ulm university universität
uulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Konzeption und Entwicklung einer modularen, ereignisgesteuerten Server-Client-Architektur zur Crowd-basierten Datenerfassung mit mobilen Endgeräten

Masterarbeit an der Universität Ulm

Vorgelegt von:
Arnim Schindler
arnim.schindler@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert
Dr. Vera Künzle

Betreuer:
Johannes Schobel

2015

Fassung 20. Oktober 2015

Satz: PDF-LaTeX 2ε

Kurzfassung

Die allgegenwärtige Präsenz von mobilen Endgeräten eröffnet neue Wege der Datenerfassung. Daten können nahezu immer und überall aufgezeichnet und verarbeitet werden. Durch die Datenerfassung können einige Alltagsaufgaben vereinfacht und neue Erkenntnisse durch die Analyse dieser Daten gefunden werden. Dazu ist die Datenerfassung über mobile Endgeräte kostengünstig, da ein großer Teil der Infrastruktur von den Benutzern verwaltet, gewartet und bezahlt wird. Dies erfordert jedoch, dass den Anwendern ein einfach zu bedienendes Werkzeug zur Verfügung gestellt wird.

Das in dieser Arbeit beschriebe System stellt ein solches Werkzeug dar. Nach umfassender Anforderungsanalyse wurde ein generisches Formularschema entwickelt, das es erlaubt, unterschiedlichste Fragestellungen abzubilden. Unterstützt wird dies durch eine modulare Architektur, welche die nachträgliche Integration solcher Fragestellungen in das System vereinfacht. Durch eine REST-API wird es ermöglicht, Clients für sämtliche Plattformen zu entwickeln und diese an das System anzubinden. Um den Benutzer an die Eingabe kontextsensitiver Daten zu erinnern, kommt eine ereignisbasierte Steuerung zum Einsatz. Zur Auswertung der erfassten Daten können diese im System aggregiert und aufbereitet oder zur weiteren Verarbeitung exportiert werden.

Inhaltsverzeichnis

1 Einleitung

Mobile Endgeräte, genauer Smartphones und Tablets, sind heute allgegenwärtig. Dadurch ist es möglich, wie nie zuvor, immer und überall Daten zu erfassen, zu sammeln und auszuwerten.

1.1 Problemstellung

Während die permanente Datenerfassung einerseits zu vielen Problemen führt, insbesondere beim Thema Datenschutz, kann sie andererseits zur Vereinfachung alltäglicher Fragestellungen führen. Beispielsweise ist es im medizinischen Bereich häufig notwendig, über einen längeren Zeitraum diverse Vitalparameter eines Patienten aufzuzeichnen. Dies ist zum einen zur Überwachung chronischer Krankheiten (Diabetes oder Bluthochdruck) notwendig, zum anderen zur Diagnose von bisher nicht diagnostizierten

Erkrankungen (Allergien). Doch auch in anderen Domänen ist es hilfreich, systematisch Daten erfassen zu können. Beispiele dafür sind die Aufzeichnung von Fahrtzeiten und -kosten, sowie die Erfassung von Arbeitszeiten. Durch die Auswertung von Daten einer breiten Nutzerbasis können zudem wichtige Erkenntnisse über Zusammenhänge von Ursache und Wirkung, als auch über Verhaltensmuster, gewonnen werden.

1.2 Zielsetzung

Alle diese Daten können mittels Eingabeformularen aufgezeichnet werden, die auf mobilen Endgeräten über den Erfassungszeitraum periodisch ausgefüllt werden. Diese Formulare sind sich von der grundlegenden Struktur her ähnlich, können sich jedoch in der expliziten Implementierung im Ablauf und der Semantik stark unterscheiden. Damit nicht für jedes Formular eine eigene Anwendung entwickelt werden muss, ist ein modulares System wünschenswert, das eine Vielzahl an heterogenen Formularen verwalten kann. Zudem soll der Benutzer von seinem mobilen Endgerät zur Eingabe seiner Daten erinnert werden. Auch die Interaktion mit anderen Benutzern, sowie der Export zur Weiterverarbeitung der Daten, sollen ermöglicht werden.

1.3 Struktur der Arbeit

Zunächst werden in Kapitel 2 einige Grundlagen erklärt, die zum Verständnis der weiteren Arbeit benötigt werden. In Kapitel 3 werden verwandte Arbeiten vorgestellt, die sich im gleichen Themengebiet bewegen. Aus diesen werden in Kapitel 4 Anforderungen definiert und erweitert, die in den folgenden Kapiteln näher besprochen werden. Die Architektur des Systems wird in Kapitel 5 anhand von Zeichnungen des Schemas erläutert. In Kapitel 6 werden schließlich interessante Details zur technischen Umsetzung näher beschrieben. Eine retrospektive Betrachtung der Arbeit, sowie mögliche Erweiterungen, finden letztendlich in Kapitel 7 statt.

2

Grundlagen

In diesem Kapitel werden einige Grundlagen erklärt, die zum Verständnis der weiteren Arbeit notwendig sind. Zudem werden naturgemäß die wichtigsten Begriffe und Technologien beschrieben, die im Rahmen dieser Arbeit verwendet werden.

2.1 Crowd Sensing

Unter *Crowd Sensing* versteht man die flächendeckende Datenerfassung, die durch menschliche Faktoren unterstützt wird. Der Begriff setzt sich aus *Crowd* (engl. für *Menschenmasse*) und *Sensing* (engl. für *Abtastung*) zusammen. In den letzten Jahren hat diese Art der Datenerfassung vor allem durch Smartphones einen enormen Aufschwung erlebt. Sollten bisher flächendeckend und kontinuierlich Daten erfasst werden, mussten weit verteilt Sensoren installiert, gewartet und in eine Infrastruktur zur Übertragung

der Daten eingebunden werden. Nun können für viele Szenarien Smartphones als Erfassungsgerät verwendet werden. Diese werden vom Benutzer gepflegt (Akku laden, Updates), weitflächig verteilt und sind nahezu immer online. Die Datenerfassung geschieht meist passiv im Hintergrund, ohne dass der Benutzer mit dem Gerät interagieren muss oder in der Bedienung des Geräts eingeschränkt wird. Zudem haben solche Geräte den Vorteil, dass sie nicht stationär sind, sondern durch die Mobilität der Menschen eine viel größere Fläche abdecken können. Auf der anderen Seite entstehen dadurch auch Probleme. So schwankt beispielsweise die Qualität der Daten erheblich, wenn sich über einen Zeitraum unterschiedlich viele Smartphones in einem Gebiet befinden [MZY14].

Neben der passiven Datenerfassung können die Daten auch aktiv erfasst werden. Dabei ist der Mensch nicht nur Träger und Transportmittel des Sensors, sondern gibt auch explizit Daten selbst ein. Durch die exzessive Nutzung von Smartphones haben die Menschen nahezu immer ein solches Eingabegerät dabei. Die eingegebenen Daten können durch die eingebauten Sensoren mit weiteren Daten, wie dem aktuellen Standort, angereichert werden. Zudem kann der Träger des Smartphones durch bestimmte Ereignisse alarmiert und zur Eingabe von Daten aufgefordert werden, beispielsweise beim Betreten eines Gebietes. Mit einer solchen ereignisgesteuerten Datenerfassung mit großer Nutzerbasis befasst sich diese Arbeit.

2.2 REST-API

Eine REST-API ist eine Programmierschnittstelle, welche auf HTTP basiert und eine hohe Skalierbarkeit als Ziel hat.

> Client programs use application programming interfaces (APIs) to communicate with web services. Generally speaking, an API exposes a set of data and functions to facilitate interactions between computer programs and allow them to exchange information. [...] The REST architectural style is commonly applied to the design of APIs for modern web services. A Web API conforming to the REST architectural style is a REST API [Mas12].

REST steht für **Re**presentational **S**tate **T**ransfer, was bedeutet, dass die Daten(übertragung) an sich den Zustand des Programms darstellt und dieser nicht explizit gespeichert werden muss. Dies wird möglich, indem bei REST-Anfragen URIs auf Ressourcen abgebildet werden. Das wiederum bedeutet, dass bei einer Anfrage an die selbe URI auch immer die selbe Ressource zurückgegeben wird. Der Webserver muss folglich nichts über den Zustand des Clients wissen, da alle nötigen Informationen in der URI stehen (oder anderweitig über den Request mitgeschickt werden).

Serveranfragen bei REST erfolgen über die HTTP-Methoden `GET`, `POST`, `PUT` und `DELETE`. Dabei werden `GET` und `DELETE` trivial verwendet, um Ressourcen anzufordern oder zu löschen. Die Methoden `POST` und `PUT` werden beide verwendet, um Daten an den Server zu schicken. Jedoch sollte dabei beachtet werden, dass `PUT` laut HTTP/1.1-Standard eine idempotente Methode ist [FGM+99, Sec 9]. Daraus resultiert, dass damit nur vom Server vergebene Ressourcen aktualisiert oder neue Ressourcen angelegt werden dürfen, bei denen der Client die URI selbst bestimmen kann. Hingegen wird `POST` verwendet, wenn eine neue Ressource angelegt werden soll, deren URI der Server vergibt, beispielsweise bei einer Liste [Mas12]. In Abbildung 2.1 wird diese Aussage verdeutlicht. Auf der linken Seite werden drei `POST`-Requests an den Server gesendet, der jeweils eine neue Ressource anlegt. Auf der rechten Seite hingegen werden `PUT`-Requests gesendet. Diese überschreiben die explizit genannte Ressource.

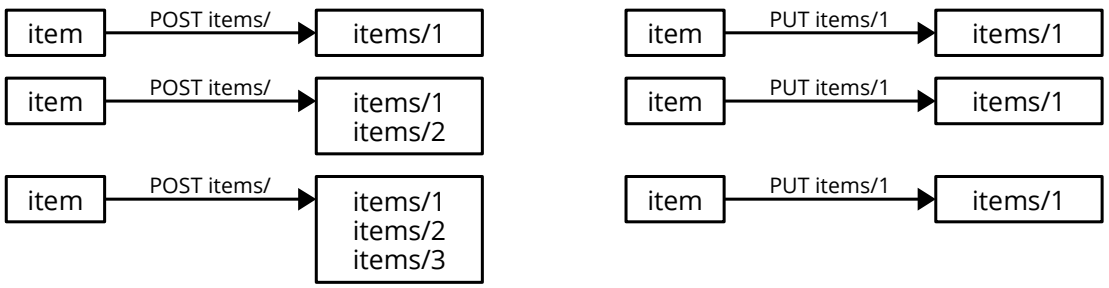

Abbildung 2.1: Senden von Daten mittels REST: `POST` erstellt neue Ressourcen, während sie von `PUT` überschrieben werden.

2.3 Ereignisgesteuerte Systeme

Bei einem ereignisgesteuerten System werden die Systemkomponenten über Events aufgerufen, die als Nachrichten über einen *Event-Bus* (auch *Message Broker*) geschickt werden. Dieser Ablauf ist mit dem Versand von E-Mails vergleichbar: Auf einer Seite werden Nachrichten generiert und verschickt (vgl. E-Mail Programm), der Message Broker (vgl. Mail-Server) verarbeitet diese und sortiert sie in die jeweilige *Message Queue* (Empfänger-Warteschlange, vgl. Posteingang) ein, von der sie der Adressat auslesen und verarbeiten kann. Ein solcher Message Broker ist beispielsweise *RabbitMQ*[1], eine mit Erlang[2] umgesetzte Implementierung des *AMQP-0.9.1*-Protokolls[3]. AMQP ist das **A**dvanced **M**essage **Q**ueuing **P**rotocol und spezifiziert sowohl die Kommunikation zwischen den AMQP-Clients und dem Message Broker, als auch die Semantik der Dienste, die der Broker implementieren muss:

> To enable complete interoperability for messaging middleware requires that both the networking protocol and the semantics of the server-side services are sufficiently specified. AMQP, therefore, defines both the network protocol and the server-side services through a defined set of messaging capabilities [and] a network wire-level protocol [ARA+08].

Das AMQP unterstützt mehrere Arten der Nachrichtenzustellung. Bei der *direkten Zustellung* werden die Nachrichten empfangen und sofort in eine vorgegebene Queue weitergeleitet. *Broadcasting* wird verwendet, um mehrere Empfänger gleichzeitig zu erreichen. Dazu werden die Nachrichten kopiert und in mehrere Queues einsortiert. Mit der Zustellungsart *Topic* können die Nachrichten mit Themen beschrieben und in verschiedene themenbezogene Queues geleitet werden. Die Themen haben eine hierarchische Struktur der Form `a.b.c.···.z`, wobei nach jedem Teilthema gefiltert werden kann. In Abbildung 2.2 wird die Verwendung der Zustellungsart *Topic* am Beispiel eines Logging-Systems gezeigt. Durch den Platzhalter `*` werden alle Themen an dieser Stelle akzeptiert. Der Platzhalter `#` hingegen ersetzt die Themen an beliebig vielen Stellen. Im

[1] `http://www.rabbitmq.com`
[2] `http://www.erlang.org/`
[3] `http://www.amqp.org/specification/0-9-1/amqp-org-download`

Beispiel werden folglich alle Nachrichten, deren Thema mit `log` beginnt, in die untere Queue einsortiert [BS13].

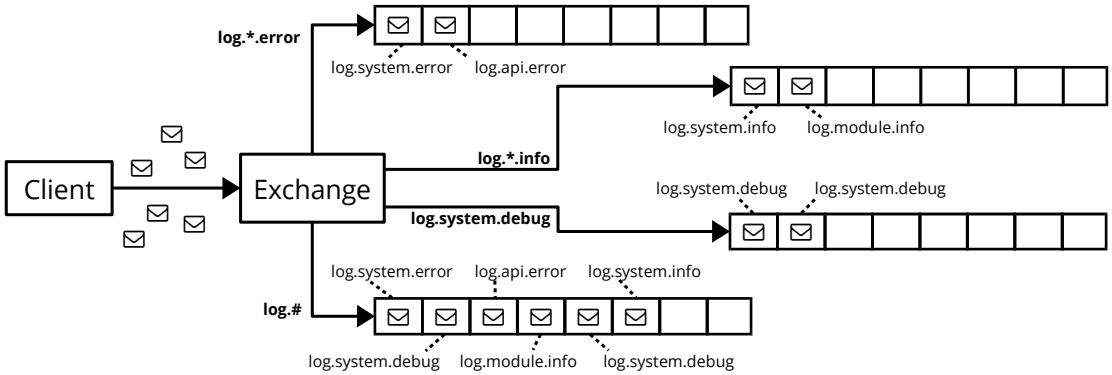

Abbildung 2.2: AMQP-Zustellungsart *Topic*: Nachrichten werden nach Themen in Queues einsortiert. Die Themen können Platzhalter enthalten.

Zudem wird durch das AMQP ermöglicht, *Remote Procedure Calls (RPC)* abzusetzen. Dabei ruft der Client eine Methode (Procedure) auf einem Server (Remote) auf, woraufhin der Server die Anfrage bearbeitet und das Ergebnis an den aufrufenden Client zurückschickt. Im Gegensatz zu den anderen Zustellungsarten ist hierbei die Rücksendung des Ergebnisses ein zentraler Bestandteil der Kommunikation. Um dies zu ermöglichen, erstellt der Client eine eigene Antwort-Queue und teilt dem Server in der Nachricht mit, dass die Antwort an ebendiese Queue geschickt werden muss. Zudem erhält jeder Aufruf eine `Correlation-ID`, die mit der Antwort zurück an den Client geschickt wird. Über diese ID kann die Antwort-Nachricht dem ursprünglichen Aufruf eindeutig zugeordnet werden, auch wenn es auf dem Server zu *Race-Conditions* kommt [BS13]. In Abbildung 2.3 wird ein RPC mit den Antwort-Queues und Correlation-IDs veranschaulicht.

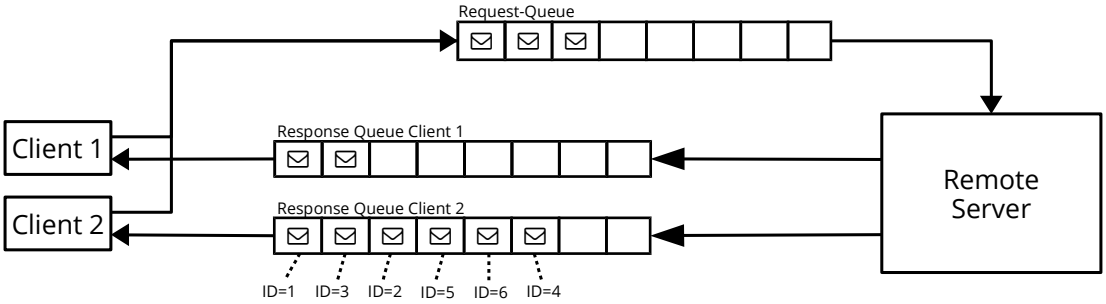

Abbildung 2.3: Remote Procedure Call über AMQP: Jeder Akteur benötigt seine eigene Eingangs-Warteschlange.

2.4 Docker

Docker[4] ist ein Framework, um verteilte Anwendungen zu erstellen und diese zu verwalten. Dazu werden *Docker-Container* erstellt, welche die Anwendungen oder Teile davon enthalten. Die Container starten, im Vergleich zu einer virtuellen Maschine, extrem schnell, da sie direkt im System laufen und keine separate Hardware virtualisiert werden muss. Dennoch sind die Container in sich abgeschlossen, nach außen hin abgeschottet und können auf jedem System gestartet werden, auf dem Docker läuft. Die Docker-Webseite schreibt dazu:

> Docker is an open platform for building, shipping and running distributed applications. [...] Docker containers spin up fast and provide a layer of isolation from other services running in containers. [...] Package your application, dependencies and configurations together to ensure that your application will work seamlessly in any environment on any infrastructure just like it did on your machine [Doc15].

Docker-Container enthalten in der Regel immer nur einen Service, beispielsweise eine Datenbank oder einen Webserver. Um ein System aus mehreren dieser Komponenten zu erstellen, können die Container untereinander verlinkt werden und so gegenseitig auf ihre Dienste zugreifen [Hol15]. Bauen die Container auf dem selben Grundsystem auf, sind sie sehr leichtgewichtig, da das Grundsystem nur ein mal vorhanden sein muss.

[4] `https://www.docker.com/`

Dazu werden nur die Änderungen gespeichert, durch die der Container seiner speziellen Aufgabe zugeführt wird [Han15].

Die Container werden aus sogenannten *Images* erstellt. Diese können entweder fertig vom *Docker-Hub*[5] heruntergeladen oder über ein *Dockerfile* selber gebaut werden. Bekannte Dienste, wie MySQL und Nginx, bieten bereits fertige Images an. Diese können im Bedarfsfall angepasst werden, indem sie durch das bereitgestellte Dockerfile selbst gebaut werden.

Da die Docker-Container in der Regel nur eine Komponente bzw. einen Befehl ausführen, mit dem sie bereits gestartet werden, kann die Software innerhalb des Containers nicht ohne weiteres aktualisiert werden. Zudem ist die Idee hinter Docker, dass die Container nicht verändert werden, sondern von einem Image instanziiert werden, das bereits den gewünschten Anforderungen entspricht. Ein Systemupdate erfolgt konsequenterweise durch den Austausch des ganzen Containers. Um dabei keine Daten zu verlieren, werden diese in sogenannte *data-only-Container* ausgelagert. Wie der Name bereits verrät, werden diese *nur* verwendet, um die Daten zu speichern. Sie enthalten somit keine Komponenten, die später eines Updates bedürften. Ein data-only-Container kann beim Erstellen eines Containers mittels `--volumes-from` referenziert werden, wodurch die freigegebenen Verzeichnisse (*Volumes*) direkt in den Container eingebunden werden [Han15].

In Abbildung 2.4 werden links die beiden Container ohne Verknüpfung gezeigt. Auf der rechten Seite wurde das Volume des `data-only`-Containers im `service`-Container gemountet.

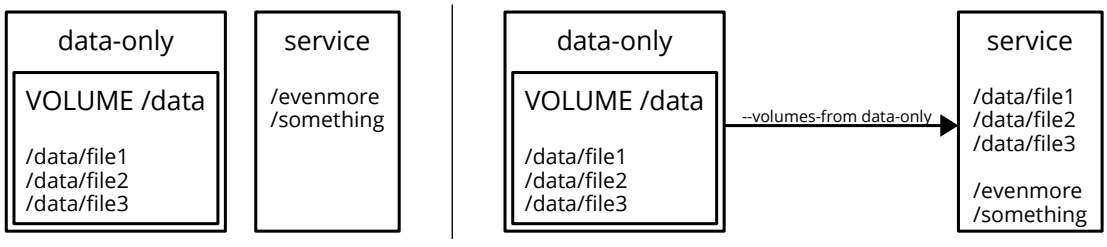

Abbildung 2.4: Die Pfade des Volumes werden in den `service`-Container eingebunden.

