Standort-basierte Daten via MQTT

Seminararbeit “Webservice Security”

Marco Wettstein

2015-05-14

Contents

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).

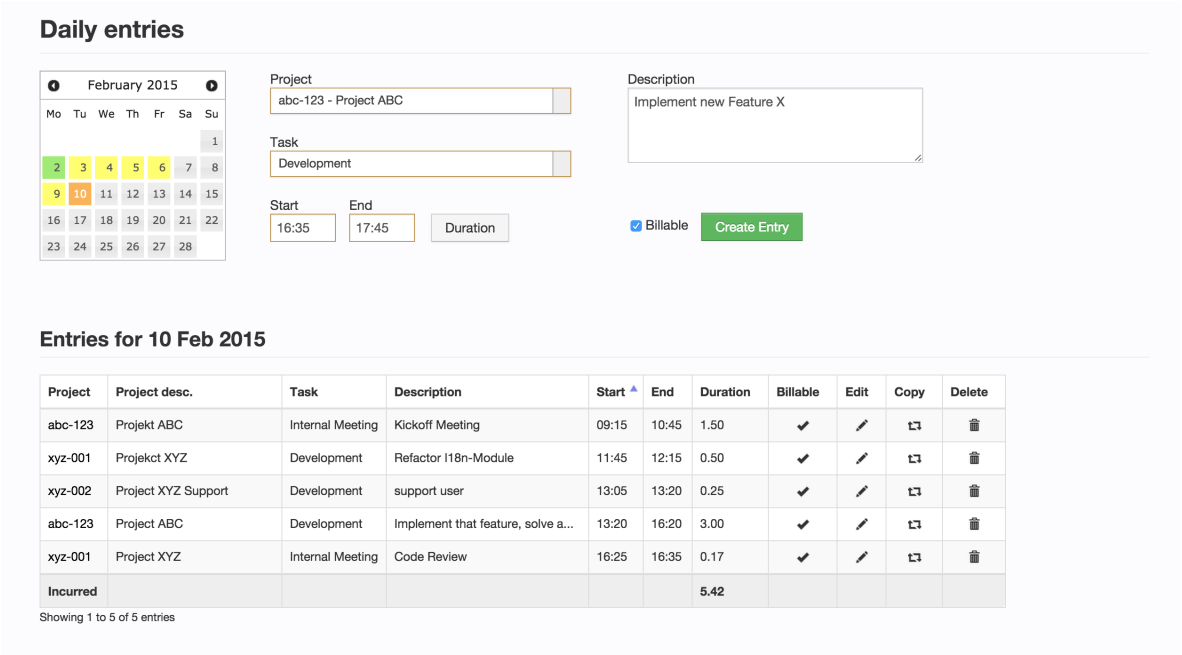

Figure 1: Screenshot von "Controllr" (Quelle (Wettstein, 6))

"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. 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

