkriterien:</i>	<input type="checkbox"/> Benutzer hat alle nötigen Informationen, die er bei OwnTracks eingeben muss
<i>Priorität</i>	MUST

Sc-04	Als Benutzer möchte ich meine Standortdaten über eine Schnittstelle abfragen können
<i>Story</i>	Der Service soll Schnittstellen zur Verfügung stellen, mit welcher ein Anwendungen das Standort-Protokoll des Benutzers abfragen können
<i>Akzeptanzkriterien:</i>	<input type="checkbox"/> Kann nur autorisiert benutzt werden <input type="checkbox"/> Zeigt nur eigene Standort-Daten an <input type="checkbox"/> Bereich kann mittels von- bis-Datum eingeschränkt werden
<i>Priorität</i>	MUST

Sc-05	Als Benutzer möchte ich Standort-Einträge löschen können.
<i>Story</i>	Bestehende Einträge des Standort-Protokolles sollen sich permanentl öschen lassen können
<i>Akzeptanzkriterien:</i>	<input type="checkbox"/> Benutzer kann ein Datum angeben. Die Zeiteinträge vor diesem Datum werden gelöscht
<i>Priorität</i>	MAY

5.4.3 System

TODO

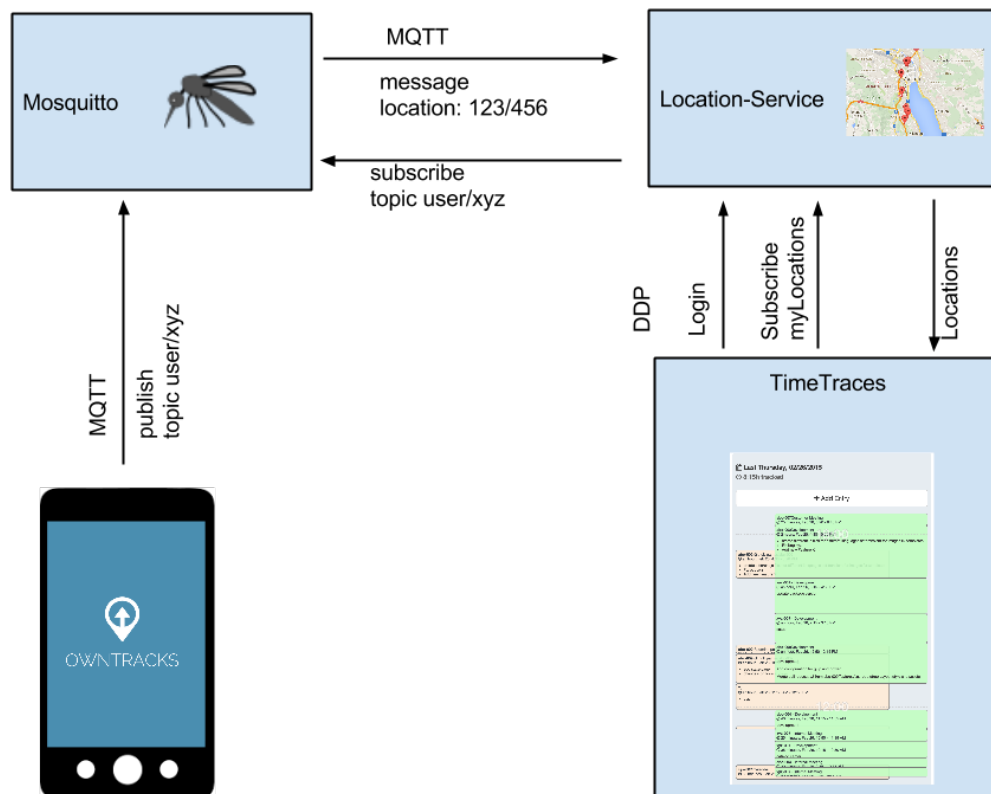


Abbildung 6: Systemübersicht

5.4.4 Sicherheitsaspekte

Der Dienst kommuniziert mit zwei weiteren Diensten und benötigt daher auch zwei Übertragungskanäle, welche potentielle Sicherheitsrisiken bergen.

Beide Kanäle müssen über eine aktuelle Technologie verschlüsselt werden.

5.4.4.1 Datenschutz Der Dienst speichert die schützenswerten Standort-Daten und muss entsprechend vor Missbrauch geschützt werden.