[5] `https://hub.docker.com/`

3

Verwandte Arbeiten

Crowd Sensing ist ein vergleichsweise neues Forschungsgebiet. Gerade deshalb gibt es viele Forschungsarbeiten zu diesem Thema. Auch einige kommerzielle Projekte wurden in den letzten Jahren mithilfe mobiler Datenerhebung mit weit verbreiteten Sensoren umgesetzt. Im Folgenden werden einige interessante Projekte aus diesem Gebiet vorgestellt, die direkt oder indirekt zu dieser Arbeit geführt haben. Wie bereits in Abschnitt 2.1 erwähnt, kann beim Crowd Sensing zwischen aktiver und passiver Datenerfassung unterschieden werden.

3.1 Passive Datenerfassung

Obwohl die passive Datenerfassung in dieser Arbeit nur eine untergeordnete Rolle spielt, sollen zwei Projekte aus diesem Bereich nicht unerwähnt bleiben. Sie stellen anschaulich

dar, welches Potential in Crowd Sensing steckt. Zudem zeigen sie interessante Aspekte auf, die bei einer Weiterentwicklung des Systems in Betracht gezogen werden sollten.

3.1.1 Echtzeitverkehrsdaten

Eine prominente Anwendung von passivem Crowd Sensing ist die Echtzeitanzeige der Verkehrslage auf einer Landkarte, wie beispielsweise von Google Maps[1] angeboten wird. Je nach Verkehrsfluss werden die Straßen in verschiedenen Farbtönen dargstellt, von rot (Stau), über gelb (stockender Verkehr) bis grün (fließender Verkehr). Neben der Anzeige von Verkehrshindernissen können Navigationgeräte diese Daten auch dazu verwenden, alternative Routen zu berechnen, damit das Hindernis umfahren werden kann. Auch für Verkehrsleitsysteme ist es von Bedeutung, wo aktuell erhöhtes Verkehrsaufkommen oder stockender Verkehr herrscht. [MR12]

Die Daten für die Verkehrslage werden dabei über mobile Endgeräte ermittelt. Nahezu jeder Verkehrsteilnehmer führt inzwischen ein Smartphone mit sich, das die meiste Zeit mit dem Internet verbunden ist. Auf diese Weise können permanent Positionsdaten an einen Server geschickt werden, der so die aktuelle Verkehrslage berechnet. Aufgrund der hohen Messdichte sind einzelne Ausreißer irrelevant. Außerdem können die gemessenen Daten aufgrund bekannter Vorbedingungen interpoliert werden, da beispielsweise die Straßenführung bekannt ist. Die Positionsdaten selbst werden vom mobilen Endgerät im Hintergrund mittels Triangulation von Mobilfunkantennen, empfangbaren WLANs mit bekanntem Standort oder GPS ermittelt.

Die Ortung des mobilen Endgeräts ist auch in dieser Arbeit von Bedeutung. Jedoch wird das Gerät nicht ständig verfolgt. Nur unter bestimmten Bedingungen wird der aktuelle Standort an den Server übertragen, um ein Ereignis auszulösen (siehe Abschnitt 4.2.1).

[1] `https://maps.google.de`

3.1.2 Blitzortung

Eine weitere Anwendung, die auf passivem Crowd Sensing basiert, ist das Community-Projekt *Blitzortung.org*[2]. Im Gegensatz zu den meisten anderen Crowd Sensing Anwendungen, basiert dieses Projekt nicht auf der Nutzung von Smartphones, sondern auf eigens gebauten Sensoren. Diese erfassen elektromagnetische Wellen in einer Frequenz von 3 kHz-30 kHz, wie für Blitze charakteristisch. Da sich die Wellen sehr weit über die Atmosphäre ausbreiten, können die Blitze über mehrere hundert Kilometer wahrgenommen werden. Dies spricht zunächst dafür, dass einige wenige Sensoren für die Erkennung ausreichen würden. Allerdings sind diese, beispielsweise durch elektrische Felder von Überlandleitungen, sehr störanfällig. Um dennoch relevante Daten zu erhalten, werden ausreichend viele Sensoren ohne Störung benötigt. Des weiteren wird der Einschlagort des Blitzes über Triangulation berechnet. Dies benötigt ebenfalls mehrere Sensoren, um den Standort genau berechnen zu können. Dabei sollten die Messdaten am besten von Sensoren in verschiedenen Himmelsrichtungen relativ zum Einschlagort des Blitzes erfasst werden. [WAV14]

Die Platzierung der Sensoren geschieht über freiwillige Teilnehmer aus aller Welt. Nur dadurch ist eine flächendeckende und für den Betreiber der Plattform kostengünstige Datenerfassung möglich. Auch die Wartung, der Betrieb, sowie die Datenübertragung werden von den Teilnehmern selbst getragen. Lediglich das Filtern nach brauchbaren Sensordaten und die Berechnung des Einschlagortes müssen von der Plattform durchgeführt werden.

Dieses Beispiel ist zum einen interessant, weil es ortsbasiertes Crowd Sourcing auf globaler Ebene zeigt. Während kleinere Bereiche auch noch ohne die Hilfe einer großen Teilnehmerbasis überwacht werden können, ist es ungleich schwieriger, eine derart weitläufige Infrastruktur aufzubauen. Zum anderen zeigt es, dass nicht zwingend mobile Endgeräte benötigt werden, um crowd-basierte Daten zu erfassen. So könnten auch diese Sensordaten von dem System erfasst werden, das in den nächsten Kapiteln beschrieben wird, beispielsweise über eine REST-API (siehe Abschnitt 4.12).

[2] `http://blitzortung.org`

3.2 Aktive Datenerfassung

Während bei der passiven Datenerfassung die Daten im Hintergrund gesammelt und übermittelt werden, müssen die Daten bei der aktiven Erfassung vom Benutzer selbst eingegeben werden. Diese Art der Datenerhebung eignet sich folglich besonders gut, um Meinungen, Erlebnisse und Empfindungen von Menschen zu erfassen.

3.2.1 Track your Tinnitus

Track your Tinnitus[3] ist eine Plattform, welche die Wahrnehmung von Tinnitus über einen langen Zeitraum erfasst. Das Team aus Ärzten, Psychologen und Informatikern hat es sich zum Ziel gesetzt, eine verlässliche und statistisch relevante Datenbasis zu schaffen, wie Patienten im Alltag ihren Tinnitus wahrnehmen. Die Daten sollen zur weiteren Erforschung der Ursachen, Wahrnehmung unter bestimmten Bedingungen, sowie zur Behandlung der Ohrgeräusche dienen.

Die Patienten müssen zuerst drei standardisierte Einführungsfragebögen ausfüllen, bevor die eigentliche Datenerhebung beginnen kann. Da sich die Wahrnehmung des Tinnitus über den Tag durch viele Faktoren ändert, erfolgt die Erfassung mehrmals am Tag zu zufälligen Zeiten. So kann sichergestellt werden, dass über einen längeren Zeitraum statistisch zuverlässige Daten erhoben werden. Der Benutzer bekommt dazu auf seinem Smartphone einen kurzen Fragebogen angezeigt, in den er eintragen kann, wie er den Tinnitus momentan wahrnimmt. Gleichzeitig nimmt die Smartphone-Anwendung im Hintergrund die Lautstärke der Umgebungsgeräusche auf, welche später mit den Benutzereingaben korreliert wird. [PRLS15]

Die Plattform besteht aus jeweils einer mobilen Anwendung für Android und iOS, einem Webinterface sowie einem Backend für die Wissenschaftler. Dadurch ist sie sehr nahe an der Idee des Systems, das in dieser Arbeit beschrieben wird. Besonders die Erfahrungen nach einem Jahr Betrieb von Track your Tinnitus sind von großem Wert. So wurde herausgefunden, dass ein Großteil der Fragebögen von einer Minderheit der Benutzer ausgefüllt wurde. Dabei handelte es sich um die Gruppe mit starkem Tinnitus. Es

[3] https://www.trackyourtinnitus.org

wird erwähnt, dass Maßnahmen zur Motivation der anderen Benutzer ergriffen werden müssen. Des weiteren wird das Interesse anderer Forschungsgruppen hervorgehoben, für die einige Anpassungen an der Plattform vorgenommen werden müssen [PRH+15]. Diesem Sachverhalt versucht das hier beschriebene System durch seine modulare Architektur Rechnung zu tragen (siehe Abschnitt 4.1.1).

3.2.2 Modified Adverse Childhood Experiences (MACE)

Ein weiteres Projekt zur Datenerhebung, Erforschung und Behandlung von medizinischen Datensätzen ist der MACE-Test [TP]. Dieser beschäftigt sich mit der detaillierten Erfassung von traumatischen Kindheitserlebnissen in den ersten 18 Lebensjahren. Basierend auf dem ursprünglichen ACE-Test [DFD+03], ist der MACE-Test eine modifizierte und im Detailgrad erhöhte Version, um differenziertere und umfassendere Daten zu erheben [IRLP+13]. Der Fragebogen wurde zunächst auf Papier ausgefüllt und händisch in eine Datenbank übertragen, was bei den über 2000 Datenfeldern zu erheblichem Aufwand und potentiellen Übertragungsfehlern führte.

Um den Aufwand bei der Durchführung von klinischen Studien zu minimieren, wurde der Fragebogen als iOS-Anwendung für Tablets umgesetzt. Neben dem Übertragungsaufwand konnte auch der Aufwand beim Ausfüllen erheblich minimiert werden, wodurch die emotionale Belastung der Probanden möglichst gering gehalten wird. Der Fragebogen ist in mehrere Abschnitte unterteilt, die nur unter bestimmten Vorbedingungen ausgefüllt werden dürfen. Beispielsweise kann der Fragenkatalog über das Verhältnis zu den Geschwistern nur dann ausgefüllt werden, wenn der Proband in einer einführenden Frage angegeben hat, dass er Geschwister hat. Bei der Papiervariante war es nötig, diese Abschnitte selbst als obsolet zu erkennen, zu überspringen und den Wiedereinstieg an der richtigen Stelle zu finden. Bei der elektronischen Variante hingegen wird der Durchlauf vollständig durch die Ablauflogik der Andwendung gesteuert und überflüssige Fragebogenteile gar nicht angezeigt. [Sch13]

In diesem Umfeld befasst sich auch das Forschungsprojekt QuestionSys[4] mit der Erstellung von elektronischen Fragebögen. Dazu wird ein Framework entwickelt, das deren

[4] http://www.uni-ulm.de/en/in/dbis/research/projects/questionsys.html

kompletten Lebenszyklus abdecken soll. Dieser erstreckt sich von der Modellierung und Installation auf mobilen Endgeräten bis hin zur Datenerhebung und Analyse [SSPR15]. Als Grundlage sollen dabei etablierte Geschäftsprozess-Technologien verwendet werden, die eine flexible Ausführung garantieren. Jedoch müssen diese abstrahiert werden, um Experten aus verschiedenen Domänen (beispielsweise Medizin oder Psychologie) eine einfache Modellierung der Fragebögen zu ermöglichen. Ein solches System könnte die Datenerhebung, insbesondere im klinischen Bereich, durch mobile Anwendungen unterstützen und – wie der MACE-Test – Papierfragebögen überflüssig machen. [SSP+14]

Bei den klinischen Studien handelt es sich nicht direkt um Crowd Sourcing, da sie in vergleichsweise kleinen Gruppen und unter kontrollierten Bedingungen durchgeführt werden. Allerdings zeigt der MACE-Test relevante Aspekte für die Erfassung komplexer Datenstrukturen mithilfe mobiler Endgeräte. Insbesondere die kontextsensitive Benutzerführung ist eine Funktion, die bei künftigen Forschungsarbeiten in diesem Bereich beachtet werden sollte. In Abschnitt 4.10 wird genauer auf den Kontrollfluss eingegangen.

3.2.3 Runtastic

Runtastic[5] ist eine Plattform zur Aufzeichnung von Sportaktivitäten. Die Erfassung der zurückgelegten Strecke erfolgt mit einer Smartphone-Anwendung passiv über GPS. Die Datenerfassung muss jedoch explizit gestartet werden, weshalb sie hier als aktiv angesehen wird. Zudem muss vor der Aufzeichnung die Sportart manuell festgelegt werden. Nach beenden der Aktivität wird ein kurzer Fragebogen angezeigt, über den weitere Informationen erfasst werden können.

Die Sportaktivitäten werden an einen Server übertragen und können für andere Mitglieder der Plattform freigegeben werden. Zudem werden umfangreiche Statistiken erstellt, die teilweise in der Smartphone-Anwendung und vollständig in der Weboberfläche betrachtet werden können.

[5] http://runtastic.com

Das Freigeben (siehe Abschnitt 4.9), sowie das aggregierte Darstellen (siehe Abschnitt 4.8) von Datensätzen, ist auch in dieser Arbeit von Bedeutung. Im Vergleich zur vorgestellten Plattform, bei der nur kollektive Freigaben möglich sind, soll bei der Umsetzung auf die Möglichkeit gezielter Freigaben geachtet werden. Diese und weitere Anforderungen werden im nächsten Kapitel im Detail beschrieben.

4

Anforderungen

In diesem Kapitel werden alle Anforderungen definiert, die das fertige System erfüllen soll. Sie dienen als Leitfaden zur Implementierung und als grundlegende Beschreibung des Systems. Dabei bleibt zunächst unbeachtet, ob diese Vorgaben im Rahmen dieser Arbeit umgesetzt werden – alleine die Aufstellung der Anforderungen sind ein wesentlicher Bestandteil.
Eine Übersicht über den Zusammenhang der Komponenten befindet sich in Kapitel 5, technische Details werden in Kapitel 6 behandelt. Eine Zusammenfassung und ein Ausblick auf das weitere Vorgehen finden in Kapitel 7 statt.

4.1 Modularer Aufbau

Das System soll modular aufgebaut werden, damit einzelne Komponenten einfach hinzugefügt, ausgetauscht, verteilt und ausgelagert werden können.

4.1.1 Systeminterne Module

Im System sollen vom Administrator neue Formulare in Form von Modulen nachinstalliert, sowie von den Benutzern anschließend aktiviert werden können. Dazu muss das System zahlreiche Schnittstellen bereitstellen.

AF 01 Systeminterne Module Das System soll eine Möglichkeit bereitstellen, um in sich abgeschlossene Formulare als Modul installieren zu können. Dies soll über ein Webinterface ermöglicht werden und kein tiefgehendes Wissen über Serverkonfiguration voraussetzen.

AF 02 Mehrkomponenten-Module Das System soll es ermöglichen, dass Module aus mehreren Komponenten bestehen können. Dadurch können beispielsweise mehrseitige Fragebögen erstellt oder Module, welche in mehrere thematisch verwandte Formulare zusammengefasst werden.

4.1.2 Servermodule

Auch das System selbst soll auf dem Server modular installiert werden, um einzelne Teile mit geringem Aufwand austauschen zu können oder diese aus Skalierbarkeitsgründen auf mehreren Servern zu verteilen.

AF 03 Austauschbare Servermodule Das System soll so modular aufgebaut sein, dass auf dem Server einzelne Komponenten ausgetauscht werden können, ohne dass andere Komponenten davon betroffen sind.

AF 04 Skalierbare Architektur Durch die modulare Architektur des Systems soll gewährleistet werden, dass die Module zur Lastverteilung auf mehreren Servern

verteilt werden können. Dafür ist insbesondere eine lose gekoppelte Kommunikation zwischen den Modulen notwendig.

AF 05 Portierbare Module Die Servermodule sollen auch nach der Installation leicht von einem Server zu einem anderen umgezogen werden können. Dabei sollen sowohl die Konfiguration als auch die gespeicherten Daten erhalten bleiben.

4.2 Ereignisbasierte Steuerung

Das System soll ereignisbasiert gesteuert werden, d.h. alle Aktionen werden durch sogenannte Trigger ausgelöst und über einen Enterprise Service Bus (ESB) kommuniziert. Das System muss eine Schnittstelle bereitstellen, an dem sich die Module für die gewünschten Trigger registrieren können.

AF 06 ESB Zur ereignisbasierten Steuerung soll ein Enterprise Service Bus eingebunden werden.

Das System soll drei unterschiedliche Arten von Triggern anbieten:

4.2.1 Externe Trigger

Externe Trigger sind Auslöser von Ereignissen, die von äußeren Faktoren abhängen. Das System soll dabei die zwei wichtigsten Trigger *Ort* und *Zeit* unterstützen.

AF 07 Ort Befindet sich ein Mobilgerät am Ort eines registrierten Location-Triggers, muss sich dieses mit seinem aktuellen Ort und dessen Genauigkeit beim Server melden, damit der Server das entsprechende Ereignis auslösen kann.

AF 08 Ort mit mehreren Geräten Befinden sich mehrere Endgeräte in Benutzung, soll die Aktion nur auf demjenigen ausgeführt werden, welches das Ereignis ausgelöst hat und sich daher am dazugehörigen Ort befindet oder kürzlich befand.

AF 09 Zeit Zeitereignisse sollen vom Server ausgelöst werden.

AF 10 Zeitzonen Um beispielsweise bei einer Reise die Zeitverschiebung beachten zu können, muss der Server über die Zeitzonen der Endgeräte Buch führen. Zu diesem Zweck sollen sich die Endgeräte beim Wechsel der Zeitzone beim Server melden und ihren Status aktualisieren.

AF 11 Zeitzonen mit mehreren Geräten Befinden sich mehrere Endgeräte in verschiedenen Zeitzonen, so sollen nur die Geräte die Aktion ausführen, welche sich in der entsprechenden Zeitzone befinden.

4.2.2 Interne Trigger

Neben den externen Triggern, die auf äußere Faktoren reagieren, soll es auch interne Trigger geben. Diese lösen Ereignisse aus, wenn innerhalb des Systems bestimmte Aktionen ausgeführt werden.

AF 12 Systemereignisse Um über gewisse Systemabläufe zu informieren, soll das System vordefinierte Trigger bereitstellen, auf die von den Endgeräten reagiert werden kann. Hierzu zählen insbesondere:

- Anmelden eines neuen Endgeräts
- Installieren / Deinstallieren eines Moduls
- Ändern von Stammdaten

AF 13 Modulereignisse Module sollen spezifische Ereignisse auslösen können, welche bei bestimmten Aktionen des Moduls auf den ESB gelegt werden können. Andere Module müssen sich für diese spezifischen Ereignisse registrieren können.

4.2.3 Explizite Trigger

Als explizite Trigger gelten alle Systemkomponenten, bei denen der Benutzer selbst Ereignisse generiert.

AF 14 Benutzerinteraktion Alle Aktionen sollen auch unabhängig von automatischen Triggern vom Benutzer selbst ausgeführt werden können.

4.3 Repository

Dem Benutzer soll es ermöglicht werden, seine Module aus einem Respository, ähnlich dem *Apple App Store*[1] bzw. *Google Play Store*[2], herauszusuchen und zu aktivieren. Ein Download des Moduls ist nicht erforderlich, da sich die Programmlogik auf dem Server befindet.

Neben dem Benutzerbereich muss es einen Administrationsbereich geben, in dem die verfügbaren Module verwaltet werden können.

4.3.1 Benutzerbereich

Der Benutzerbereich des Repositories soll es den Benutzern ermöglichen, ein Modul in wenigen einfachen Schritten zu finden und zu aktivieren. Als gängiges Designpattern ist „Feature, Search and Browse“ vorgesehen, welches aus einem groß beworbenen Artikel bzw. Modul (Feature), einem Suchfeld (Search) und einer Auflistung von weiteren Produkten und deren Kategorien besteht.

> Use when: Your site offers users long lists of items — articles, products, videos, and so on — that can be browsed an searched. You want to engage incoming users immediately by giving them something interesting to read or watch. [Tid11]

Demnach werden folgende Anforderungen benötigt:

AF 15 Modulübersicht Damit der Benutzer durch die Modulsammlung stöbern kann, ist eine Übersicht aller Module gewünscht. Die Module sollen mit ihrem Namen, einem Icon, einer kurzen Beschreibung und ggf. weiteren Metadaten übersichtlich dargestellt werden.

AF 16 Mehrseitige Übersicht Um bei sehr vielen Modulen die Übersichtlichkeit zu bewahren, soll die Auflistung in mehrere Seiten unterteilt und paginiert werden.

[1] `https://itunes.apple.com/de/genre/ios/id36?mt=8`
[2] `https://play.google.com`

AF 17 Suchfunktion Es soll eine Suchfunktion geben, welche eine Liste zurückgibt, in der nur die zur Suche passenden Module aufgeführt werden. Auch hier soll die Anzeige bei entsprechend großer Menge an Modulen auf mehrere Seiten verteilt werden (siehe AF 16).

AF 18 Hervorhebung von Modulen Es soll die Möglichkeit bestehen, besonders beliebte bzw. wichtige Module hervorzuheben (Feature).

AF 19 Modulbewertung Um die Beliebtheit von Modulen zu erfassen, soll es ein Bewertungssystem in Form einer Skala von 1 (schlecht) bis 5 (sehr gut) geben.

AF 20 Modulaktivierung Wenn sich der Benutzer für ein Modul entschieden hat, soll er dieses für sich aktivieren können. Im Zuge der Aktivierung soll es möglich sein, die Zugriffsrechte des Moduls individuell einzustellen (siehe Abschnitt 4.4). Nach der Aktivierung soll das Modul sofort zur Verfügung stehen.