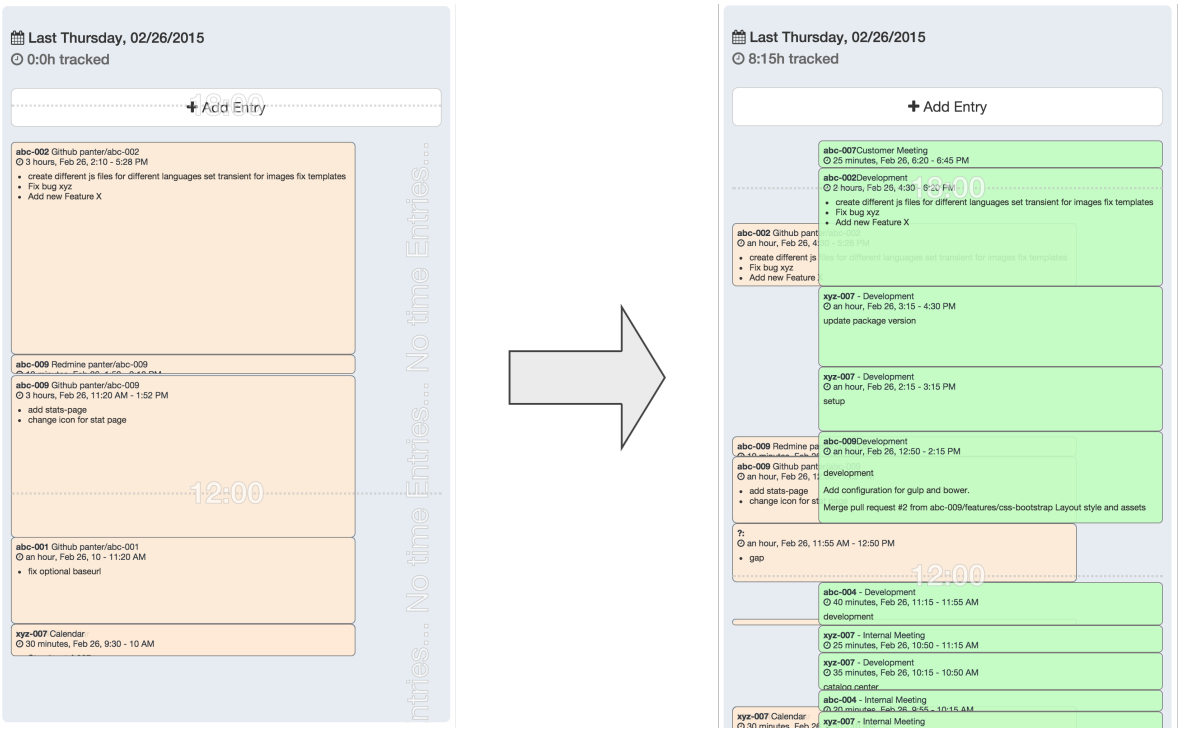

Figure 2: Darstellung der Event-Liste eines Tages in “timeTraces” (Quelle (Wettstein, 22))

Startzeitpunkt einer Arbeit zu erfassen: es kann z.b. festgestellt werden, wann das Büro betreten wurde.

2.2 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.[1]

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.[2]

Die Verwendung von *OwnTracks* ist im Rahmen dieser Arbeit als Ausgangslage vordefiniert und gibt das Protokoll *MQTT* vor, es sollen aber auch Alternativen betrachtet werden[3]

[1] Siehe Quelle (“OwnTracks”)
[2] Lizenz und Quelle unter (“OwnTracks Lizenz”)
[3] Owntracks nutzt MQTT als Übertragungsprotokoll. Es ist aber auch denkbar, dass anderen Anwendungen ein anderes Protokoll verwenden.