(TODO NFR):

- Ein angemeldeten Benutzer kann nur seine Standort-Daten auslesen und bearbeiten
- Ein nicht-angemeldeter Benutzer kann keine Daten sehen
- Die Zugriffsdaten für einen Benutzer müssen hinreichend geschützt sein.

6 Umsetzung

TODO

6.1 Integration von TimeTraces

TODO (Siehe Anhang)

6.2 Screenshots

TODO

7 Diskussion

TODO

8 Ausblick

8.1 Indoor-Standorte mittels Beacons

TODO

8.2 Generischer Event-Service

TODO

A Annotierter Quelltext - Location-Service

Nachfolgend der Quellcode des Location-Services [location.macrozone.ch](https://github.com/macrozone/location_service/) im *Literate CoffeeScript*-Stil.²² Er ist verfügbar unter https://github.com/macrozone/location_service/

(app.coffee.md)

A.1 Simple location service with Meteor - app.coffee.md

This app connects to a mqtt-broker that will receive location-data from “OwnTracks” and will expose it in

- a table (using aldeed:tabular)
- a google map
- a DDP-API
- a REST-API (using nimble:restivus)

A.1.1 Collections

First we need a collection to store the locations of the users.

```
1 Collections =  
2   Locations: new Meteor.Collection "Locations"
```

We define now a shared helper functions to get the topic string for a user or vis-verca. The topics have the pattern location/:userId

```
1 Topic =  
2   getTopicForUser: (user) -> "location/#{user._id}"  
3   getUserIdForTopic: (topic) -> topic.split("/") [1]  
4   getUserForTopic: (topic) -> Meteor.users.findOne  
   @getUserIdForTopic topic
```

A.1.2 DDP-API

We publish the data, so that a ddp client can subscribe to the data with subscribe “myLocations”, from: Date, to: Date

Note: we do not need publishes for the client-view, because aldeed:tabular handles that for us.

```
1 if Meteor.isServer then Meteor.publish "myLocations", (params)->  
2   selector = userId: @userId  
3   if params?.from?  
4     selector.tst ?= {}  
5     selector.tst["$gte"] = params.from  
6   if params?.to?  
7     selector.tst ?= {}  
8     selector.tst["$lte"] = params.to  
9   Collections.Locations.find selector
```

²²Siehe (“Literate CoffeeScript”)

A.1.3 REST-API

We provide additionally a REST-API using nimble:restivus. Here are the settings:

```

1 if Meteor.isServer
2   Restivus.configure
3     useAuth: yes
4     prettyJson: yes
5   Restivus.addCollection Collections.Locations,
6     routeOptions:
7       authRequired: yes

```

A.1.4 MQTT-Bridge

We now connect the mqtt-broker with our collection.

First, start a mqtt connection to the broker as soon as the server starts:

```

1 if Meteor.isServer then Meteor.startup ->
2   mqttClient = mqtt.connect "mqttp://lab.macrozone.ch:8883",
3     clientId: "location.macrozone.ch-#{ process.env.NODE_ENV }"
4     rejectUnauthorized: no
5   mqttClient.on "error", Meteor.bindEnvironment (error) -> console.
    log error

```

We initialize a geocoder (google by default), that will resolve our position to addresses.

```

1   geoCoder = new GeoCoder

```

A.1.4.1 Handle messages We listen now to messages from the broker.

```

1   mqttClient.on "message", Meteor.bindEnvironment (topic, data) ->

```

Owntracks sends data as JSON-Strings, so it needs to be decoded. The timestamp 'tst' gets converted to a Javascript-Date. We use the old timestamp as the id for the message to prevent duplicates:

```

1     message_id = data.tst
2     data = JSON.parse data.toString "utf-8"
3     data.tst = new Date parseInt(data.tst,10)*1000

```

Find the User for the topic "location/:userId", decode the position (lat, lon) to an address with the geocoder and insert it into "Locations"

```

1     user = Topic.getUserForTopic topic
2     if user? and data._type is "location"
3       data.userId = user._id
4       data.geo = _.first geoCoder.reverse data.lat, data.lon
5       Collections.Locations.upsert message_id, data

```

A.1.4.2 Handle mqtt-subscriptions for users We subscribe for every user and stop subscriptions for removed users. We start observing users after (re-) connect to the broker. In case of re-connect, stop the old observation

```

1     userObserveHandle = null
2     mqttClient.on "connect", Meteor.bindEnvironment ->
3       console.log "connected to mqtt-broker"
4       userObserveHandle?.stop()