AF 21 Moduldeaktivierung Möchte der Benutzer ein Modul nicht mehr benutzen, soll er dieses einfach deaktivieren können. Dabei soll das Modul nur als deaktiviert markiert und anschließend nicht mehr im System angezeigt werden. Daten sollen erhalten bleiben, damit der Benutzer das Modul bei Bedarf reaktivieren kann.

AF 22 Moduldeaktivierung mit Löschen der Daten Möchte der Benutzer ein Modul endgültig nicht mehr benutzen, soll ihm die Möglichkeit angeboten werden, sämtliche Datensätze und Konfigurationen des Moduls aus dem System zu entfernen.

AF 23 Einstellungen zurücksetzen Die Einstellungen der Module sollen jederzeit auf deren Standardkonfiguration zurückgesetzt werden können.

AF 24 Daten löschen Die Daten, die ein Modul anlegt, sollen jederzeit vom Benutzer gelöscht werden können. Hierbei ist auf eine eindeutige Sicherheitsabfrage zu achten, damit das Löschen der Daten nicht durch Unachtsamkeit geschieht.

4.3.2 Administrationsbereich

Im Administrationsbereich sollen die Module, die den Benutzern zur Verfügung stehen, installiert, aktualisiert und deinstalliert werden können. Die Benutzer können nur aus

einer Vorauswahl an Modulen wählen, welche der Administrator für sie im System installiert hat.

AF 25 Modulübersicht im Administrationsbereich Analog zur Modulübersicht im Benutzerbereich soll es diese auch im Administrationsbereich geben. Hierfür kann allerdings die Hervorhebung von Modulen (Feature) entfallen.

AF 26 Versionierung Dem Administrator sollen die verfügbaren Module mit all deren Versionen zur Installation angeboten werden. Je Modul soll nur jeweils eine Version installierbar sein.

AF 27 Installation Durch Auswahl eines Moduls und dessen Version soll dieses nach Bestätigung automatisch vom Remote-Repository heruntergeladen und installiert werden.

AF 28 Up- und Downgrade Es soll eine einfache Möglichkeit geben, die Versionen der Module zu wechseln.

AF 29 Deinstallation Es soll jederzeit möglich sein, ein Modul aus dem System zu entfernen. Es ist dabei nicht vermeidbar, dass das entsprechende Modul bei den Benutzern deaktiviert wird (siehe AF 22).

4.4 Rechteverwaltung für Module

Die Module sollen einer feingranularen Rechteverwaltung unterliegen. So soll der Benutzer selbst bestimmen können, auf welche Stammdaten ein Modul Zugriff bekommt und welche Art von modulspezifischen Daten es empfangen bzw. verschicken kann.

AF 30 Notwendige Rechte Ein Modul soll Rechte als *notwendig* markieren können. Diese Rechte muss der Benutzer dem Modul einräumen, damit es funktionieren kann. Verweigert der Benutzer mindestens eines der benötigten Rechte, so darf bzw. kann das Modul nicht aktiviert werden.

AF 31 Optionale Rechte *Optionale* Rechte hingegen sollen beliebig ein- und ausgeschaltet werden können. Die Module müssen sich dem Zustand der optionalen Rechte bewusst sein und entsprechend reagieren.

AF 32 Nachrichtenarten Als schnelle Lösung für den Benutzer sollen ganze Kategorien an Nachrichten ein- und ausgeschaltet werden können. Diese sind im Speziellen:

- Push-Nachrichten verschicken
- Nachrichten an das Postfach des Benutzers im System verschicken
- Modulspezifische Daten verschicken / empfangen

AF 33 Modulspezifische Daten Die modulspezifischen Daten, welche das System senden und empfangen kann, sollen nach Modulen und Themen sortiert aufgelistet werden. So kann der Benutzer einfach auswählen, welche Datenflüsse er zulassen möchte.

AF 34 Stammdaten Der Benutzer soll genau auswählen können, welche Stammdaten ein Modul lesen oder schreiben kann.

4.5 Nutzerverwaltung

Das System soll für mehrere Benutzer mit jeweils eigenen Benutzerkonten zugänglich sein. Diese sollen auf zwei Ebenen verwaltet werden können: zum einen vom Benutzer selbst, zum anderen von einem Administrator.

4.5.1 Benutzerbereich

Der Benutzer soll all seine Einstellungen selbst vornehmen können. Dafür soll es einen separaten Benutzerbereich geben, der nur dem Zweck der Kontoverwaltung dient und die folgenden Anforderungen erfüllt:

AF 35 Registrierung Dem Benutzer soll es ermöglicht werden, sich selbst mit einer Mailadresse, einem Passwort und ggf. weiteren Stammdaten am System zu registrieren.

AF 36 Passwort ändern Der Benutzer soll jederzeit die Möglichkeit haben, sein Passwort zu ändern. Das neue Passwort muss vom System auf eine gewisse Mindestlänge bzw. -sicherheit überprüft werden.

AF 37 Passwort zurücksetzen Hat der Benutzer sein Passwort vergessen, soll er es über seine Mailadresse zurücksetzen können. Um Missbrauch zu verhindern, soll das Passwort nicht direkt geändert, sondern dem Benutzer eine E-Mail mit einem Bestätigungs-Link zur Verifizierung gesendet werden.

AF 38 Mailadresse ändern Der Benutzer soll seine Mailadresse selbständig ändern können. Die Änderung darf erst übernommen werden, wenn die neue Mailadresse durch den Bestätigungs-Link in einer an diese Adresse geschickte E-Mail verifiziert wurde.

AF 39 Stammdaten ändern Es soll ein Formular zur Verfügung gestellt werden, über das der Benutzer seine Stammdaten (Name, Geburtstag, etc.) ändern kann.

AF 40 Account löschen Der Benutzer soll die Möglichkeit haben, seinen Account zu löschen. Dabei sollen sämtliche mit ihm verknüpften Daten ebenfalls aus dem System entfernt werden.

AF 41 Personensuche Dem Benutzer soll eine Personensuche zur Verfügung gestellt werden, über die er andere Benutzer anhand deren Mailadresse oder Namen suchen kann, sofern es deren Sicherheitseinstellungen zulassen.

AF 42 Adressbuch Das System soll es ermöglichen, dass der Benutzer seine Kontakte abspeichern kann und nicht für jede Interaktion mit ihnen neu suchen muss (siehe AF 41).

AF 43 Benutzergruppen Die Kontakte sollen in verschiedene Benutzergruppen eingeteilt werden können. Diese dienen einerseits zur Organisation vieler Kontakte, andererseits zur gruppenweisen Zuteilung von Rechten oder Freigaben, siehe Abschnitt 4.9.

4.5.2 Administrationsbereich

Parallel zum Benutzerbereich soll es auch im Administrationsbereich die Möglichkeit zur Kontoverwaltung geben. Allerdings soll der Administrator Zugriff auf alle Konten haben. Im Gegensatz zur Kontoverwaltung durch den Benutzer ist die Möglichkeit zur Änderung der Stammdaten im Administrationsbereich nicht erforderlich, auch ein Adressbuch ist nicht nötig. Es bleiben die nachfolgend aufgelisteten drei Anforderungen:

AF 44 Nutzer anlegen Der Administrator soll in der Lage sein, Benutzerkonten anzulegen. Es soll ein geheimes Passwort generiert werden, das nicht per E-Mail verschickt, sondern nur der Kontosicherheit dient. Stattdessen soll dem Benutzer eine E-Mail mit einem Link zum Zurücksetzen des Passworts geschickt werden.

AF 45 Nutzer sperren Benutzerkonten sollen vom Administrator gesperrt werden können. Alle Daten sollen dabei erhalten bleiben, allerdings soll der Benutzer nicht mehr mit dem System interagieren können, bis sein Konto wieder entsperrt wird.

AF 46 Nutzer löschen Der Administrator soll Benutzerkonten entfernen können. Dabei sollen alle mit dem Konto verknüpften Daten gelöscht werden.

AF 47 Passwort zurücksetzen Der Administrator soll das Passwort zurücksetzen können. Dies soll – wie beim Anlegen eines neuen Nutzers (siehe AF 44) – durch ein temporäres geheimes Passwort und eine E-Mail an den Benutzer geschehen, die einen Bestätigungs-Link zum Zurücksetzen des Passworts enthält.

4.6 Mehrsprachigkeit

Alle Systemkomponenten sollen Mehrsprachigkeit unterstützen, um eine internationale Nutzerbasis zu erreichen.

AF 48 Mehrsprachigkeit Der Client soll dem System seine bevorzugte Sprache mitteilen können, woraufhin der Server den angeforderten Inhalt in der entsprechenden Sprache ausliefert.

AF 49 Fallback-Sprache Wird vom Client keine Sprachinformation an den Server geschickt oder steht die Ressource nicht in der gewünschten Sprache zur Verfügung, soll der Inhalt in einer vom Administrator konfigurierbaren Fallback-Sprache ausgeliefert werden.

4.7 Messaging Center

Das Messaging Center soll als zentrale Komponente, sowohl zur Kommunikation zwischen System und Benutzer als auch der Benutzer untereinander, dienen. Hierfür ist eine Möglichkeit zum Empfangen und Versenden von Nachrichten nötig. Außerdem sollen auch alle Kommunikationseinstellungen im Messaging Center angeboten werden.

4.7.1 Senden und empfangen

Im Messaging Center sollen alle eingehenden Nachrichten übersichtlich dargestellt werden. Außerdem soll der Benutzer in der Lage sein, selbst Nachrichten an andere Benutzer zu verschicken.

AF 50 Nachrichten verfassen Der Benutzer soll ein Formular angezeigt bekommen, über das er Nachrichten an andere Benutzer schicken kann. Zur Auswahl des Empfängers soll ihm das Adressbuch aus Abschnitt 4.5.1 zur Verfügung stehen (siehe AF 42).

AF 51 Nachrichten von Benutzern Das Messaging Center soll einen Posteingang enthalten, in dem alle eingehenden Nachrichten von Benutzern gesammelt, gelesen und beantwortet werden können.

AF 52 Nachrichten von Modulen Auch Module sollen Nachrichten an den Benutzer senden können. Diese sollen in einem separaten Posteingang für Modulnachrichten gesammelt werden.

4.7.2 Einstellungen

Im Einstellungsbereich soll der Benutzer alle kommunikationsrelevanten Einstellungen vornehmen können.

AF 53 Benutzerkommunikation Der Benutzer soll entscheiden können, von wem er Nachrichten erhalten möchte. Dazu soll er sowohl gruppen- als auch personenbasiert einstellen können, von wem das System Nachrichten erlauben soll. Zur gruppenbasierten Zuordnung sollen die Benutzergruppen aus dem Adressbuch verwendet werden (siehe AF 42).

AF 54 Intermodulare Kommunikation Da die Einstellungen für die Kommunikation zwischen Modulen semantisch sowohl in die Moduleinstellungen (Abschnitt 4.4) als auch ins Messaging Center gehören, soll dem Benutzer die Einstellung der zu versendenden und empfangenden Nachrichtenarten in beiden Systemkomponenten angezeigt werden. Es sollte ein Hinweis platziert werden, dass es sich um dieselben Einstellungen handelt.

4.8 Reports

Die von den Modulen erfassten Daten sollen übersichtlich aufbereitet, angezeigt, gedruckt und zur weiteren Verarbeitung exportiert werden können. Als zentrale Systemkomponente für diese Aufgaben soll der Reports-Manager dienen.

AF 55 Diagramme Der Reports-Manager soll in der Lage sein, die gegebenen Daten in Diagrammen aufzubereiten. Mindestens die drei wichtigsten Diagrammarten sollen unterstützt werden:

- Liniendiagramm für Verläufe
- Balkendiagramm für Verläufe oder Größenvergleiche
- Kreisdiagramm für Größenvergleiche

AF 56 Export als PDF Alle Reports sollen als PDF exportiert werden können. Damit wird eine explizite Druckfunktion überflüssig.

AF 57 Export als JSON Damit andere Systeme die Daten einfach weiterverarbeiten können, sollen diese als JSON exportiert werden können. Die JSON-Datei soll alle relevanten Daten enthalten, die benötigt werden, um einen vollständigen Report inklusive Diagramme erstellen zu können.

4.9 Freigabe von Daten

Bei einigen Datensätzen kann es für den Benutzer sinnvoll sein, diese mit anderen Personen zu teilen. Ein Anwendungsfall wäre der Verlauf eines regelmäßig erfassten Vitalparameters (z.B. Blutdruck), den ein Patient für seinen Arzt freigeben möchte. Um sämtliche Freigaben im System soll sich ein Share-Manager kümmern.

AF 58 Freigabe einzelner Ergebnisse Das System soll es ermöglichen, einzelne Ergebnisse freizugeben. Mit einem einzelnen Ergebnis ist der vollständige Datensatz gemeint, der von einem (mehrseitigen) Formulardurchlauf eines Moduls erhoben wird. Damit der Datensatz auch von Benutzern eingesehen werden kann, die das entsprechende Modul nicht aktiviert haben, soll implizit ein Report erstellt und freigegeben werden.

AF 59 Freigabe von Reports Der Benutzer soll vom System unterstützt werden, mehrere Ergebnisse in einem Report zusammenzufassen und diese an einen oder mehrere Benutzer freizugeben.

AF 60 Ein- und ausgehende Freigaben Da ein Benutzer sowohl Freigaben versenden als auch empfangen können muss, soll der Share-Manager entsprechend in zwei Bereiche unterteilt werden.

AF 61 Freigaben widerrufen Der Share-Manager muss das Aufheben von Freigaben unterstützen. Danach dürfen die (dann nicht mehr) freigegebenen Daten beim Empfänger nicht mehr angezeigt werden.

4.10 Formulare

Ein wesentlicher Bestandteil des Systems sind Formulare zur Erfassung der Daten. Die Formulare sollen von den Modulen bereitgestellt und von den Endgeräten angezeigt werden. Durch die Unterschiede der Client-Plattformen und möglicherweise verschiedenen Versionen der Formularbeschreibung sind einige Anforderungen zu erfüllen, auf die im Folgenden eingegangen wird.

AF 62 Beschreibungssprache Die Formulare der Module sollen auf den Endgeräten aus nativen Systemelementen zusammengebaut werden. Dazu müssen die Formulare als Beschreibungssprache an die Endgeräte übergeben und dort geparsed werden. Damit die volle Funktionalität der Formulare auf allen Endgeräten gewährleistet werden kann, müssen sich sowohl die Module als auch die Endgeräte an genaue Spezifikationen halten.

AF 63 Versionierung Die Funktionalität der Formulare wird sich über die Lebenszeit des Systems ändern. Um Konflikte zwischen verschiedenen Versionen zu vermeiden, muss immer die entsprechende Versionsnummer der Beschreibungssprache übermittelt werden.

AF 64 Datentypen Verschiedene Datentypen erfordern verschiedene Eingabefelder. Damit die Clients entsprechende Eingabemöglichkeiten bereitstellen können, müssen die Datentypen vorher genau definiert werden. Mindestens folgende Datentypen sollen bereitgestellt und genau spezifiziert werden:

- **Text** für kurze, einzeilige Texteingaben
- **Mengentext** für lange, mehrzeilige Texteingaben
- **Passwort** für die verdeckte Eingabe von Passwörtern
- **Zahl** für die Eingabe von Zahlen
- **Datum und Zeit** für die Auswahl von Datum und Zeit mit Zeitzone mithilfe eines Kalenders bzw. Spinning-Wheels
- **E-Mail** für gültige Mailadressen

- **Auswahl** für die Auswahl *eines* Wertes aus einer Liste mehrerer Vorgaben
- **Mehrfachauswahl** für die Auswahl *eines oder mehrerer* Werte aus einer Liste mit mehreren Vorgaben
- **Checkbox** zum Ein- und Ausschalten einer Option

AF 65 Beschriftung Alle Felder sollen beschriftet werden, damit der Benutzer sofort erkennen kann, wofür das entsprechende Feld vorgesehen ist.

AF 66 Komplexe Datentypen Das System soll es ermöglichen, dass mehrere Felder mit beliebigen einfachen Datentypen (siehe AF 64) zu komplexen Datentypen zusammenzusetzen. Dies soll zum einen der optischen Gruppierung dienen, zum anderen der Wiederverwendbarkeit von Feldgruppen.

AF 67 Pflichtfelder Einerseits müssen nicht immer alle Eingabefelder ausgefüllt werden, andererseits gibt es Daten, die für die Funktion eines Moduls notwendig sind. Es soll möglich sein, die Eingabefelder entsprechend zu markieren.

AF 68 Antwortschema Damit das System die Antworten von allen Endgeräten in gleicher Weise verarbeiten kann, muss das Antwortschema spezifiziert und von allen Beteiligten eingehalten werden.

AF 69 Kontrollfluss Es sollen alternative Ablaufpfade durch mehrseitige Formulare möglich sein. Diese sollen sowohl durch die Module als auch durch den Benutzer gesteuert werden können. Die Module sollen anhand der bereits eingegebenen Daten entscheiden können, welche Formulare dem Benutzer im nächsten Schritt angeboten werden. Gibt es mehrere Formulare zur Auswahl, sollen dem Benutzer alle Möglichkeiten zur Wahl angeboten werden.

4.11 Eingabevalidierung

Die Benutzereingaben sollen an zwei Stellen überprüft werden: zum einen auf den Endgeräten selbst, um dem Benutzer gleich bei der Eingabe ein Feedback zu geben; zum anderen auf dem Server, um mögliche Implementierungsfehler oder Manipulationsversuche des Clients zu erkennen und die Datenintegrität zu gewährleisten.

AF 70 Datentypspezifische Validierung Die verschiedenen Datentypen haben verschiedene spezifische Eigenschaften und Einschränkungen, die bei der Eingabe überprüft werden müssen. So dürfen z.B. in ein Datumsfeld nur gültige Daten eingegeben werden, ein Textfeld kann auf eine maximale Länge beschränkt werden und der Wert in einem Nummernfeld soll bei Bedarf zwischen einem Minimum und einem Maximum liegen.

AF 71 Reguläre Ausdrücke Jedem Formularfeld soll ein regulärer Ausdruck zugewiesen werden können, der die Vorgaben des entsprechenden Datentyps zusätzlich einschränkt. So kann beispielsweise ein Formularfeld erstellt werden, das nur hexadezimale Zahlen zulässt, indem ein Textfeld durch den regulären Ausdruck `^[0-9a-f]*$` eingeschränkt wird.

AF 72 Validierung von Pflichtfeldern In AF 67 wird die Unterscheidung zwischen optionalen und notwendigen Feldern gefordert. Diese Anforderung muss bei der Eingabevalidierung berücksichtigt werden. Insbesondere muss bei komplexen Datentypen (siehe AF 66) beachtet werden, dass sich die Ausfüllpflicht der untergeordneten Felder auf die anderen Elemente des komplexen Datentyps auswirkt. Das genaue Verhalten wird in Kapitel 6 spezifiziert und beschrieben.

AF 73 Fehlermeldungen Erkennt der Server eine Verletzung der Validierungsrichtlinien, soll dieser den Empfang des ausgefüllten Formulars ablehnen und eine Fehlermeldung an den Client schicken. Damit der Client adäquat reagieren kann, soll die Fehlermeldung die Art des Fehlers sowie den eindeutigen Namen des ungültigen Feldes enthalten.

4.12 API

Da die Architektur aus dem Serversystem und beliebigen Clients besteht, müssen die Serverfunktionen über eine universelle API ansprechbar sein.

AF 74 REST-API Als Kommunikationsprotokoll zwischen Server und Client soll eine REST-API über HTTP(S) zum Einsatz kommen. Dies wird von allen gängigen Plattformen für Mobilgeräte ohne Weiteres unterstützt.

AF 75 Authentifizierung Die Aufrufe der API dürfen nur nach vorheriger Authentifizierung und Autorisierung am System möglich sein, um persönliche Daten zu schützen und Missbrauch zu verhindern.

In Kapitel 4 wurden die Anforderungen erfasst, aufgelistet und detailliert beschrieben. Sie dienen als Grundlage für die Entwicklung des Systems, auf das in den nächsten Kapiteln tiefer eingegangen wird.

5

Architektur

Die Architektur des Systems besteht aus zahlreichen modularen Komponenten, die über ein Bus-System miteinander kommunizieren. In diesem Kapitel wird zunächst ein Überblick über die gesamte Architektur des Servers gegeben. Im Anschluss werden die grundlegenden Aufgaben der einzelnen Komponenten erklärt, bevor abschließend die wichtigsten Kommunikationsabläufe in der Architektur dargestellt werden. Die genaue Implementierung der Komponenten und die Kommunikation über den Event-Bus werden in Kapitel 6 beschrieben.

5.1 Übersicht

Eine Übersicht der Systemarchitektur wird in Abbildung 5.1 gezeigt. In der Mitte befindet sich als Kommunikations-Backbone ein Enterprise Service Bus, über den der

Nachrichtenaustausch zwischen den Servermodulen (*Webservice Manager*, *Report Manager*, usw.) abgewickelt wird. Als zweiter Kommunikationskanal dient die Datenbankverbindung, die von jedem Servermodul selbst aufgebaut werden muss. Dadurch kann theoretisch jedes Servermodul seine eigene Datenbank erhalten, wofür in der Standardkonfiguration allerdings keine Notwendigkeit besteht.