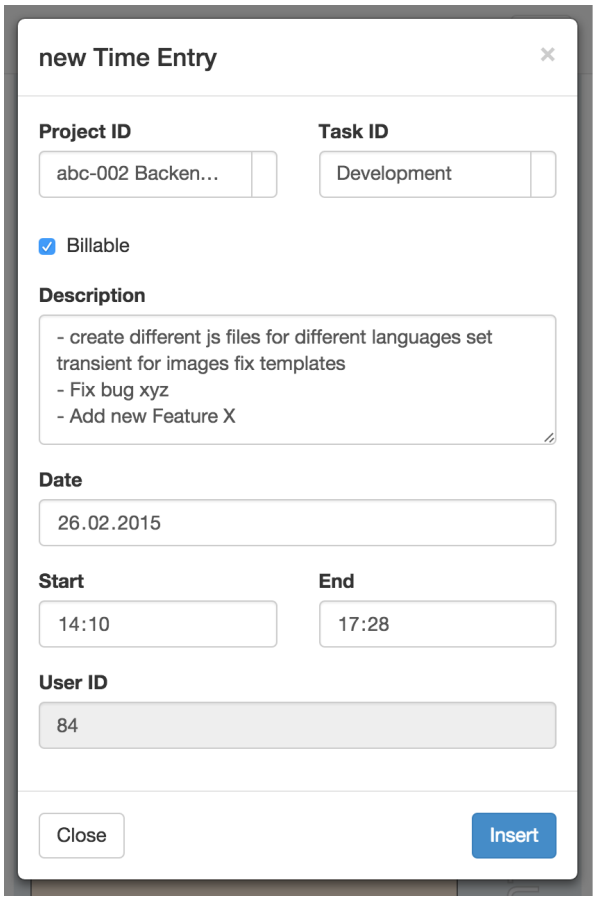

Figure 3: Eingabemaske für einen Zeiteintrag in “timeTraces”. Alle Felder werden vorausgefüllt. Quelle (Wettstein, 23)

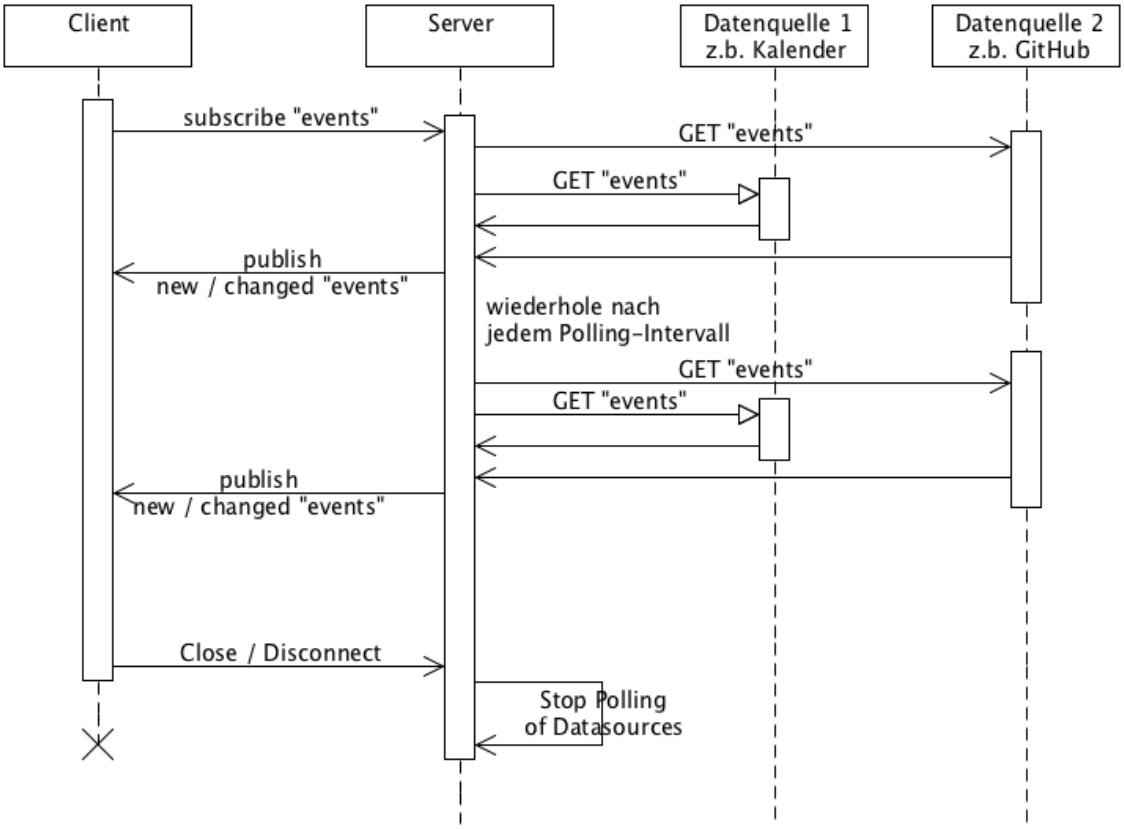

Figure 4: Ablauf einer Subscription von “timeTraces” zwischen Client - Server und externen Quellen (Quelle (Wettstein, 19))

3 Zielsetzung

Die Ziele dieser Arbeit sind

- Vertiefung in das Thema MQTT
- Betrachtung sicherheitsrelevanter Aspekte von MQTT,
- Betrachtung generell sensitiver 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 Location-Apps & -Dienste

https://play.google.com/store/apps/details?id=com.glympse.android.glympse http://onetouchlocation.creativeworkline.com

4.1.1 Owntracks

tODO: runter schieben?