```



```

5     userObserveHandle = Meteor.users.find().observe
6       added: (user) ->
7         startSubscriptionForUser user
8       removed: (user) ->
9         stopSubscriptionForUser user

```

Helpers to start and stop subscriptions on the mqttClient

```

1     startSubscriptionForUser = (user) ->
2       startSubscription Topic.getTopicForUser user
3     stopSubscriptionForUser = (user) ->
4       stopSubscription Topic.getTopicForUser user
5     startSubscription = (topic) ->
6       console.log "subscribed to #{topic}"
7       mqttClient.subscribe topic
8     stopSubscription = (topic) ->
9       console.log "unsubscribed to #{topic}"
10      mqttClient.unsubscribe topic

```

A.1.5 Client Views

A.1.5.1 Tabular The package `aldeed:tabular` will provide us with a data-table-like html-component. We use it to show the location history to the user. It is included in `app.jade` as `+tabular`. Subscriptions are handled automatically, depending on the filters on the table.

```

1 TabularTables =
2   Locations: new Tabular.Table
3     name: "Locations"
4     collection: Collections.Locations
5     order: [[0, "desc"]]
6     columns: [
7       {
8         data: "tst"
9         title: "Time"
10        render: (val) ->
11          moment(val).calendar()
12      },
13      {
14        data: "geo"
15        title: "Address"
16        render: (geo) ->
17          "#{geo.city}, #{geo.streetName} #{geo.streetNumber}" if geo?
18      }
19      {data: "lat", title: "Latitude", width: "80px"}
20      {data: "lon", title: "Longitude", width: "80px"}
21      {data: "batt", title: "Battery", width: "40px"}
22    ]

```

Restrict the data to the current user.

```

1     selector: (userId) -> {userId}

```

We define additionally some template-helpers on the client view

```

1 if Meteor.isClient
2   Template.registerHelper "dateFormat", (date) ->
3     (moment date).format("YYYY-MM-DD HH:mm")
4   Template.registerHelper "topicForCurrentUser", ->

```

```

5     Topic.getTopicForUser Meteor.user()
6     Template.registerHelper('TabularTables', TabularTables)

```

A.1.5.2 Routes We define a route to show the user the current location history and where unauthorized users can login or register. This will map to the template defined in [app.jade](#)

```

1 Router.route "/",
2     template: "locations"
3     layoutTemplate: "layout"

```

A.1.5.3 Google Maps We additionally show the current published locations in a google-map. We use the package `dburles:google-maps` for that.

```

1 if Meteor.isClient
2     Template.locations.helpers
3         googleMapsOptions: ->
4             if GoogleMaps.loaded()
5                 zoom: 8
6                 center: new google.maps.LatLng(-37.8136, 144.9631)
7
8     Template.locations.onCreated ->
9         GoogleMaps.load()
10        GoogleMaps.ready "locationsMap", (map) =>
11            markers = {}
12
13            bounds = new google.maps.LatLngBounds
14            shouldSetBounds = no
15            getPosition = (location) -> new google.maps.LatLng
16                location.lat, location.lon

```

We now observe now the current data in the Locations-Collection and add markers to google maps for them

```

1         observeHandle = Collections.Locations.find().observe
2             added: (location) ->
3                 position = getPosition location
4                 markers[location._id] = new google.maps.Marker
5                     position: position
6                     map: map.instance
7                     title: "#{location.tst} - #{location.geo?.city
8                         }, #{location.geo?.streetName}, #{location
9                             .geo?.streetNumber}"
10                bounds.extend position
11                map.instance.fitBounds bounds
12            changed: (location, oldLocation) ->
13                position = getPosition location
14                markers[location._id].setPosition position
15                bounds.extend position
16            removed: (location) ->
17                markers[location._id].setMap null
18                delete markers[location._id]
19
20        Template.locations.onDestroyed -> observeHandle.stop()

```

A.2 File app.jade

```

1
2 template(name="locations")
3   .row.locations
4     .map.col-sm-5
5       +googleMap(name="locationsMap" options=googleMapsOptions)
6     .locations.col-sm-7
7       +tabular(table=TabularTables.Locations class="table table-
8         striped table-bordered table-condensed")
9
10  template(name="layout")
11    .container-fluid
12      +loginButtons
13      if currentUser
14        +userInfo
15        +yield
16      else
17        .container
18          .alert.alert-warning
19            i.glyphicon.glyphicon-log-in
20            | Please login or register.
21
22  template(name="userInfo")
23    p Owntracks-Topic: #{topicForCurrentUser}

```

A.3 File app.styl

```

1
2 .map
3   height: 400px

```

B Integration des Location-Service in timeTraces

B.1 LocationService.coffe.md

This object enables one user to login to a location-service running at location.macrozone.ch and subscribe to it. It will receive Locations of the user in the LocationService.Locations-collection.