Die in der abgebildeten Architektur eingezeichneten Servermodule werden im System als vorhanden vorausgesetzt. Sie können durch ihre Abgeschlossenheit zwar beliebig ausgetauscht werden, müssen ihre Schnittstellen jedoch vollständig und korrekt implementieren. Die systeminternen Module (M_1, M_2, ..., M_x) hingegen sind optional und werden vom *Module Manager* verwaltet. Im folgenden Kapitel werden die einzelnen Komponenten genauer beschrieben und deren Funktionalität erläutert.

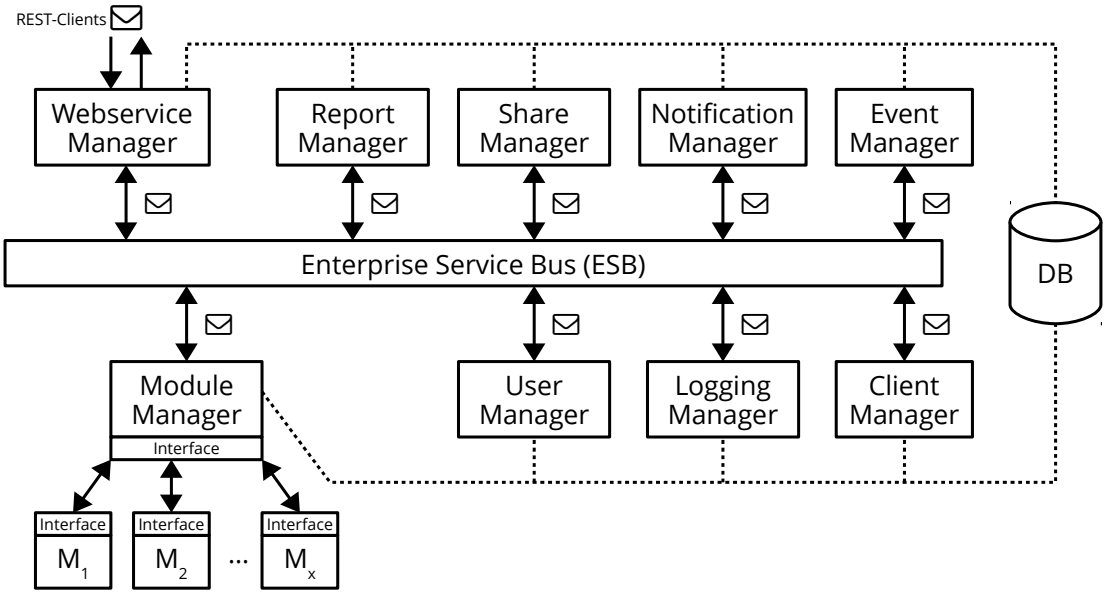

Abbildung 5.1: Übersicht der Architektur: Die neun Hauptkomponenten kommunizieren mit dem ESB und haben parallel eine separate Datenbankverbindung. Oben links in der Abbildung ist die REST-Schnittstelle eingezeichnet.

5.2 Komponenten

Das System besteht auf Serverseite aus neun Hauptkomponenten, die größtenteils über den ESB miteinander kommunizieren.

5.2.1 Enterprise Service Bus

Der Enterprise Service Bus (ESB) dient als zentraler Kommunikationskanal zwischen den einzelnen Servermodulen (siehe Abbildung 5.1). Diese müssen sich alle mit einem eigenen *Listener* am ESB anmelden, um die für sie bestimmten Nachrichten empfangen zu können. Die Logik zur Übermittlung der Nachrichten übernimmt der ESB. Sollte ein Servermodul vorübergehend nicht verfügbar sein, kümmert sich der ESB auch um das Zwischenspeichern der Nachrichten bzw. lehnt den Versand der Nachricht ab, falls es sich um eine Nachricht handelt, die in Echtzeit verarbeitet werden muss.

Eine andere Aufgabe des Enterprise Service Busses ist die ereignisbasierte Steuerung der systeminternen Module (siehe Abschnitt 4.2). Durch den ESB können die Module lose gekoppelt sein, d.h. sie müssen nicht über die anderen Module Bescheid wissen. Stattdessen kommunizieren sie lediglich über den *Module Manager* (siehe Abschnitt 5.2.7) mit dem ESB, der die Ereignisse entgegen nimmt und an die entsprechenden Module verteilt.

5.2.2 Webservice Manager

Der Webservice Manager ist die zentrale Komponente zur Kommunikation mit den Clients. Alle API-Anfragen, die von außerhalb des Servers kommen (beispielsweise von mobilen Endgeräten), werden vom Webservice Manager entgegengenommen, geprüft und über den ESB an das entsprechende Servermodul weitergereicht. Nach der Verarbeitung wird das Ergebnis vom Webservice Manager an den Client zurückgegeben. Da die API-Anfragen nur von autorisierten Benutzern angenommen werden dürfen (siehe Abschnitt 4.12), muss sich der Webservice Manager zudem um die Authentifizierung der

Clients kümmern. Ein vollständiger Durchlauf einer API-Anfrage wird in Abschnitt 5.3 genauer erläutert.

5.2.3 Report Manager

Der Report Manager ist das Servermodul, das für die Erstellung und Verwaltung von Reports zuständig ist (siehe Abschnitt 4.8). Es wird für gewöhnlich vom Webservice Manager über den ESB aufgerufen und fordert Daten von systeminternen Modulen über den Module Manager an. Möchte der Benutzer über den Share Manager einzelne Datensätze freigeben, so wird der Report Manager von diesem aufgerufen (siehe AF 58). Es ist folglich essenziell, dass die Nachrichten auf dem ESB das aufrufende Modul als Absender enthalten und die Antwortnachrichten an dieses Modul adressiert werden.

5.2.4 Share Manager

Sämtliche Freigaben im System (siehe Abschnitt 4.9) werden vom Share Manager verwaltet. Er wird vom Webservice Manager aufgerufen und fordert die freizugebenden Daten vom Report Manager an. Alternativ müssen zunächst die Daten vom Module Manager angefordert und an den Report Manager weitergeleitet werden, wenn der Benutzer explizit einen einzelnen Datensatz freigeben möchte. Um die möglichen Empfänger der Freigabe zu ermitteln, muss der User Manager (siehe Abschnitt 5.2.8) hinzugezogen werden.

Neben dem Erstellen neuer Freigaben gehören auch das Löschen und Aufrufen vorhandener Freigaben zu den Aufgaben des Share Managers. Auch diese Modulteile werden vom Webservice Manager aufgerufen und liefern eine Übersicht der vorhandenen Freigaben zurück. Dabei ist wichtig, dass der Benutzer nur die für ihn oder von ihm freigegebenen Datensätze einsehen kann. Die Kommunikation mit dem User Manager ist dabei nicht notwendig, da das System durch die Authentifizierung bereits weiß, von welchem Benutzer die Anfrage kommt und dieser beim Anlegen der Freigaben in den Datensatz gespeichert wurde.

5.2.5 Notification Manager

Vom Notification Manager werden alle vom System unterstützten Möglichkeiten zum Senden und Empfangen von Nachrichten angeboten. Dabei sind jedoch nicht die Systemnachrichten auf dem ESB gemeint, sondern lediglich Mitteilungen, die explizit an den Benutzer oder den Client gerichtet sind. Dies betrifft zum einen die Servermodule, die dem Benutzer etwas mitteilen möchten, zum anderen die Kommunikation zwischen Benutzern.

Das Benutzer-Frontend auf den Clients, das von diesem Servermodul mit Inhalten versorgt wird, wird Messaging Center (siehe Abschnitt 4.7) genannt. Es zeigt die empfangenen Nachrichten an und verfügt über eine Eingabemaske zum Erstellen von Nachrichten. Zudem kann der Benutzer das Verhalten des Notification Managers über die Konfigurationskomponente im Messaging Center einstellen. Um all diese Funktionen bereitstellen zu können, muss der Notification Manager über den Webservice Manager mit den Endgeräten kommunizieren können und umgekehrt.

Eine weitere Komponente des Notification Managers ist der Versand von Nachrichten an externe Dienste wie E-Mail oder SMS. Bei der Mailadresse und der Telefonnummer handelt es sich um Stammdaten des Benutzers. Daher müssen diese über den User Manager konfiguriert und vom Notification Manager angefordert werden. Alternativ werden sie vom Benutzer im Konfigurationsmodul des Messaging Centers eingetragen und anschließend vom Notification Manager im User Manager aktualisiert. In beiden Fällen muss der Notification Manager über den ESB mit dem User Manager kommunizieren.

Als zusätzliche Aufgabe wird der Notification Manager zum Versenden von Push-Nachrichten (siehe AF 32) an die Clients verwendet. Diese werden entweder beim Eingang neuer Nachrichten vom Notification Manager selbst ausgelöst oder vom Module Manager, falls ein systeminternes Modul ausgeführt werden soll.

5.2.6 Event Manager

In Abschnitt 4.2 werden Ereignisse beschrieben, die das System unterstützen soll. Während die internen (siehe Abschnitt 4.2.2) und expliziten (siehe Abschnitt 4.2.3)

Trigger durch Module bzw. den Benutzer ausgelöst und alleinig durch die Logik des ESBs verarbeitet werden können, müssen die externen Trigger (Abschnitt 4.2.1) gesondert behandelt werden.

Der Event Manager führt eine Liste über alle Zeit- und Ort-Trigger und löst diese im Bedarfsfall aus. Außerdem ist er in der Lage, die Zeitzonen der angemeldeten Geräte umzurechnen und die Ereignisse entsprechend nur an diese Geräte zu adressieren. Die am System angemeldeten Clients erfährt der Event Manager vom Client Manager (siehe Abschnitt 5.2.10). Die Ereignisse, die vom Event Manager ausgelöst werden, werden auf den ESB gelegt und von den Servermodulen abgearbeitet. Soll ein Modul auf den Clients aufgerufen werden, so muss sich der Modul Manager darum kümmern, dass die Endgeräte eine Push-Nachricht erhalten und das Modul aufrufen. Dieser Ablauf wird in Abschnitt 5.3.3 genau erläutert.

Ort-Trigger kann der Event Manager nicht eigenständig auslösen, da die Endgeräte ihren Standpunkt nicht permanent an den Server übermitteln. Um diesem den Standpunkt nur im Bedarfsfall mitzuteilen, müssen die Endgeräte über alle für sie in Frage kommenden Ort-Trigger Bescheid wissen. Dazu können die Clients eine Liste dieser Trigger über die REST-API beim Event Manager anfordern. Wenn sich die Trigger ändern, werden die Clients über eine Push-Nachricht aufgefordert, ihre lokale Liste zu aktualisieren.

5.2.7 Module Manager

Der Module Manager dient als Schnittstelle zwischen systeminternen und Servermodulen. Er ermöglicht ein konsistentes Aufrufen und eine Kapselung der systeminternen Module, die von verschiedenen Anwendungsentwicklern programmiert werden können. Sämtliche Interaktionen mit den systeminternen Modulen müssen über den Module Manager abgewickelt werden, der die Anfragen überprüft und gegebenenfalls abweist. Eine genaue Beschreibung eines Modulaufrufs vom Client wird in Abschnitt 5.3 gezeigt.

Auch das Modulrepository (siehe Abschnitt 4.3) ist eine Komponente des Module Managers. Das Repository dient einerseits der Installation und Wartung von Modulen auf dem Server durch einen Administrator (siehe Abschnitt 4.3.2), andererseits stellt es die

Inhalte für das Frontend bereit, mit dem der Benutzer die Module für sich aktivieren und verwalten kann (siehe Abschnitt 4.3.1).

5.2.8 User Manager

Der User Manager wird für die Benutzerverwaltung benötigt und ist in einen Administrationsbereich, einen Benutzerbereich und einen systeminternen Bereich unterteilt. Die beiden erstgenannten Teile werden über den Webservice Manager angesprochen, der systeminterne Bereich wird von den anderen Servermodulen zur Abfrage von Benutzerinformationen verwendet. Zudem stellt der interne Bereich ein Interface zur Verfügung, über das die Servermodule Stammdaten abfragen und ändern können (siehe Abschnitt 5.2.5).

Über den Benutzerbereich kann der Benutzer alle Aktionen durchführen, die mit seinem Benutzerkonto zu tun haben (siehe Abschnitt 4.5.1). Dazu zählen sowohl die Registrierung am System, als auch das Ändern der Stammdaten. Auch für das Löschen des Accounts ist der User Manager zuständig.

Der Administrationsbereich ist weitgehend gleich zum Benutzerbereich. Allerdings ist er auf die nötigsten Funktionen (Erstellen, Anzeigen und Löschen von Benutzern) eingeschränkt, kann dafür auf alle Benutzerkonten zugreifen. Auf den mobilen Endgeräten muss der Administrationsbereich nicht implementiert werden, da die Administrationsaufgaben primär über die Plattform durchgeführt werden können und sollen. Jedoch steht es Anwendungsentwicklern frei, den Administrationsbereich auch auf mobilen Endgeräten zu implementieren, da dieser, wie oben erwähnt, auch über den Webservice Manager aufgerufen wird.

5.2.9 Logging Manager

Der Logging Manager ist für die Log-Daten des kompletten Systems zuständig. Alle Systemkomponenten können Log-Nachrichten auf den ESB schicken, die vom Logging Manager abgerufen, einsortiert, abgespeichert und vom Administrator ausgelesen wer-

den können. Der Logging Manager kann acht Log-Levels unterscheiden, die in RFC 5424 aufgeführt werden [Ger09]:

1. **Emergency**: system is unusable
2. **Alert**: action must be taken immediately
3. **Critical**: critical conditions
4. **Error**: error conditions
5. **Warning**: warning conditions
6. **Notice**: normal but significant condition
7. **Informational**: informational messages
8. **Debug**: debug-level messages

5.2.10 Client Manager

Der Client Manager führt über die verfügbaren Endgeräte und deren Eigenschaften Protokoll. Dazu zählen der ungefähre Standort, die Zeitzone und das Betriebssystem. So kann sich der Benutzer einen Überblick über seine kürzlich angemeldeten Geräte verschaffen.

Die wichtigste Aufgabe des Client Managers ist es, den Event Manager (siehe Abschnitt 5.2.6) zu informieren, wenn ein Client die Zeitzone wechselt bzw. ein neuer Client mit einer bisher unbeachteten Zeitzone hinzukommt. Dadurch können die Trigger entsprechend angepasst werden.

Als Nebeneffekt der Protokollierung kann der Benutzer eine Liste aller kürzlich von ihm verwendeten Endgeräte abrufen, sofern es sein Client unterstützt.

5.2.11 Datenbank

In der Standardkonfiguration gibt es eine zentrale Datenbank für alle Servermodule. Innerhalb dieser Datenbank haben diese Module alle ihre eigenen Tabellen und kommen

daher nicht miteinander in Konflikt. Diese Architektur ermöglicht es jedoch, die Daten der Servermodule auf eigene Datenbanken zu verteilen, wobei in den Modulen nur die Konfiguration für die Datenbankverbindung geändert werden müssen. Die Verbindung wird über TCP/IP aufgebaut, wodurch es für das Modul weitgehend[1] unbedeutend ist, wo die Datenbank liegt. Die alternative Architektur mit verteilten Datenbanken wird in Abbildung 5.2 dargestellt. Im weiteren Verlauf der Arbeit wird aus Gründen der Verständlichkeit eine zentrale Datenbank verwendet.

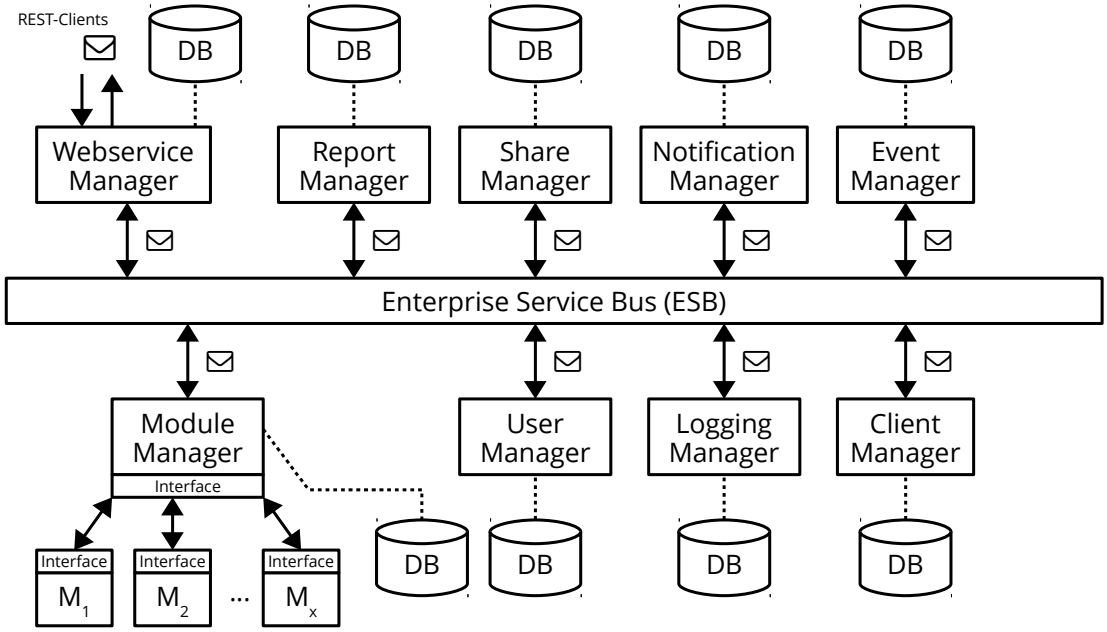

Abbildung 5.2: Architektur mit dezentraler Datenbank: Die neun Hauptkomponenten erhalten jeweils ihre eigene Datenbank.

5.3 Abläufe

Im Folgenden werden einige Systemabläufe anhand der Architektur aus Abbildung 5.1 erklärt. Da sich diese in vergleichbarer Form für die anderen Module wiederholen, werden hier nur exemplarisch die wichtigsten Abläufe gezeigt.

[1] Liegt die Datenbank auf dem selben Server wie die Module, ist die Latenz natürlich deutlich geringer.

5.3.1 Anfrage eines Formulars

Wird vom Client ein Formular angefragt, so durchläuft die Anfrage den Webservice Manager, den ESB, den Module Manager und das entsprechende systeminterne Modul, welches das Formular bereitstellt. In Abbildung 5.3 wird dieser Durchlauf dargestellt.

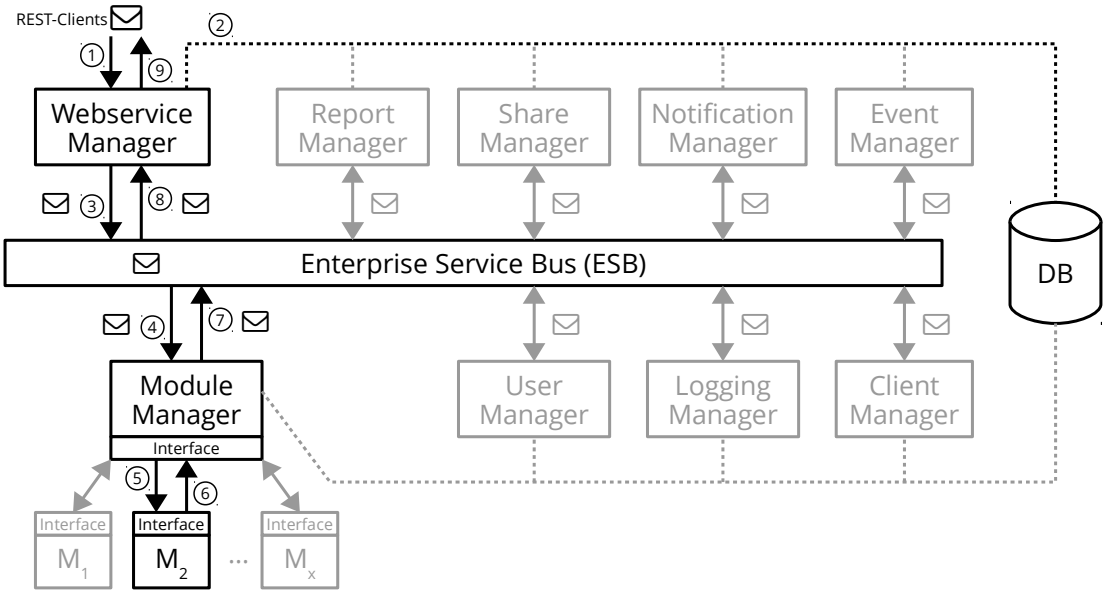

Abbildung 5.3: Anforderung eines Formulars: Die Anfrage wird von der REST-API über den ESB an den Module Manager weitergeleitet.