4.1.2 Google+

4.2 MQTT

MQ 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.[4]

MQTT wurde 1999 von Dr Andy Stanford-Clark (IBM) und Arlen Nipper (Arcom, Eurotech) entwickelt.[5]

4.2.1 Netzwerklayer und Sicherheit

MQTT ist im TCP/IP-Referenzmodell in der Anwendungs-Schicht angesiedelt und nutzt TCP als Übetrtragungsprotokoll. Die Übetragung 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.[6]

Mit *MQTT-SN* steht eine Variante für nicht-TCP/IP-Netzwerke, wie *ZigBee* zur Verfügung.[7]

4.2.2 Anwendungen

Durch diese Optimierungen ist MQTT für Sensoren, wie Temperatur-, Feuchtigkeits oder Durckmesser, 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.[8]

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

[4] Siehe Einleitung unter der Offiziellen Seite von MQTT (**???**).
[5] Siehe (“MQTT - Frequently Asked Questions”)
[6] Siehe (“MQTT - Frequently Asked Questions”)
[7] Siehe (“MQTT for Sensor Networks – MQTT-SN”).
[8] Quelle (“Wikipedia - MQ Telemetry Transport”)

4.2.3 Topics und Publish-Subscribe

MQTT folgt dem Konzept einer *Message oriented Middleware* und ermöglicht das *Beobachter-Entwursfmuster*, welches auch *publish-subscribe* genannt wird.[9] Dabei *abonnniert* ("subscribe") ein Client ein bestimmtes *Topic*. Ein Client kann auf ein *Topic* eine Nachricht *veröffentlichen* ("publish"), welche dann alle Clients erhalten, die dieses *Topic* abbonniert 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 Qualitiy of Serivce (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.[10]

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 abbonniert 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 Betriebsysteme stehen vorkompilierte Pakete als Download oder in Paketmanagern zur Verfügung.[11]

Mosquitto speichert Daten im Arbeitspeicher und persistiert die Daten periodisch auf den Datenträger. [ˆfn_mosquitto_ autosave_interval]

Weiterhin kann Mosquitto sich mit weiteren Brokern via *Bridge* verbinden.[12]

[ˆfn_mosquitto_ autosave_interval]: Siehe Option * autosave_interval* in Quelle ("Mosquitto General Options")

[9] Siehe Quelle [ˆfn_observer_pattern].
[10] Die in der Quelle ("What Is MQTT and How Does It Work with WebSphere MQ?") angebene Seite zeigt eine Übersicht über MQTT und die Verschiedenen QoS-Level.
[11] Siehe ("Mosquitto Homepage")
[12] Siehe ("Mosquitto Bridges Options")

4.3.1.1 Verschlüsselung und Authentifizierung

Mosquitto erlaubt Zertifikat-basierte Veschlüsselung mittels TLS/SSL. Der Server weisst 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 identifzieren. 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.[13]

Nutzt mann 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.[14] Es ist darüber hinaus auch möglich, nicht-authentifizierte Benutzer auszuschliessen.

4.3.2 Wahl eines Brokers

4.3.3 Setup

[13] Siehe ("Mosquitto Authenfication")
[14] Siehe Option * acl_file* in Quelle ("Mosquitto General Options")

5 Design

5.1 Sicherheitsapsekte

5.1.1 Sensible Daten

5.1.2 Verschlüsselung

5.1.3 Authentifizierung

5.2 Architektur

DDP, Datenspeicherung auf timetraces

6 Umsetzung

6.1 Screenshots

7 Diskussion

8 Ausblick

8.1 Indoor-Standorte mittels Beacons

Generischer Event-Service

"Mosquitto Authenfication." 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/.