First, startup the connection to the server and initialize an empty collection:

```

1 connection = null
2 loggedIn = no
3 Meteor.startup ->
4   connection = DDP.connect "location.macrozone.ch"
5   LocationService.Locations = new Meteor.Collection "Locations",
6     connection

```

Wherever the saved locationServiceUser changes (or is initially set), refresh the login to the service:

```

1 Meteor.autorun ->
2   user = UserSettings.get "locationServiceUser"
3   if not loggedIn and user?.token?
4     LocationService.refresh user.token

```

Expose the LocationService with its methods globally (singleton):

```
1 @LocationService =
2   subscribe: (name, params) ->
3     connection.subscribe name, params
4   login: (email, password, callback) ->
5     console.log "login in"
6     params =
7       user: email: email
8       password: password
9     connection?.call "login", params, (error, user) ->
10      loggedIn = yes
11      UserSettings.set "locationServiceUser", user
12   refresh: (token) ->
13     console.log "refreshing"
14     connection?.call "login", resume: token, (error, user) ->
15      loggedIn = yes
16      UserSettings.set "locationServiceUser", user
17   logout: ->
18     connection?.call "logout"
19     UserSettings.set "locationServiceUser", null
20     console.log "login out"
21     loggedIn = no
22   fetchFor: (from, to) ->
23     @Locations.find({tst: {$gte:from, $lte:to }}, {sort: tst:
      -1}).fetch()
```

Quellenangaben

“Chaos Computer Club Veröffentlicht Stellungnahme Zur Vorratsdatenspeicherung.” <http://www.ccc.de/updates/2009/vds-gutachten>.

“Field ‘CurrentLocation’ Is Not Provided in API Result (or Empty), While the Field Is Clearly Filled on the Page.” <https://code.google.com/p/google-plus-platform/issues/detail?id=620>.

“Github Moquette.” <https://github.com/andsel/moquette>.

“Github Mosca.” <http://mcollina.github.io/mosca/>.

“Google+ Platform - People (Personen).” <https://developers.google.com/+/api/latest/people>.

“Latitude Wurde Eingestellt.” <https://support.google.com/gmm/answer/3001634?hl=de>.

“Literate CoffeeScript.” <http://coffeescript.org/#literate>.

“Masterarbeit - Location Privacy - Probleme Und Lösungen.” <http://www.ifi.uzh.ch/isr/teaching/masterarbeiten/locationprivacy.html>.

“Mosquitto Authentification.” <http://mosquitto.org/man/mosquitto-conf-5.html#idp49111296>.

“Mosquitto Bridges Options.” <http://mosquitto.org/man/mosquitto-conf-5.html#idp49322480>.

“Mosquitto General Options.” <http://mosquitto.org/man/mosquitto-conf-5.html#idp49117824>.

“Mosquitto Homepage.” <http://mosquitto.org/>.

“MQTT - Frequently Asked Questions.” <http://mqtt.org/faq>.

“MQTT for Sensor Networks – MQTT-SN.” <http://mqtt.org/tag/mqtt-s>.

“OwnTracks.” <https://github.com/owntracks/owntracks/wiki>.

“OwnTracks Lizenz.” <https://github.com/owntracks/android/blob/master/LICENSE>.

Wettstein, Marco. “TimeTraces - Seminararbeit ‘Entwickeln von Anwendungen Für Hand Held’.”

“What Is MQTT and How Does It Work with WebSphere MQ?” https://www.ibm.com/developerworks/mydeveloperworks/blogs/aimsupport/entry/what_is_mqtt_and_how_does_it_work_with_websphere_mq?lang=en.

“Wikipedia - MQ Telemetry Transport.” http://de.wikipedia.org/wiki/MQ_Telemetry_Transport.
. <http://mqtt.org/>.