1. Der Client stellt über die REST-API eine Anfrage an den Webservice Manager.
2. Um den Benutzer zu authentifizieren, werden die Logindaten mit der Datenbank abgeglichen.
3. Nach erfolgreicher Authentifizierung erstellt der Webservice Manager eine an den Module Manager adressierte Systemnachricht und legt diese auf den ESB.
4. Der Module Manager nimmt die Nachricht vom ESB und überprüft den Inhalt.
5. Kennt der Module Manager das angeforderte Modul, leitet er die Anfrage an dieses weiter. Das Modul erstellt daraufhin das angeforderte Formular.
6. Das Modul gibt das Formular an den Module Manager zurück.

7. Der Module Manager verpackt das Formular in eine Systemnachricht und adressiert diese an den Webservice Manager.
8. Der Webservice Manager wartet bereits auf die Antwort des Module Managers, empfängt diese und ordnet sie der ursprünglichen API-Anfrage zu.
9. Der Webservice Manager schickt die HTTP-Response mit dem Formular an den Client.

5.3.2 Speichern eines ausgefüllten Formulars

Nachdem das Formular auf dem Client ausgefüllt wurde, müssen die Daten zurück an den Server geschickt werden. Dieser stellt durch die Eingabevalidierung sicher, dass die Felder richtig ausgefüllt wurden. Ist dies der Fall, können die Datensätze in der Datenbank persistiert werden.

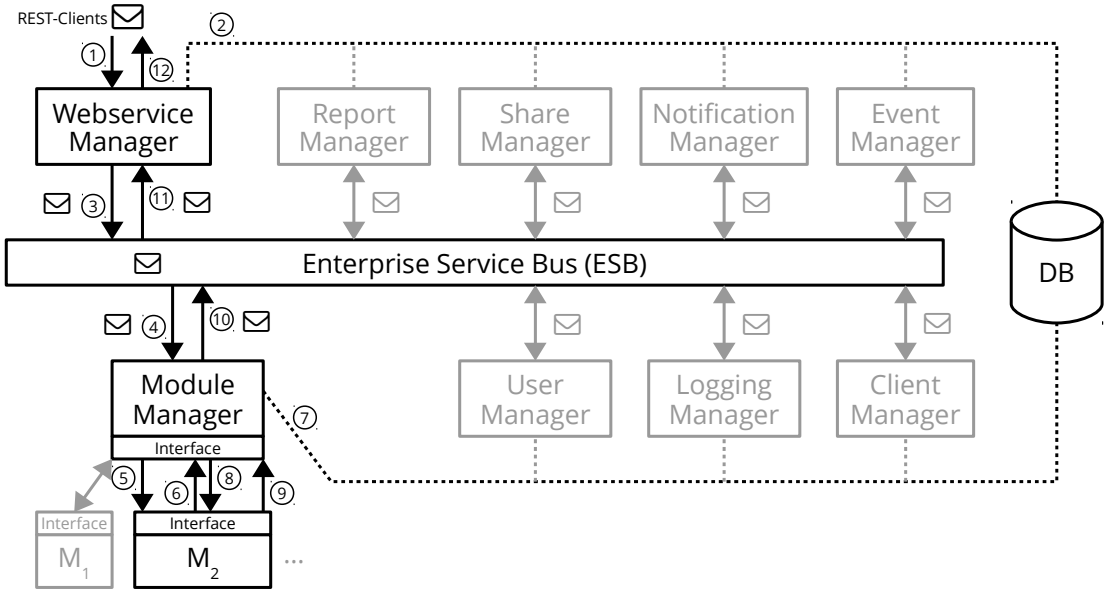

Abbildung 5.4: Antwort eines Formulars: Für Datenbankzugriffe der systeminternen Module muss der Module Manager verwendet werden.

1. Der Client stellt über die REST-API eine Anfrage an den Webservice Manager.
2. Um den Benutzer zu authentifizieren, werden die Logindaten mit der Datenbank abgeglichen.
3. Nach erfolgreicher Authentifizierung erstellt der Webservice Manager eine an den Module Manager adressierte Systemnachricht mit dem ausgefüllten Formular und legt diese auf den ESB.
4. Der Module Manager nimmt die Nachricht vom ESB, überprüft deren Format und führt danach eine Eingabevalidierung durch (siehe Abschnitt 4.11). Schlägt diese fehl, muss die Annahme verweigert und eine Fehlermeldung an den Client zurückgeliefert werden.
5. War die Eingabevalidierung erfolgreich und kennt der Module Manager das angeforderte Modul, leitet er den Datensatz an dieses weiter. Das Modul modifiziert ggf. die empfangenen Daten.
6. Das Modul hat selbst keinen direkten Zugriff auf die Datenbank und muss daher zum Speichern den Module Manager aufrufen.
7. Der Module Manager prüft die Anfrage des Moduls und persistiert den Datensatz bei Korrektheit in der Datenbank.
8. Nach erfolgreichem Speichern gibt der Module Manager die Kontrolle an das Modul zurück.
9. Das Modul erstellt eine Liste mit den Anfragen, die es als nächstes vom Client erwartet und übergibt sie an den Module Manager. Ist das Modul korrekt beendet, ist diese Liste leer.
10. Der Module Manager verpackt die Liste in eine Systemnachricht und adressiert diese an den Webservice Manager.
11. Der Webservice Manager wartet bereits auf die Antwort des Module Managers, empfängt diese und ordnet sie der ursprünglichen API-Anfrage zu.
12. Der Webservice Manager schickt die HTTP-Response mit der Liste an den Client.

5.3.3 Triggern eines Zeitereignisses

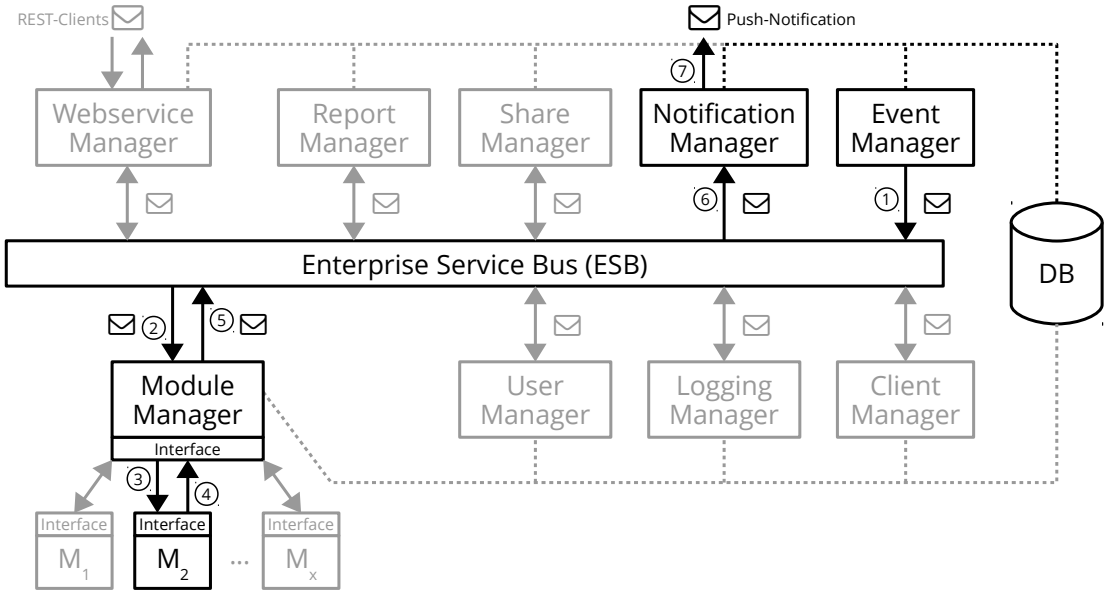

Abbildung 5.5: Ablauf eines Zeit-Ereignisses: Der Event Manager triggert ein Zeit-Ereignis, das vom Modul M_2 bestätigt und über den Notification Manager propagiert wird.

1. Der Event Manager überprüft jede Minute, ob sich in seiner Liste ein Zeitereignis befindet, das ausgelöst werden muss. Ist dies der Fall, wird eine Systemnachricht auf den ESB gelegt, die an den Module Manager adressiert ist. Die Nachricht enthält unter anderem eine Liste der Clients, die sich in der entsprechenden Zeitzone befinden, um aufgerufen zu werden. Der Client Manager ist für diesen Vorgang nicht nötig, da mit dessen Hilfe im Vorhinein die Trigger-Liste des Event Managers mitsamt der Client-Informationen erstellt wurde.

2. Der Module Manager nimmt die Anfrage des Event Managers entgegen und lädt das entsprechende Modul.

3. Die Anfrage wird an des Modul übergeben, welches entscheiden kann, ob es wirklich aufgerufen werden möchte.

4. Soll das Modul aufgerufen werden, gibt es die Anfrage an den Module Manager zurück.
5. Der Module Manager erstellt eine Nachricht an den Notification Manager. Wichtig ist dabei, dass die Empfängerliste in der Nachricht enthalten ist.
6. Der Notification Manager nimmt die Nachricht vom ESB und verarbeitet die Anfrage.
7. Die Clients werden per Push-Nachricht informiert, dass sie ein Formular vom getriggerten Modul anfordern sollen.

Bisher wurden die einzelnen Komponenten des Systems beschrieben und deren Zusammenhänge erläutert. Zum tieferen Verständnis und der Erklärung einiger Besonderheiten wird im folgenden Kapitel näher auf deren technische Umsetzung eingegangen.

6

Technische Umsetzung

In diesem Kapitel werden ausgewählte Aspekte der Implementierung im Detail vorgestellt. Einerseits wird die technische Umsetzung von Komponenten besprochen, die bereits in Kapitel 5 diskutiert wurden. Andererseits werden auch Komponenten besprochen, die zwar für die technische Realisierung wichtig sind, jedoch nicht zur Grundarchitektur des Systems gehören und daher in Kapitel 5 außer Acht gelassen wurden. Diese sind beispielsweise die Docker-Container (siehe Abschnitt 6.2) oder das Formularschema (siehe Abschnitt 6.3.1).

6.1 Verwendete Technologien und Frameworks

Das System ist sehr umfangreich und komplex, zudem soll es stabil laufen und skalierbar sein. Deshalb wurde bei der Implementierung auf einige bewährte Technologien und Frameworks gesetzt.

PHP 5.6 Als Skriptsprache wurde PHP 5.6 gewählt. Sie ist weit verbreitet, läuft auf fast allen Webservern und liefert eine sehr gute Unterstützung für die Verarbeitung von Text. Außerdem stehen wohldefinierte Schnittstellen für die anderen verwendeten Frameworks zur Verfügung.

Laravel 5.0 Ein Framework für PHP zur Erstellung von RESTful Webservices, welches einfaches Routing von URLs auf PHP-Methoden und viele weitere Funktionen unterstützt, wie Logging, Authentifizierung und Migration von Datenbankschemen.

MySQL Als freie und vielfach bewährte Datenbanklösung zur persistenten Speicherung aller anfallenden Daten im System. Dazu zählen sowohl die Konfigurationsdaten der Module, als auch deren Ergebnisdatensätze.

Docker Zur Modularisierung der Serverkomponenten wurde Docker gewählt. Diese Modularisierungstechnik ist sehr leichtgewichtig und läuft nativ auf Linux, ohne eigene Hardware virtualisieren zu müssen (siehe Abschnitt 2.4).

RabbitMQ ist eine Implementierung des AMQP-0.9.1-Protokolls, wurde in Erlang geschrieben und kommt auf dem Server als Enterprise Service Bus zum Einsatz. [ARA+08]

Pushbots ist ein Dienst zum einfachen Versenden von Push-Nachrichten, der sowohl eine PHP-API, als auch APIs für die gängigen mobilen Betriebssysteme bereitstellt.

Composer für die Modulverwaltung. Composer ist ein in PHP geschriebenes Framework für die Verwaltung von Abhängigkeiten. Obwohl die Entwickler von Composer behaupten, dass es keine Paketverwaltung wäre, wird es dennoch genau dafür im System benutzt.

> Composer is *not* a package manager in the same sense as Yum or Apt are. Yes, it deals with “packages” or libraries, but it manages them on

a per-project basis, installing them in a directory (e.g. vendor) inside your project. By default it will never install anything globally. Thus, it is a dependency manager [AB15].

GitLab ist eine web-basierte Verwaltung für Git-Repositories und wird als Quelle für die Pakete im Modul-Repository verwendet.

REST API als leichtgewichtige Kommunikationsschnittstelle zwischen dem Server und den Clients.

6.2 Docker

Die Komponenten der in Kapitel 5 beschriebenen Architektur werden auf dem Server in mehrere Docker-Container verteilt. Dadurch wird es möglich, die Komponenten ohne Neuinstallation mitsamt ihrer Umgebung und ihren Daten auf einen anderen Server zu verschieben. Es werden zwei alternative Konfigurationen erläutert. Die aggregierte Struktur in Abschnitt 6.2.1 wird in dieser Arbeit verwendet. Die verteilte Architektur in Abschnitt 6.2.2 eignet sich besser, wenn die Systemkomponenten, beispielsweise zur Lastverteilung, auf mehrere Server verteilt werden sollen.

6.2.1 Aggregierte Struktur

In Abbildung 6.1 wird die Struktur gezeigt, wie die Docker-Container in dieser Arbeit zum Einsatz kommen. Dabei werden so viele Komponenten in einen Container installiert, wie es für eine spätere Verteilung bzw. den Austausch von Modulen für Updates praktikabel ist. Die Farben der Komponenten dienen lediglich der besseren Übersichtlichkeit und haben keine weitere Bedeutung.

Im Folgenden werden die Docker-Container mit den enthaltenen Systemkomponenten aufgelistet und beschrieben:

web Der `web`-Container enthält einen Nginx-Webserver, der die API-Anfragen entgegennimmt und an den Webservice Manager (siehe Abschnitt 5.2.2) weitergibt. Dieser

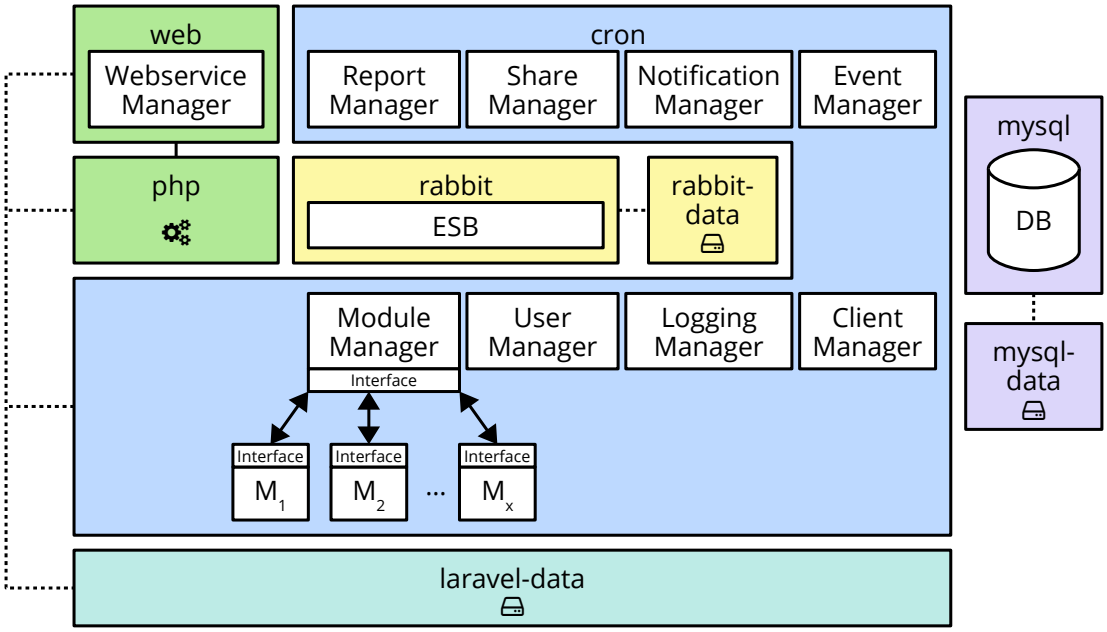

Abbildung 6.1: Aggregierte Docker-Struktur: Bis auf den Webservice Manager befinden sich alle Servermodule im `cron`-Container.

ist in PHP geschrieben und muss zur Ausführung an den `php`-Container weitergegeben werden. Die Skript-Dateien, sowie das Laravel-Framework, auf dem der Webservice Manager aufbaut, liegen im `laravel-data`-Container. Dessen Volumes werden beim Start des Containers gemountet (siehe Abschnitt 2.4). Außerdem liegen dort die Konfigurationsdateien für die virtuellen Hosts des Webservers.

php Der `php`-Container wird vom `web`-Container über TCP aufgerufen, wenn ein PHP-Skript ausgeführt werden muss und könnte daher auch auf einem entfernten Server liegen. Dies würde die Ausführung allerdings deutlich verlangsamen. Außerdem übergibt der Webserver nur den Pfad der auszuführenden Datei und nicht die Skript-Datei selbst. Daher müssen beim Start des `php`-Containers ebenfalls die Volumes des `laravel-data`-Containers gemountet werden. Da die Volumes nicht ohne Weiteres von einem entfernten Server gemountet werden können, müsste bei einer Verteilung auf zwei Server der `laravel-data`-Container dupliziert und synchron gehalten werden.

rabbit Für den ESB kommt RabbitMQ (siehe Abschnitt 6.4) zum Einsatz. Dieser Docker-Container wird bereits von den Entwicklern fertig bereitgestellt und kann beim ersten Start über Umgebungsvariablen konfiguriert werden. Es stehen zwei Modi des Containers zur Auswahl. Beim einen läuft nur RabbitMQ selbst, der andere stellt zusätzlich eine Webanwendung zum Monitoring und zur Administration bereit.

rabbit-data Der `rabbit-data`-Container beherbergt das Volume für den `rabbit`-Container. Weil keine anderen Container auf das Volume zugreifen, könnte der Container eingespart und die Daten direkt im `rabbit`-Container gespeichert werden. Allerdings können dadurch Updates einfacher eingespielt werden, da der `rabbit`-Container mit der alten Version einfach gelöscht und durch einen neuen ersetzt werden kann, ohne dabei die Daten zu verlieren.

cron Alle Manager-Module (siehe Abschnitt 5.2, mit Ausnahme des Webservice Managers) sind im `cron`-Container enthalten. Auch die systeminternen Module werden in den `cron`-Container installiert. Durch das Mounten der Volumes des `laravel-data`-Containers werden diese persistent gespeichert, auch wenn der `cron`-Container ausgetauscht wird.

Beim Start des `cron`-Containers wird ein Cronjob gestartet, der jede Minute den Task-Scheduler von Laravel aufruft. In diesem wiederum werden die Listener der einzelnen Servermodule eingetragen, die auf dem ESB hören und auf Nachrichten warten. Zum vollständigen Funktionieren müssen im `cron`-Container einige Programme installiert werden:

- PHP mit einigen Plugins, da die Servermodule in PHP geschrieben sind.
- cron für den Cronjob.
- Composer für die Modulverwaltung.
- Git für die Modulverwaltung mit Composer und GitLab.

Ein annotiertes *Dockerfile* befindet sich im Anhang (siehe Abschnitt A.1.1).

laravel-data Alle PHP-Skripte, die Konfigurationsdateien für die virtuellen Hosts, sowie die Log-Files des Webservers werden nach dem Data-Only-Paradigma (siehe

Abschnitt 2.4) im `laravel-data`-Container persistent gespeichert. Außerdem enthält dieser Container die zur Laufzeit angelegten Daten, die nicht in der Datenbank gespeichert werden, wie beispielsweise vom Administrator installierte systeminterne Module.

Obwohl die Komponenten voneinander logisch unabhängig sind, teilen sie sich die Volumes in diesem Container. Die Module, die zusammen im `cron`-Container untergebracht sind, teilen sich zwangsläufig auch die Volumes. Warum `web` und `php` auf die selben Volumes zugreifen, wurde bereits beim `php`-Container erläutert. Dass sich `cron` und `web` ihre Volumes teilen, hat einen praktischen Grund, solange sie zusammen auf einem Server liegen: Beim Update des Laravel-Frameworks sind gleich die Module beider Docker-Container betroffen.

mysql In diesem Container läuft die MySQL-Datenbank, die via TCP/IP erreichbar ist. Wie der `rabbit`-Container wird auch der `mysql`-Container bereits von den Entwicklern als fertiger Docker-Container bereitgestellt und kann beim ersten Start über Umgebungsvariablen konfiguriert werden.

mysql-data ist der Data-Only-Container (siehe Abschnitt 2.4) für `mysql`. Der Grund für diesen separaten Container ist analog zum `rabbit-data`-Container.

6.2.2 Verteilte Struktur

Die in Abbildung 6.2 gezeigte verteilte Struktur unterscheidet sich insofern von der aggregierten Struktur (siehe Abbildung 6.1), dass sämtliche Servermodule in eigene Container ausgelagert wurden. Der `modules`-Container erhält als einziger einen eigenen Datencontainer, da er der einzige ist, der Daten erzeugt, die in einem Volume persistiert werden müssen. Im Speziellen handelt es sich dabei um die installierten systeminternen Module. Die anderen Module legen ihre zu persistierenden Daten in der Datenbank des `mysql`-Containers ab, die Programmdaten müssen beim Erstellen des Containers integriert werden. Der `laravel`-Container aus Abbildung 6.1 wird in die Container `web-data` und `modules-data` aufgeteilt. Beide müssen eine eigene Laravel-Instanz enthalten, die bei Bedarf separat aktualisiert werden muss.

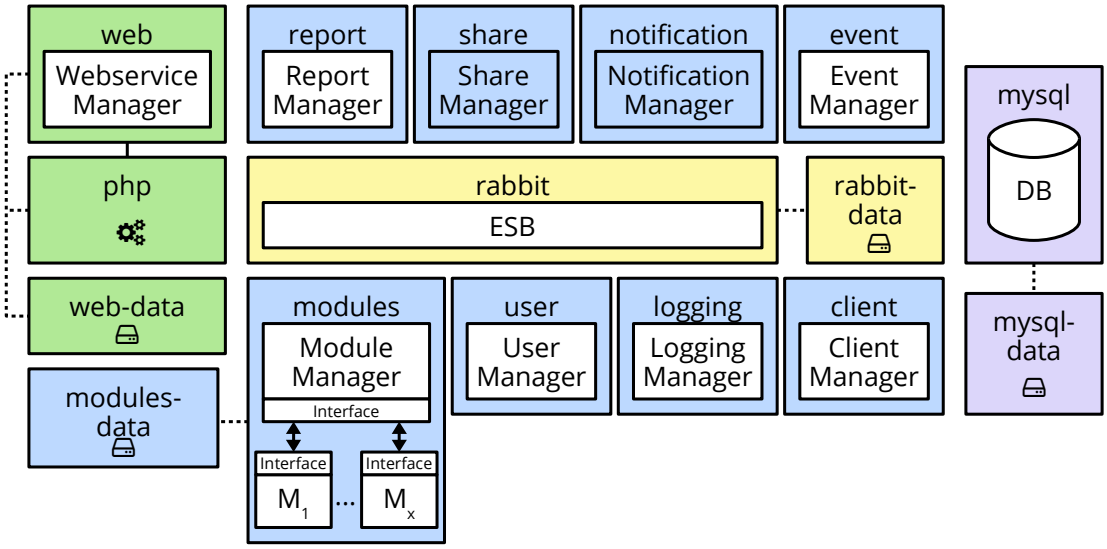

Abbildung 6.2: Verteilte Docker-Struktur: Alle Servermodule befinden sich in einem separaten Docker-Container.

6.2.3 Einrichten der Docker-Container

Um Docker-Container auf dem Server verwenden zu können, muss zunächst Docker installiert und der Service gestartet werden. Anschließend können die Container auf zwei Arten erstellt werden, die hier beide zum Einsatz kommen. Einerseits können fertige Container vom Docker-Repository verwendet, andererseits können diese selbst aus einem *Dockerfile* erstellt werden.

Die Images für die Container `mysql` und `rabbit` werden von den Entwicklern fertig über das Docker-Repository zur Verfügung gestellt, wie bereits in Abschnitt 6.2.1 erwähnt. Diese Images können mit `docker pull` einfach heruntergeladen werden, um später daraus einen Docker-Container zu erstellen. Die anderen Images müssen zunächst mittels `docker build` aus einem Dockerfile gebaut werden. Das bereits in Abschnitt 6.2.1 erwähnte Dockerfile findet sich als Beispiel im Anhang (siehe Abschnitt A.1.1).

Sind alle Images vorbereitet, können daraus die lauffähigen Docker-Container erstellt werden. In Listing 6.1 wird das grundlegende Vorgehen gezeigt, eine vollständige Anleitung für die aggregierte Struktur wird im Anhang beschrieben (siehe Abschnitt A.2).

Die Reihenfolge der Aufrufe ist wichtig, da nur Container verbunden werden können, die bereits bestehen.

```
1 docker run --name laravel-data arnimschindler/laravel-data &&\
2 docker run --privileged=true --name php \
3     --volumes-from laravel-data -d arnimschindler/php &&\
4 docker run --privileged=true --name web \
5     --volumes-from laravel-data -p 80:80 --link php:fpm
6     -d arnimschindler/web-nginx
```

Listing 6.1: Erstellen von Docker-Containern aus Images: Bei der ersten Ausführung werden die Container mit dem Befehl `docker run` erzeugt.

Die Container können jederzeit mit dem Befehl `docker start|stop|restart` gestartet, angehalten oder neugestartet werden.

6.3 Formulare

Formulare sind ein wichtiger Bestandteil des Systems. Sie werden von den Systeminternen Modulen bereitgestellt und die Antwort vom Module Manager validiert. Für eine einheitliche Verarbeitung auf allen Clients muss das Schema, wie die Formulare definiert und aufgebaut sind, genau spezifiziert werden.

6.3.1 Schema

Im Folgenden werden einige ausgewählte Teile des Formularschemas beschrieben, die eine Übersicht über den Aufbau und die Funktionalität des Schemas bieten. Das vollständige Formularschema wird in Anhang B aufgeführt. Die Formulare werden durch JSON beschrieben. Ein solches Formular besteht aus den vier Hauptabschnitten `version`, `header`, `fields` und `compositeFields`.

Versionen

Durch die Versionierung des Beschreibungsschemas wird es möglich, die Formulardefinition zu verändern, ohne auf Konflikte durch verschiedene Beschreibungen zu stoßen. Die Versionsnummer des Formularschemas ist unabhängig von der API-Version. So können Änderungen am Schema vorgenommen werden, ohne andere Bereiche des Systems auf eine neue API-Version aktualisieren zu müssen. Die Versionsnummern sollen per Konvention der semantischen Versionierung[1] folgen.

Header

Das `header`-Feld enthält Meta-Informationen zum Forumular, beispielsweise eine Transaktions-ID. Diese wird vom Server automatisch generiert, falls der Client keine bereits bekannte Transaktions-ID mitschickt. Sie ist bei mehrseitigen Formularen wichtig, um die Ergebnisdatensätze zu einem Formulardurchlauf zusammenzufassen. Sendet der Client einen `GET`-Request an den Server, so ist die Transaktions-ID als Querystring zu übergeben (`?transactionId={transactionId}`). Bei `POST`-Requests ist sie Teil des JSON-Schemas (siehe Listing 6.2).

```
1 { "form":{
2     "header":{
3       "transactionId": "h7erpH98zer" }}}
```

Listing 6.2: Ist die Transaktions-ID bereits bekannt, wird sie bei der Formularanfrage im `header`-Feld angeben.

Die Serverantwort (siehe Listing 6.3) enthält zusätzlich die Lebensdauer der Transaktion in Sekunden, sowie das Enddatum als `datetime` nach RFC 3339 [KN02]. So kann der Client entsprechend reagieren, wenn die Transaktion abläuft, bevor die Anfrage vom Benutzer bearbeitet wird.

[1] `http://semver.org`

```
{ "form":{
    "header":{
      "transactionId": "h7erpH98zer",
      "transactionTTL": 1800,
      "transactionExpires": "2015-07-28T14:32:48Z" }}}}
```

Listing 6.3: Bei der Antwort des Servers wird außer der Transaktions-ID auch deren Lebensdauer überliefert.

Fields

Das `fields`-Attribut enthält ein Array mit den Formularfeldern, die auf dem Client angezeigt werden müssen. Die `field`-Objekte müssen bzw. können einige notwendige sowie optionale Eigenschaften annehmen.

`name` (required, unique) Der eindeutige Name des Formularfeldes, welcher für die Datenübertragung und -speicherung verwendet wird. Er dient ausschließlich der Datenverwaltung und Identifikation des Formularfeldes und wird nicht angezeigt.

`label` (optional) Die für den Benutzer sichtbare Beschriftung des Formularfeldes. Sie wird als Überschrift des Formularblocks gezeigt, wenn das Feld vom Datentyp `compositeField` ist. Der Inhalt wird bereits vom Server an die Sprache des Clients angepasst, die über den HTTP-Header `Accept-Languages` angefordert wird.

`type` (required) Der Datentyp des Formularfeldes. Der Umgang mit Datentypen wird in Abschnitt 6.3.2 genauer beschrieben.

`compositeField` (conditional) Dieses Attribut kommt zum Einsatz, wenn das Feld kein einfaches Formularfeld, sondern ein Formularblock ist. Das Attribut referenziert ein eindeutiges `compositeField`, welches im Schema unter `compositeFields` definiert wird. Das Attribut wird nur beachtet, wenn unter `type` der Datentyp `compositeField` definiert ist.

`pattern` (optional) Zusätzlich zu den Eigenschaften der einzelnen Datentypen kann ein regulärer Ausdruck (siehe AF 71) nach JavaScript Pattern Syntax ECMA262[2] angegeben werden, wie er in HTML5 verwendet wird. Dieser muss bei der Eingabevalidierung (siehe Abschnitt 6.3.4) zusätzlich zu den datentypspezifischen Einschränkungen beachtet werden.

`required` (optional) Das Attribut gibt an, ob das Feld notwendig (`true`) oder optional (`false`) ist. Fehlt das Attribut, wird als Standardwert `true` angenommen und das Formularfeld muss somit ausgefüllt werden.

Listing 6.4 zeigt den Aufbau eines einfachen Formularfeldes, das optional ausgefüllt werden kann (`"required": false`). Das Feld implementiert das Beispiel mit der hexadezimalen Zahl aus AF 71.

```
1 { "form": {
2   "fields": [
3   { "name": "myHexNumber",
4     "type": "text",
5     "label": "Eine beliebige hexadezimale Zahl",
6     "required": false,
7     "pattern": "^[0-9a-f]*$" }]
8 }}
```

Listing 6.4: Ein Formularfeld vom Typ `text` mit Beschriftung und regulärem Ausdruck.

CompositeFields

Der letzte der vier Hauptabschnitte des Schemas, das `compositeFields`-Attribut, besteht aus einem Array von `compositeField`-Objekten. Diese enthalten jeweils ein eindeutiges `name`-Attribut zur Identifikation, sowie ein Array aus `field`-Objekten (siehe Abschnitt 6.3.1). `CompositeFields` dienen der Wiederverwendung von Formularabschnitten, beispielsweise einer Eingabemaske für Adressen. Als Überschrift für den

[2] `http://people.mozilla.org/~jorendorff/es5.1-final.html#sec-15.10.1`

Formularblock dient das `label`-Attribut des `field`-Objekts. Listing 6.5 zeigt, wie das gleiche `compositeField` in einem Formular für die Liefer- und Rechnungsadresse mit verschiedenen Überschriften verwendet werden kann.

```
{ "form": {
    "fields": [
    { "name": "shippingAddress",
      "type": "compositeField",
      "compositeField": "address",
      "label": "Lieferadresse" },
    {
       "name": "billingAddress",
       "type": "compositeField",
       "compositeField": "address",
       "label": "Rechnungsadresse" }
    ],
    "compositeFields": [
    { "name": "address",
      "fields": [
      { "name": "name",
        "label": "Vor- und Zuname",
        "type": "text" },
      {
        "name": "street",
        "label": "Strasse und Hausnummer",
        "type": "text" }
      ]}
    ]
}}
```

Listing 6.5: Mit `CompositeFields` können Feldgruppen in Formularen erstellt werden.

6.3.2 Datentypen

In Abschnitt 6.3.1 wurden die Datentypen (siehe AF 64) bereits erwähnt. Sie werden gebraucht, um Formularfeldern spezifische Eigenschaften für die Benutzereingabe und Eingabevalidierung (siehe Abschnitt 6.3.4) zu ermöglichen. Um in allen Systemkomponenten auf einen gemeinsamem Bestand an Datentypen zugreifen zu können, definiert das Formularschema genau, welche Datentypen zur Verfügung stehen, sowie welchen Einschränkungen diese unterliegen. Die Definitionen sind weitgehend von der HTML5-Spezifikation abgeleitet [BFHu14]. Im Folgenden werden einige ausgewählte Datentypen beschrieben. Die vollständige Definition der Datentypen befindet sich im Anhang (siehe Abschnitt B.1.4).

`"type":"text"`

Der Datentyp `text` wird für ein einzeiliges Formularfeld verwendet. Als Eingabe ist beliebiger Text ohne Zeilenumbruch erlaubt. Die Steuerzeichen `\r` und `\n` dürfen daher nicht enthalten sein. Die Eingabevalidierung muss ein Patternmatching auf den regulären Ausdruck `^[^\r\n]+$` durchführen.

Desweiteren besitzen Formularfelder vom Typ `text` die optionalen Attribute `value` und `maxlength`. Mit ersterem kann ein Wert vorgegeben werden, mit letzterem wird die maximale Textlänge eingeschränkt.

`"type":"number"`

Für die Eingabe von Fließkommazahlen wird ein Feld vom Typ `number` verwendet. Der Wertebereich liegt zwischen `-2^1024` und `2^1024`. Der Wert wird als String in wissenschaftlicher Schreibweise von Zahlen repräsentiert und muss daher den regulären Ausdruck `^[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?$` erfüllen. Weiterhin können `number`-Felder die folgenden optionalen Attribute erhalten:

`value` Ein vorgegebener, valider Wert.

min Der kleinste erlaubte Wert, den der Benutzer eingeben können soll.

max Der größte erlaubte Wert, den der Benutzer eingeben können soll.

step Die Schrittweite der Skala als Fließkommazahl oder der String `any`. Fehlt das `step`-Attribut, wird ein Standardwert von `1` angenommen. Alle Werte, die von `base+step*x` abweichen, sind ungültig und müssen entsprechend behandelt werden. Dabei ist `base` der erste gültige Fließkommawert aus dem Tupel `(value, min, 0)` und `x` eine Ganzzahl ($x \in \mathbb{Z}$) [BFHu14, Abschnitt 4.10.5.3.8 The step attribute].

Beispiele:

1. Seien die Attribute `value`, `min` und `step` nicht gegeben. Daraus folgt nach obiger Definition, dass `base=0` und `step=1`. Somit sind alle ganzzahligen Werte erlaubt, da `value=0+1*x`, $x \in \mathbb{Z}$.
2. Seien `value=0.4` und `min=-5.5`, das Attribut `step` nicht gegeben. Daraus folgt, dass `base=0.4` und `step=1`. Damit können alle Zahlenwerte eingegeben werden, die `value=0.4+1*x`, $x \in \mathbb{Z}$, `value` $\geq$ `-5.5` erfüllen, also `-4.6, -3.6, ..., 0.4, 1.4,` usw.
3. Seien `min=0`, `max=1` und `step=0.00392156862` ($=\frac{1}{255}$). Daraus folgt, dass `base=0` und `step` wie gegeben. Damit können alle Werte im Bereich von `0` bis `1` in `255` Zwischenschritten angegeben werden: `value=0+0.00392156862*x`, `0` $\leq$ `value` $\leq$ `1`. Diese Werte werden beispielsweise bei der Angabe von Farbwerten verwendet, wenn die Skala auf den Wertebereich $[0, 1]$ normiert ist.

"type":"datetime"

Eingabefelder vom Typ `datetime` sind eine Eingabemöglichkeit für einen Zeitpunkt bestehend aus Datum, Zeit und Zeitzone. Die Eingabe soll über ein Kalender- und Uhr-Widget oder eine ähnliche grafische Bedienoberfläche erfolgen. Der interne Wert, der an den Server übergeben wird, erfolgt in der Form `{datum}T{zeit}{zeitzone}`, wobei `zeitzone` aus der Zeitverschiebung zur UTC-Time besteht. Das Format der Zeitzone ist `+hh:mm`, `-hh:mm` oder `Z` als Alias für `+00:00` [KN02].

Der Wert muss dem regulären Ausdruck `^[0-9]{4,}(-[0-9]{2}){2}T[0-9]{2}:[0-9]{2}(:[0-9]{2}(\.[0-9]{1,3})?)?(Z|[+-][0-9]{2}:[0-9]{2})$` entsprechen und ein kalendarisch existentes Datum mit gültiger Uhrzeit sein. Ein valider Wert ist beispielsweise `2015-07-28T14:32:48+02:00` für den 28.07.2015 um 14:32:48 Uhr in Berlin zur Sommerzeit.

Der Datentyp `datetime` orientiert sich an der HTML5-Spezifikation [BFHu14] und erweitert RFC 3339 [KN02] insofern, dass `T` und `Z` immer Großbuchstaben sein müssen und die Jahreszahl aus mindestens vier Ziffern bestehen, sowie größer als `0` sein muss.

Des weiteren unterstützt `datetime` die Attribute `value`, `min` und `max` zur Vorgabe bzw. Einschränkung des Wertes.

`"type":"multiselect"`

Zur Auswahl einer oder mehrerer Optionen aus einer Liste von mehreren vorgegebenen Möglichkeiten dient der Datentyp `multiselect`. Im Gegensatz zu den bisher beschriebenen Datentypen, kann dieser Typ mehrere Werte gleichzeitig annehmen., die zudem alle vorgegeben werden müssen. Diese Werte werden über ein Array im `options`-Attribut übergeben. Eine Vorauswahl an Einträgen kann über das `value`-Attribut auf `field`-Ebene erzielt werden. In Listing 6.6 wird die Struktur eines `multiselect`-Feldes gezeigt, bei dem die Einträge mit den Werten `v1` und `v3` vorausgewählt wurden.

```
1 { "form": {
2   "fields": [
3   { "name": "multiselect_example",
4     "label": "Ein Beispiel fuer Multiselect",
5     "options": [
6     { "value": "v1", "label": "Wert 1" },
7     { "value": "v2", "label": "Wert 2" },
8     { "value": "v3", "label": "Wert 3" }],
9     "value": ["v1","v3"]} }}
```

Listing 6.6: Der Datentyp `multiselect` mit vorausgewählten Optionen.

6.3.3 Antwortschema

Während das Schema aus Abschnitt 6.3.1 die Formulare beschreibt, wie sie an die Endgeräte gesendet werden, definiert das Antwortschema, wie die Ergebnisse zurück an den Server übertragen werden. Die Schemata sind sich sehr ähnlich, da sie beide das entsprechende Feld samt Wert über die Attribute `name` und `value` beschreiben. Das Antwortschema kann jedoch auf die anderen Attribute verzichten, da die Werte bereits auf dem Server hinterlegt sind. Auch die `compositeFields` müssen nicht erneut übertragen werden, da diese schon bei der Anzeige des Formulars dereferenziert werden und somit nur noch einzelne Instanzen der `compositeFields` übrig bleiben. Das Schema selbst muss keine Zuordnung zur aktuellen Formularseite beinhalten, da dies über die URL der REST-API erfolgt. Zur Unterscheidung wird die Formularantwort in ein `formreply`-Objekt verpackt (vgl. `form`-Objekt für Formularschema).

Listing 6.7 zeigt ein Antwortschema mit dem Textfeld aus Listing 6.4, der Rechnungsadresse aus Listing 6.5 und dem `multiselect`-Feld aus Listing 6.6. Dabei ist sowohl zu erkennen, dass das `compositeField` aufgelöst wurde, als auch, dass das `value`-Attribut des `multiselect`-Feldes in seiner Syntax aus dem Formularschema übernommen wurde.

```
1  { "formreply": {
2    "fields": [
3    { "name": "myHexNumber", "value": "caffee1337affe"},
4    { "name": "multiselect_example", "value": ["v3"]},
5    { "name": "billingAddress",
6      "value": [
7      { "name": "name", "value": "Sherlock Holmes"},
8      { "name": "street", "value": "Baker Street 221b"}]
9    }]
10 }}
```

Listing 6.7: Das Antwortschema ist analog zum Formularschema aufgebaut.

6.3.4 Eingabevalidierung

In Abschnitt 4.11 gefordert und in Abschnitt 6.3.2 bereits angesprochen, sowie teilweise behandelt, ist die Eingabevalidierung ein sehr wichtiger Bestandteil des Systems, um fehlerhafte Eingaben zu vermeiden. Die Validierung findet an zwei Stellen statt: einerseits auf dem Endgerät, andererseits auf dem Server.

Überprüfung auf Client- und Serverseite

Die erste Prüfung der Daten soll schon auf dem Endgerät selbst stattfinden, während der Benutzer diese eingibt. Dadurch ist ein unmittelbares Feedback möglich, wodurch Eingaben abgewiesen und sofort korrigiert werden können, ohne die Daten zuerst an den Server zu schicken.

Die zweite Prüfung findet auf dem Server im Module Manager (siehe Abbildung 5.1) statt. So werden mögliche Implementierungsfehler des Clients abgefangen und die systeminternen Module können sich darauf verlassen, dass die eingehenden Daten verifiziert wurden.

Datentypen und Eingabemuster

Wie in Abschnitt 6.3.2 beschrieben, besitzen einige Datentypen detaillierte Eigenschaften, die bei der Eingabevalidierung überprüft werden müssen. So muss bei einem `number`-Feld anhand des datentypspezifischen Eingabemusters zunächst überprüft werden, ob es sich bei der Eingabe um die Repräsentation einer Zahl handelt. Anschließend muss der String in eine Zahl umgewandelt werden und der Wertebereich mittels der Attribute `min`, `max` und `step` validiert werden. Wurde außerdem mit dem `pattern`-Attribut ein regulärer Ausdruck definiert, dient dieses als *zusätzliche* Einschränkung. Es sollte daher bei der Zuweisung eines Patterns darauf geachtet werden, dass sich dieses nicht mit den spezifischen Eigenschaften des Datentyps widerspricht, da das Feld sonst nie valide sein kann.

Pflichtfelder

Während die Validierung des `required`-Attributs bei einfachen Datentypen trivial ist, müssen `compositeFields` gesondert betrachtet werden. Diese können sowohl als Ganzes optional bzw. notwendig sein, als auch deren einzelne Felder. Das Verhalten kann mit zwei Regeln abgedeckt werden:

1. Ist an einem `compositeField` das `required`-Attribut auf **`true`** gesetzt (oder durch Weglassen des Attributs implizit `true`), so müssen alle Felder innerhalb dieses `compositeFields` gemäß deren `required`-Attribute ausgefüllt werden (siehe Abbildung 6.3, Abschnitt 1).
   ⇒ Alles ausfüllen entsprechend den `required`-Attributen der jeweiligen Felder.
2. Ist an einem `compositeField` das `required`-Attribut auf **`false`** gesetzt, so müssen alle Felder innerhalb des `compositeFields` gemäß deren `required`-Attribute ausgefüllt werden, sobald mindestens eines der Felder innerhalb des `compositeFields` ausgefüllt wird (siehe Abbildung 6.3, Abschnitt 2 und 3). Dabei spielt es keine Rolle, ob das `required`-Attribut dieses Feldes `true` oder `false` ist.
   ⇒ Alles *oder nichts* ausfüllen entsprechend `required`-Attributen der jeweiligen Felder.

Enthält ein `compositeField` ausschließlich `required`-Attribute mit dem Wert `false`, macht es folglich keinen Unterschied, ob das `required`-Attribut des `compositeFields` selbst `true` oder `false` ist.

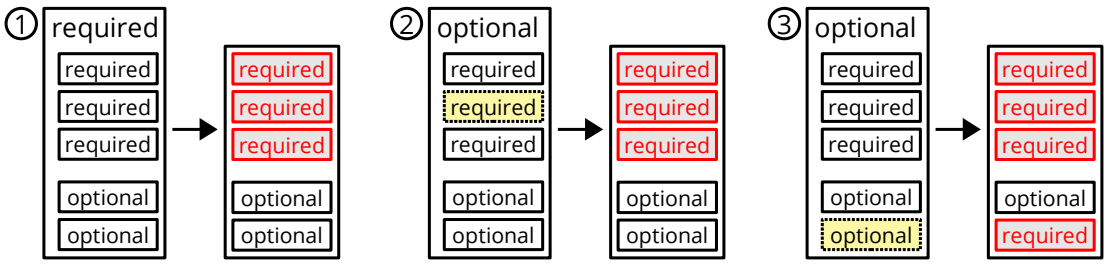

Abbildung 6.3: Verhalten des `required`-Attributs bei Formularblöcken.

Fehlermeldungen

Neben dem HTTP-Statuscode `200 OK` für erfolgreiche Übertragungen eines ausgefüllten Formulars, muss der Client auch mit dem Fehlercode `400 Bad Request` umgehen können. Dieser tritt immer dann auf, wenn das Schema verletzt wurde oder die Eingabevalidierung fehlschlägt. Genauere Informationen zum Fehler werden im Response-Body als JSON mitgeliefert, falls dies möglich ist. Dabei ist zu beachten, dass die HTTP-Anfrage möglicherweise formale Fehler enthält und somit schon vom Webserver mit `400 Bad Request` abgelehnt wird, bevor sie ans System selbst weitergereicht wird. Folglich kann die Fehlermeldung in diesem Fall nicht mit Details im JSON-Format angereichert werden. In Listing 6.8 wird eine solche Fehlermeldung gezeigt, bei der die Validierung der Felder `multiselect_example`, sowie der beiden Felder von `billingAddress`, aus Listing 6.7 fehlgeschlagen ist.

```
{ "formreply": {
  "errors": {
    "inputValidation": [
    { "name":"multiselect_example" },
    { "name":"billingAddress",
      "fields": [
      { "name":"name"},
      { "name":"street"}]} ]} }}
```

Listing 6.8: Eine Fehlermeldung der Eingabevalidierung für das Beispiel aus Listing 6.7.

6.4 ESB mit Listener-Pool

Der ESB (siehe Abbildung 5.1) wird auf Basis von RabbitMQ und dem AMQP[3] umgesetzt. Die Servermodule können sich mit dem RabbitMQ-Server verbinden und Nachrichten auf den ESB schicken. Zum Empfang der Nachrichten vom ESB muss sich ein *Listener* mit einer *Queue* verbinden und warten, bis Nachrichten eintreffen. Da PHP in der Regel eine

[3] Advanced Message Queuing Protocol

maximale Skriptlaufzeit hat, müssen die Listener periodisch über Cronjobs neugestartet werden. Wann das Skript beendet wird, kann nicht genau vorhergesagt werden, da die Wartezeit des Prozesses nicht als Skriptlaufzeit gezählt wird. Es würden daher häufig einige Sekunden zwischen Beenden des Skripts und dessen Neustart vergehen. In dieser Zeit könnten somit keine Nachrichten vom ESB abgearbeitet werden. Um dieses Problem zu umgehen, wird ein Listener-Pool mit mehreren aktiven Listenern verwendet. Durch einen versetzten Start der Listener wird selbst bei Gleichverteilung der Last und somit gleicher Skriptlaufzeit dafür gesorgt, dass die Skripte zu verschiedenen Zeitpunkten beendet werden und somit immer mindestens ein Listener arbeitet. Um die Verteilung der Nachrichten auf die verschiedenen Listener kümmert sich der ESB, sodass keine Nachricht mehrfach bearbeitet wird.

In Abbildung 6.4 sind die Queue für den Module Manager im ESB, sowie der Module Manager mit seinem Listener Pool abgebildet. Die Nachrichten werden vom ESB direkt auf die aktiven Listener (Listener 1-3) verteilt. Sobald ein inaktiver Listener vom Cronjob neugestartet wurde, wird er an der Verteilung der Nachrichten beteiligt (Listener 4).

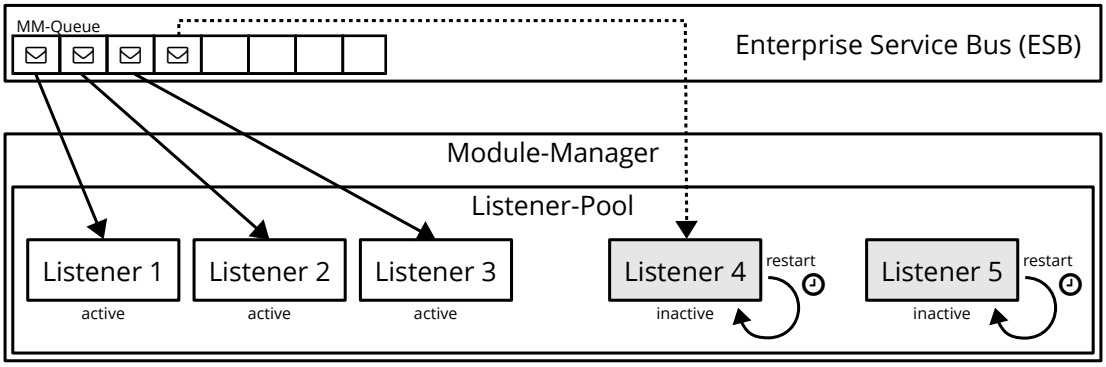

Abbildung 6.4: Listener-Pool: Inaktive Listener werden periodisch reaktiviert.

6.5 Module

Wie in der Architektur in Abbildung 5.1 gezeigt, werden systeminterne Module über den Module Manager aufgerufen. Der Client fordert von der REST-API die gewünschte

Methode des Moduls, woraufhin der Webservice Manager den Module Manager mit einem RPC-Call über den ESB aufruft [Sri95].

Weder der Webservice Manager, noch der Module Manager, müssen über die vom Modul bereitgestellten Methoden Bescheid wissen. Lediglich die Klasse `Main` muss implementiert sein, damit das Modul instanziiert werden kann. Um die Anfragen dennoch zur richtigen Methode leiten zu können, wird ein dynamisches Routing verwendet, wie es in Listing 6.9 gezeigt wird. Die Variablen `{vendor}` und `{modulename}` geben an, welches Modul vom Module Manager geladen werden soll (siehe Listing 6.10). Die Variablen `{command}` und `{args}` sind optionale Parameter. Sie enthalten die Methode, die aufgerufen werden soll, sowie deren Argumente. Der reguläre Ausdruck in Zeile 5 ist nötig, damit mehrere Argumente in der Form `arg1/arg2/arg3` erkannt werden. Ansonsten würde der Router nach dem ersten Argument abbrechen und könnte die Route möglicherweise nicht zuordnen. Die Middleware in Zeile 4 erzielt, dass die API nur von authentifizierten Benutzern verwendet werden kann.

```
1 Route::any(
2   'api/modules/{vendor}/{modulename}/{command?}/{args?}',
3   ['uses' => 'WebServiceController@loadModule',
4   'middleware' => 'auth.basic'])
5   ->where('args','.+');
```

Listing 6.9: Anfragen an die REST-API werden dynamisch geroutet.

Auch auf Seite des Module Managers werden die systeminternen Module dynamisch geladen. Zunächst wird anhand der vom Webservice Manager übergebenen Variablen der Klassennamen inklusive Namespace erstellt (Listing 6.10, Zeile 1f). Anschließend wird ein Objekt dieser Klasse instanziiert (Listing 6.10, Zeile 3). Der Classloader von Laravel findet anhand des Namespaces den Speicherort der PHP-Klasse, sofern sich das interne Modul an das PSR-4-Pattern[4] hält.

[4] `http://www.php-fig.org/psr/psr-4/`

```
1 $moduleNamesace =
2   '\\Crowdsensr\\Module\\'.$vendor.'\\'.$modulename.'\\Main';
3 $this->module = new $moduleNamesace($this);
```

Listing 6.10: Modulnamen werden dynamisch erstellt und geladen.

Kapitel 6 erlaubte tiefe Einblicke in die technische Umsetzung wichtiger Systembestandteile. So wurde sowohl die Modularisierung der Serverkomponenten mithilfe von Docker erklärt, als auch die Funktionsweise der systeminternen Module. Darüber hinaus wurde das JSON-Schema zum Erstellen von Formularen im Detail besprochen.

7

Zusammenfassung und Ausblick

In den vorhergehenden Kapiteln wurde das im Kontext dieser Arbeit entwickelte System ausführlich beschrieben. In diesem Kapitel werden Erkenntnisse aus dieser Arbeit zusammengefasst, Kritikpunkte geäußert und mögliche Erweiterungen diskutiert.

7.1 Zusammenfassung

Über den Zeitraum der Arbeit wurde die Architektur eines Systems definiert und prototypisch implementiert. Um dies zu erreichen, wurden zunächst die Anforderungen umfassend analysiert, bevor das daraus resultierende System erstellt werden konnte. Durch dessen modularen Aufbau wird es ermöglicht, generische Formulare zu erstellen und diese in das System zu integrieren. Durch die ereignisbasierte Steuerung können die Module kontextsensitiv aufgerufen werden. Zudem können diese über den Enterprise

Service Bus Daten austauschen, ohne voneinander wissen zu müssen. Dadurch können Module einfach hinzugefügt und entfernt werden, ohne dass auf Seiteneffekte geachtet werden muss.

Mithilfe der REST-API können beliebige Clients an das System angebunden werden. Die Erfassung der Daten beschränkt sich somit nicht nur auf mobile Endgeräte. Beispielsweise kann für die Erhebung der Daten auch Unterhaltungselektronik, wie ein Android TV, verwendet werden [Sch15]. Mobile Endgeräte sind jedoch von großem Nutzen, da sie in der Regel mit GPS und weiteren Ortungstechnologien ausgestattet sind. Clients, die ihren Standort nicht wissen, können folglich keine Ort-Ereignisse auslösen. Mit ihnen ist es dennoch möglich, Fragebögen manuell aufzurufen oder durch andere Ereignisse zu triggern.

Durch den Share- und Report-Manager wird es möglich, die erfassten Daten einzusehen und für andere Benutzter des Systems freizugeben. Der Export als JSON ermöglicht die einfache Weiterverarbeitung in externen Systemen. Die grafische Aufbereitung und Aggregierung in Reports soll der Benutzermotivation (siehe Abschnitt 3.2.1) dienen. Ob dies den gewünschten Effekt erzielt, muss in Feldversuchen und Studien gezeigt werden.

Der Notification Manager erlaubt die Kommunikation der Benutzer untereinander, sowie das Senden von Benachrichtigungen des Systems an den Benutzer. Dabei sind vor allem Push-Nachrichten von großer Bedeutung, mit denen der Server die Clients bei Bedarf aufrufen kann. Eine flexible Konfiguration des Notification Managers erlaubt es Benutzern, die Benachrichtigungen so einzustellen, dass er informiert, jedoch nicht vom System gestört wird.

Alles in allem lässt sich sagen, dass das System eine solide Grundlage für weitere Forschungen im Bereich modulares und ergeinisgesteuertes Crowd Sensing bietet. Der Einsatz von populären Technologien hat sich bewährt. Zum einen arbeiten diese stabil und bedeuten dadurch wenig Wartungsaufwand. Zum anderen sind diese Technologien sehr gut dokumentiert, was der weiteren Entwicklung zugute kommt.

7.2 Kritikpunkte

Im Verlauf der genaueren Spezifikation des Systems und späteren Umsetzung traten einige Aspekte auf, die bei der Weiterentwicklung in Betracht gezogen werden sollten.

7.2.1 Servermodule

Bei einer verteilten Installation auf mehreren Servern ist die modulare Architektur mit Docker-Containern eine praktikable Lösung. Die Container können einfach zwischen Servern verschoben werden, für Updates werden die Container durch die neue Version ersetzt. Allerdings reicht ein einziger Server in den meisten Fällen völlig aus, da das System weder rechen- noch lese-/schreibintensive Prozesse ausführt. In diesem Fall wäre es sowohl einfacher in der Installation bzw. Wartung, als auch ressourcensparender, auf die Modularisierung der Serverkomponenten zu verzichten.

7.2.2 Datenschutz und Sicherheit

In dieser Arbeit wurde der Fokus auf die Modularisierung, ereignisbasierte Steuerung, sowie das Formularschema gelegt. Dabei wurden die Themen Datenschutz und Sicherheit vernachlässigt, die vor einem produktiven Einsatz zwingend nachgeholt werden müssen. Zum einen wird die Authentifizierung der REST-API nur über das Basic-Auth-Verfahren durchgeführt, bei dem Benutzername und Passwort quasi im Klartext übertragen werden [FHBH+99]. Als vorläufige Lösung hierfür kann konsequent auf HTTPS gesetzt werden, bei dem die komplette Datenübertragung auf einer tieferen Ebene verschlüsselt wird [II94]. Zum anderen dient die E-Mailadresse als Benutzername und die Daten werden personenbezogen gespeichert. Hier sollte eine Möglichkeit geschaffen werden, um beispielsweise für klinische Studien anonymisierte Daten zu erheben.

Als positiver Aspekt der bisherigen Systemarchitektur ist die feingranulare Freigabe von Daten auf Gruppen- und Benutzerebene hervorzuheben. Diese sollte bei der Weiterentwicklung des Systems beibehalten werden.

7.2.3 Komplexe Formulare

Da nach jeder Formularseite die nächsten möglichen Schritte vom jeweiligen systeminternen Modul abgefragt werden, kann der Ablauf anhand der ausgefüllten Daten gesteuert werden. Innerhalb einer Seite kann jedoch nicht auf die Benutzereingabe eingegangen werden. Beim MACE-Test (siehe Abschnitt 3.2.2) ist es beispielsweise notwendig, dass einige Datenfelder nur eingeblendet werden, wenn ein anderes Formularfeld auf dieser Seite entsprechend ausgefüllt wurde. Um solche komplexen Formulare zu ermöglichen, sollte das Formularschema gegebenenfalls um logische Verknüpfungen zwischen den Eingabefeldern erweitert werden.

7.2.4 Onlinezwang

Ein weiterer Kritikpunkt ist die Tatsache, dass sämtliche Ablauflogik auf dem Server ausgeführt wird. Daher muss das Endgerät zwingend eine Internetverbindung haben, damit die Anwendung funktioniert. Um dieses Problem zu umgehen, könnten Teile der Logik auf das Endgerät ausgelagert und die eingegeben Daten zwischengespeichert werden. Erst wenn wieder eine Internetverbindung besteht, könnten die Daten an den Server übertragen werden.

7.3 Ausblick

Ob sich die Architektur und Umsetzung im Produktivbetrieb bewähren, muss noch in einem längeren Testzeitraum evaluiert werden. Interessant ist dabei auch die Frage, ob sich bei den Modulen bestimmte Muster herausbilden. In diesem Fall müsste das Formularschema entsprechend angepasst werden, um generischer zu werden. Für diesen Fall ist mit der Versionierung des Formularschemas bereits vorgesorgt.

Ein weiterer Mehrwert könnte durch ein Sensorframework erreicht werden, mit dessen Hilfe sowohl die internen Sensoren des mobilen Endgeräts, als auch externe Sensoren angesprochen werden können [SSP+13]. Dadurch könnten sich weitere Forschungsbereiche in der Medizin oder Messtechnik ergeben.

Literaturverzeichnis

[AB15] ADERMANN, Nils ; BOGGIANO, Jordi: *Introduction - Composer*. https://getcomposer.org/doc/00-intro.md. Version: August 2015. – [Online; Zugriff 17. August 2015]

[ARA+08] AIYAGARI, Sanjay ; RICHARDSON, Alexis ; ARROTT, Matthew ; RITCHIE, Martin ; ATWELL, Mark ; SADJADI, Shahrokh ; BROME, Jason ; SCHLOMING, Rafael ; CONWAY, Alan ; SHAW, Steven ; GODFREY, Robert ; SUSTRIK, Martin ; GREIG, Robert ; TRIELOFF, Carl ; HINTJENS, Pieter ; RIET, Kim van d. ; O'HARA, John ; VINOSKI, Steve ; RADESTOCK, Matthias: Advanced Message Queuing Protocol / AMQP Working Group. Version: November 2008. http://www.amqp.org/specification/0-9-1/amqp-org-download. AMQP Working Group, November 2008 (0.9.1). – AMQP

[BFHu14] BERJON, Robin ; FAULKNER, Steve ; HICKSON, Ian ; U.A.: HTML5 / W3C. Version: Oktober 2014. http://www.w3.org/TR/2014/REC-html5-20141028. 2014. – Forschungsbericht

[BS13] BOSCHI, Sigismondo ; SANTOMAGGIO, Gabriele: *RabbitMQ Cookbook*. Packt Publishing, 2013. – ISBN 978–1849516502

[DFD+03] DUBE, S. R. ; FELITTI, V. J. ; DONG, M. ; CHAPMAN, D. P. ; GILES, W. H. ; ANDA, R. F.: Childhood Abuse, Neglect, and Household Dysfunction and the Risk of Illicit Drug Use: The Adverse Childhood Experiences Study. In: *PEDIATRICS* 111 (2003), März, Nr. 3, S. 564–572. http://dx.doi.org/10.1542/peds.111.3.564. – DOI 10.1542/peds.111.3.564

[Doc15] DOCKER INC.: *Docker - Build, Ship, and Run Any App, Anywhere*. https://www.docker.com/, August 2015. – [Online; Zugriff 29. August 2015]

[FGM+99] FIELDING, Roy T. ; GETTYS, James ; MOGUL, Jeffrey C. ; NIELSEN, Henrik ; MASINTER, Larry ; LEACH, Paul J. ; BERNERS-LEE, Tim: Hypertext Transfer Protocol – HTTP/1.1 / RFC Editor. RFC Editor, Juni 1999 (2616). – RFC. – ISSN 2070–1721

[FHBH+99] FRANKS, John ; HALLAM-BAKER, Phillip M. ; HOSTETLER, Jeffery L. ; LAWRENCE, Scott D. ; LEACH, Paul J. ; LUOTONEN, Ari ; STEWART, Lawrence C.: HTTP Authentication: Basic and Digest Access Authentication / RFC Editor. RFC Editor, Juni 1999 (2617). – RFC. – ISSN 2070–1721

[Ger09] GERHARDS, R.: The Syslog Protocol / RFC Editor. RFC Editor, März 2009 (5424). – RFC. – ISSN 2070–1721

[Han15] HANE, Oskar: *Build your own PaaS with docker : create, modify, and run your own PaaS with modularized containers using docker*. Birmingham, England Mumbai, India : Packt Publishing, 2015. – ISBN 978–1–78439–394–6

[Hol15] HOLLA, Shrikrishna: *Orchestrating Docker : manage and deploy Docker services to containerize applications efficiently*. Birmingham : Packt Publishing, 2015. – ISBN 978–1–78398–478–7

[II94] ISO/IEC JTC 1 ; ITU-T: Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model. 1994 (ISO/IEC 7498-1:1994). – ISO

[IRLP+13] ISELE, Dorothea ; RUF-LEUSCHNER, Martina ; PRYSS, Rüdiger ; SCHAUER, Maggie ; REICHERT, Manfred ; SCHOBEL, Johannes ; SCHINDLER, Arnim ; ELBERT, Thomas: Detecting adverse childhood experiences with a little help from tablet computers. In: *XIII Congress of European Society of Traumatic Stress Studies (ESTSS) Conference*, 2013, 69–70. – http://dbis.eprints.uni-ulm.de/977/

[KN02] KLYNE, G. ; NEWMAN, C.: Date and Time on the Internet: Timestamps / RFC Editor. RFC Editor, Juli 2002 (3339). – RFC. – ISSN 2070–1721

[Mas12] MASSÉ, Mark: *REST API design rulebook*. Sebastopol, CA : O'Reilly, 2012. – ISBN 978–1449310509

[MR12] MAGTOTO, J. ; ROQUE, A.: Real-time traffic data collection and dissemination from an Android Smartphone using proportional computation and freesim as a practical transportation system in Metro Manila. In: *TENCON 2012 - 2012 IEEE Region 10 Conference*, 2012. – ISSN 2159–3442, S. 1–5

[MZY14] MA, Huadong ; ZHAO, Dong ; YUAN, Peiyan: Opportunities in mobile crowd sensing. In: *Communications Magazine, IEEE* 52 (2014), August, Nr. 8, S. 29–35. – ISSN 0163–6804

[PRH+15] PRYSS, Rüdiger ; REICHERT, Manfred ; HERRMANN, Jochen ; LANGGUTH, Berthold ; SCHLEE, Winfried: Mobile Crowd Sensing in Clinical and Psychological Trials - A Case Study. In: *28th IEEE Int'l Symposium on Computer-Based Medical Systems*, IEEE Computer Society Press, Juni 2015

[PRLS15] PRYSS, Rüdiger ; REICHERT, Manfred ; LANGGUTH, Berthold ; SCHLEE, Winfried: Mobile Crowd Sensing Services for Tinnitus Assessment, Therapy and Research. In: *IEEE 4th International Conference on Mobile Services (MS 2015)*, IEEE Computer Society Press, Juni 2015

[Sch13] SCHINDLER, Arnim: *Technische Konzeption und Realisierung des MACE-Tests mittels mobiler Technologie*. Januar 2013. – Bachelorarbeit an der Uni Ulm

[Sch15] SCHREIBER, Michael: *Realisierung und Evaluierung einer generischen Crowd-Sensing-Anwendung für Android TV*. September 2015. – Masterarbeit an der Uni Ulm

[Sri95] SRINIVASAN, R.: RPC: Remote Procedure Call Protocol Specification Version 2 / RFC Editor. RFC Editor, August 1995 (1831). – RFC. – ISSN 2070–1721

[SSP+13] SCHOBEL, Johannes ; SCHICKLER, Marc ; PRYSS, Rüdiger ; NIENHAUS, Hans ; REICHERT, Manfred: Using Vital Sensors in Mobile Healthcare Business Applications: Challenges, Examples, Lessons Learned. In: *9th Int'l Conference on Web Information Systems and Technologies (WEBIST 2013), Special Session on Business Apps*, 2013, 509–518

[SSP+14] SCHOBEL, Johannes ; SCHICKLER, Marc ; PRYSS, Rüdiger ; MAIER, Fabian ; REICHERT, Manfred: Towards Process-Driven Mobile Data Collection Applications: Requirements, Challenges, Lessons Learned. In: *10th Int'l Conference on Web Information Systems and Technologies (WEBIST 2014), Special Session on Business Apps*, 2014, 371–382

[SSPR15] SCHOBEL, Johannes ; SCHICKLER, Marc ; PRYSS, Rüdiger ; REICHERT, Manfred: Process-Driven Data Collection with Smart Mobile Devices. In: *Web Information Systems and Technologies - 10th International Conference, WEBIST 2014, Barcelona, Spain, Revised Selected Papers*. Springer, 2015 (LNBIP)

[Tid11] TIDWELL, Jenifer: *Designing Interfaces*. O'Reilly Media, 2011. – ISBN 1449379702

[TP] TEICHER, Martin H. ; PARIGGER, Angelika: MACE: Modified Adverse Childhood Experience Scale, version 0.9. In: *Schauer, M., Neuner, F., Elbert, T. (2011 2nd Edition) Narrative Exposure Therapy (NET). A Short-Term Intervention for Traumatic Stress. Cambridge/Göttingen: Hogrefe & Huber Publishers* ISBN 978–1–61676–388–6

[WAV14] WANKE, Egon ; ANDERSEN, Richo ; VOLGNANDT, Tobias: *Blitzortung.org – A World-Wide Low-Cost Community-Based Time-of-Arrival Lightning Detection and Lightning Location Network*. `http://www.blitzortung.org/Documents/TOA_Blitzortung_RED.pdf`. Version: Mai 2014

A

Docker

Docker wird zur Modularisierung von Systemkomponenten verwendet. Im Folgenden werden zunächst die selbst erstellten bzw. modifizierten Dockerfiles erläutert. Anschließend wird die administrative Einrichtung der aggregierten Serverarchitektur (siehe Abschnitt 6.2.1) beschrieben.

A.1 Dockerfiles

Dockerfiles sind Dateien, welche den Aufbau eines Docker-Images beschreiben. In ihnen wird angegeben, auf welchem Grund-Image das beschriebene basiert, welche Programme installiert werden, ob es Volumes enthält, welche Ports freigegeben werden und der Prozess, der beim Start des Containers ausgeführt werden soll. Dockerfiles haben immer den Dateinamen `Dockerfile`.

A.1.1 Dockerfile des `cron`-Containers

Der `cron`-Container führt alle Servermodule aus, die einen Listener für den ESB besitzen, der periodisch neugestartet werden muss (siehe Abbildung 6.1). In Listing A.1 wird das Dockerfile abgebildet, aus dem das Image für diesen Container erstellt werden kann. Die Funktionsweise wird in Form von Kommentaren an den Befehlen beschrieben.

```
1  # Das Image erweitert das Image "dabian:jessie"
2  FROM debian:jessie
3  MAINTAINER "Arnim Schindler" <arnim.schindler@uni-ulm.de>
4
5  # Installation von PHP-FPM und benoetigten Erweiterungen
6  RUN apt-get update -y && apt-get install -y \
7      php5-fpm php5-curl php5-gd php5-geoip php5-imagick \
8      php5-imap php5-json php5-ldap php5-mcrypt php5-memcache \
9      php5-memcached php5-mongo php5-mssql php5-mysqlnd \
10     php5-pgsql php5-redis php5-sqlite php5-xdebug php5-xmlrpc \
11     php5-xcache
12
13 # Installation von fehlenden Systemprogrammen
14 RUN apt-get install -y \
15     cron curl git
16
17 # Installation von Composer
18 RUN curl -sS https://getcomposer.org/installer | php && \
19     mv composer.phar /usr/local/bin/composer
20
21 # Entfernen von Installationsartefakten
22 RUN apt-get remove --purge curl -y && \
23     apt-get clean
24
25 # Einrichten des Cronjobs
```

```
26 RUN echo '* * * * * www-data \
27     php /data/www/laravel/artisan schedule:run \
28     1>> /var/log/cron.log 2>&1' \
29     > /etc/cron.d/artisan
30
31 # Anlegen einer Logfile und Anpassen der Rechte
32 RUN touch /var/log/cron.log
33 RUN chown www-data:www-data /var/log/cron.log
34
35 # Aufrufen des Cronjobs und Ueberwachung der Logfile
36 CMD cron && tail -f /var/log/cron.log
```

Listing A.1: Dockerfile zum Erstellen des `cron`-Containers

Das Überwachen der Logfile ist vor allem wichtig, wenn der Container als Dämon laufen soll. Der `tail`-Prozess hält den Container am Laufen, der andernfalls nach dem Aufruf von `cron` terminieren würde.

A.1.2 Dockerfile des `web`-Containers

Der `web`-Container baut auf dem Image `dylanlindgren/docker-laravel-nginx` auf und wird angepasst, um die Konfiguration für die virtuellen Hosts in den `data-only`-Container auslagern zu können. Dadurch geht deren Konfiguration bei einem Serverupdate durch Austausch des Containers nicht verloren (siehe Abschnitt 2.4).

Alle benötigten Dateien können über Git mit dem Befehl `git clone` heruntergeladen werden: `https://github.com/dylanlindgren/docker-laravel-nginx.git`

An das Ende der Datei `config/nginx.conf` muss vor der schließenden Klammer die folgende Zeile eingefügt werden. Dadurch werden die Konfigurationsdateien für die virtuellen Hosts aus dem `data-only`-Container geladen:
`include /data/vhost-config/sites-enabled/*;`

Das Dockerfile muss auf den Inhalt aus Listing A.2 angepasst werden, um die mitgelieferte Konfigurationsdatei für den virtuellen Host nicht in das Image einzubinden. Diese muss in entsprechenden Pfad im `data-only`-Container kopiert werden.

```
1  FROM debian:jessie
2  # based on "Dylan Lindgren" <dylan.lindgren@gmail.com>
3  MAINTAINER "Arnim Schindler" <arnim.schindler@uni-ulm.de>
4
5  WORKDIR /tmp
6
7  # Install Nginx
8  RUN apt-get update -y && apt-get install -y nginx
9
10 # Apply Nginx configuration
11 ADD config/nginx.conf /opt/etc/nginx.conf
12
13 RUN rm /etc/nginx/sites-enabled/default
14
15 # Nginx startup script
16 ADD config/nginx-start.sh /opt/bin/nginx-start.sh
17 RUN chmod u=rwx /opt/bin/nginx-start.sh
18
19 # PORTS
20 EXPOSE 80
21 EXPOSE 443
22
23 WORKDIR /opt/bin
24 ENTRYPOINT ["/opt/bin/nginx-start.sh"]
```

Listing A.2: Dockerfile zum Erstellen des `web`-Containers

A.1.3 Dockerfile des `data-only`-Containers

Der `data-only`-Container wird benutzt, um die persistenten Daten getrennt von den Containern zu speichern, auf denen die Prozesse laufen. Dadurch können die Prozess-Container ausgetauscht werden, ohne die persistenten Daten zu beeinträchtigen. In Listing A.3 wird das Dockerfile gezeigt, aus dem alle `data-only`-Container der aggregierten Serverstruktur (siehe Abschnitt 6.2.1) erstellt werden. Die Prozess-Container benötigen jeweils nur eines der definierten Volumes. Da die beiden anderen zwar eingebunden, vom Prozess jedoch ignoriert werden, kann ein gemeinsames Image für die drei Prozess-Container erstellt werden.

```
1  FROM debian:jessie
2  MAINTAINER "Arnim Schindler" <arnim.schindler@uni-ulm.de>
3
4  RUN mkdir -p /data
5  VOLUME ["/data"]
6
7  RUN mkdir -p /var/lib/mysql
8  VOLUME ["/var/lib/mysql"]
9
10 RUN mkdir -p /var/lib/rabbitmq
11 VOLUME ["/var/lib/rabbitmq"]
12
13 CMD ["true"]
```

Listing A.3: Dockerfile zum Erstellen des `data-only`-Containers

A.2 Einrichten der aggregierten Struktur

In den vorherigen Abschnitten wurden die Dockerfiles erläutert, die für die selbst erstellten Images benötigt werden. Im Folgenden werden diese um bereitgestellte Images erweitert und zur Ausführung gebracht. Listing A.4 zeigt, wie Images aus den Dockerfiles

erstellt werden. Der Parameter `-t` setzt dessen Namen, das zweite Argument ist der Pfad zum Dockerfile.

```
1 docker build -t arnimschindler/data-only ./path/data-only
2 docker build -t arnimschindler/nginx ./path/nginx
3 docker build -t arnimschindler/cron ./path/cron
```

Listing A.4: Erstellen eigener Images aus Dockerfiles

Im Docker-Hub werden bereits fertige Docker-Images bereitgestellt. Diese werden mit dem Befehl `docker pull` auf das lokale System heruntergeladen und stehen so zur Erstellung von Containern bereit. In Listing A.5 werden die Images für die Container `php`, `mysql` und `rabbit` heruntergeladen.

```
1 docker pull dylanlindgren/docker-laravel-phpfpm && \
2 docker pull mysql && \
3 docker pull rabbitmq:management
```

Listing A.5: Bereitstellen der Images auf dem lokalen System aus dem Docker-Hub

Listing A.6 zeigt das Erstellen und erste Ausführen sämtlicher Docker-Container, die in Abschnitt 6.2.1 beschrieben sind. In den ersten drei Zeilen werden zunächst die `data-only`-Container erstellt. Da die Prozess-Container verschiedene Pfade verwenden, die sich nicht überschneiden, könnte auch nur ein einziger `data-only`-Container verwendet werden. Allerdings könnten die Container dadurch nicht mehr auf unterschiedliche Server verschoben werden. In Zeile 4f wird der `php`-Container erstellt, der in Zeile 7 vom `web`-Container verknüpft wird. Beide binden über `-volumes-from` den `laravel-data`-Container ein. Das Argument `-p 80:80` in Zeile 7 verbindet Port 80 des Betriebsystems mit Port 80 des Docker-Containers und erlaubt damit, dass der Webserver vom Internet über den Standard-HTTP-Port erreichbar ist. Über den Parameter `-e` in den Zeilen 10 und 13-15, werden Umgebungsvariablen an die Container übergeben. Der Parameter `-d` sorgt dafür, dass der entsprechende Container im Hintergrund ausgeführt wird.

```
docker run --name laravel-data arnimschindler/data-only && \
docker run --name rabbit-data arnimschindler/data-only && \
docker run --name mysql-data arnimschindler/data-only && \
docker run --privileged=true --name php --volumes-from \
  laravel-data -d dylanlindgren/docker-laravel-phpfpm && \
docker run --privileged=true --name web --volumes-from \
  laravel-data -p 80:80 --link cs-php:fpm -d \
  arnimschindler/nginx && \
docker run --name rabbit --volumes-from rabbit-data \
  -e RABBITMQ_NODENAME=rabbit-docker \
  -p 5672:5672 -p 8080:15672 -d rabbitmq:management && \
docker run --privileged=true --name mysql --volumes-from \
  mysql-data -p 3306:3306 -e MYSQL_ROOT_PASSWORD=__ROOT-PW__ \
  -e MYSQL_DATABASE=__DB-NAME__ -e MYSQL_USER=__USER-NAME__ \
  -e MYSQL_PASSWORD=__USER-PW__ -d mysql && \
docker run --privileged=true --name cron --volumes-from \
  laravel-data -d arnimschindler/cs-cron
```

Listing A.6: Einrichten der Docker-Container für die aggregierte Serverstruktur

Die Container müssen nur ein mal initial erstellt werden. Zukünftige Starts können mit dem Befehl `docker start laravel-data rabbit-data mysql-data php web ...` durchgeführt werden. Analog können alle Container zusammen mit `restart` neugestartet oder mit `stop` beendet werden.

B

Formulare

B.1 Formulare

Formulare sind ein zentraler Bestandteil der systeminternen Module. Sie werden über ein JSON-Schema definiert, welches aus vier Hauptabschnitten besteht:

B.1.1 Hauptabschnitte

`version` Die Versionsnummer des Protokolls. Damit wird es möglich, die Formulare später anders zu beschreiben und in der Anwendung auf die verschiedenen Versionen einzugehen. Diese Versionsnummer unterscheidet sich von der API-Version.

`header` Metainformationen zum Formular.

fields Ein Array von Formularfeldern mit all ihren Eigenschaften.

compositeFields Ein Array von mehreren `fields`. Diese Gruppen dienen der einfachen Mehrfachverwendung von Formularblöcken. Außerdem besteht so die Möglichkeit, mehreren `fields` gemeinsam eine Überschrift zu geben.

B.1.2 Versionen

Das Formularschema wird mit semantischen Versionsnummern[1] versehen (z.B. 1.3.37). So kann das Schema verändert werden, ohne auf Konflikte durch verschiedene Versionen zu stoßen. Die API-Version ist unabhängig von der des Formularschemas.

B.1.3 Header

- `transactionId` Wird verwendet, um die Fragebogenseiten eindeutig einem `Resultset` zuordnen zu können.
  Wird vom Server generiert, wenn `/api/{vendor}/{module}/[start/]` ohne `transactionId` aufgerufen wird und läuft nach einer vom Modul konfigurierten Zeit ab.
- bei `GET`-Requests über Querystring `transactionId={transactionId}`
- bei `POST`-Requests als JSON-Attribut, wie in Listing B.1 gezeigt

```
1 { "form":{
2     "header":{
3       "transactionId": "h7erpH98zer" }}}
```

Listing B.1: Formularanfrage mit bekannter Transaktions-ID

Die Response enthält zusätzlich die Lebensdauer der Transaktion in Sekunden, sowie das Enddatum als `datetime`, wie in Listing B.2 dargestellt.

[1] `http://semver.org/`

```
{ "form":{
    "header":{
      "transactionId": "h7erpH98zer",
      "transactionTTL": 1800,
      "transactionExpires": "2015-07-28T14:32:48Z" }}}}
```

Listing B.2: Formularantwort mit Transaktions-ID und Lebensdauer

B.1.4 Fields

Formularfelder werden im `fields`-Array beschrieben. Je nach Typ haben diese Felder verschiedene Eigenschaften, die für die Anzeige und Eingabevalidierung von Bedeutung sind. Zunächst werden die allgemeinen Attribute beschrieben, anschließend werden die verfügbaren Datentypen erläutert.

Attribute

- `name` (*required, unique*) Der Name des Formularfeldes. Wird nicht angezeigt, sondern dient der Datenübertragung. Der Wert des Feldes muss eindeutig sein. Wird intern als Variablennamen zur Datenspeicherung verwendet.
- `label` (*optional*) Die Beschriftung des Formularfeldes. Wird als Überschrift des Formularblocks gezeigt, wenn es sich um ein `compositeField` handelt. Enthält den Wert in der Sprache, die über den HTTP-`Accept-Languages`-Header angefordert wurde, falls verfügbar, sonst in der Fallback-Sprache.
- `type` (*required*) Der Typ des Formularfeldes. `type` ist weitgehend eine Untermenge der HTML5 Input-Types[2]
- `compositeField` (*conditional*) Dieses Feld referenziert einen Formularblock, der im JSON unter `compositeFields` definiert ist. Dieses Attribut wird nur beachtet, wenn `"type":"compositeField"` definiert ist.

[2] `http://www.w3.org/TR/html5/forms.html#attr-input-type`

- `pattern` (*optional*) Schränkt die Eigenschaften des Formularfelds durch einen reguläreren Ausdruck (nach JavaScript Pattern Syntax ECMA262[3], wie in HTML5 verwendet) weiter ein.
- `required` (*optional*) `true` oder `false`. Gibt an, ob das Feld ausgefüllt werden muss. Fehlt das Attribut, wird als Standardwert `true` angenommen.

Datentypen

- `compositeField` wird verwendet, um ein `compositeField` zu markieren. Felder vom Typ `compositeField` müssen ein Attribut `compositeField` besitzen, welches die konkrete Feldgruppe referenziert.
- `text` Einzeiliges Eingabefeld für beliebigen Text.
  `\r` und `\n` sind nicht erlaubt, sonst keine Validierung.
  Patternmatching: `^[^\r\n]+$`
  - `value` (*optional*) Vorgegebener Wert
  - `maxlength` (*optional*) Maximale Anzahl an Zeichen
- `textarea` Textbox für Fließtexte
  - `value` (*optional*) Vorgegebener Wert
- `password` Passwortfeld. Eingaben werden als Sterne bzw. Bullets angezeigt, jedoch als Klartext zurückgegeben.
- `number` Eingabefeld für Fließkommazahlen. Wertebereich `-2^1024 < number < 2^1024`, Repräsentation als String.
  Patternmatching: `^[-+]?[0-9]*\.?[0-9]+([eE][-+]?[0-9]+)?$` als praktikable Näherung zur korrekteren Lösung[4]
  Ausführliche Erklärung des HTML5-Number-Feldes: `http://www.w3.org/TR/html5/forms.html#number-state-%28type=number%29`
  - `value` (*optional*) Vorgegebener, gültiger Wert

[3] `http://people.mozilla.org/~jorendorff/es5.1-final.html#sec-15.10.1`
[4] `http://www.w3.org/TR/html5/infrastructure.html#valid-floating-point-number`

  - `min` (*optional*) Kleinster zulässiger Wert
  - `max` (*optional*) Größter zulässiger Wert
  - `step` (*optional*) Schrittweite[5] der Skala. Fließkommazahl oder der String `any`.
    Standardwert ist `1`.
    Alle Werte, die von `base+step*(int)x,` abweichen, sind ungültig und müssen entsprechend behandelt werden. Dabei ist `base` der erste gültige Fließkommawert von `value, min, 0`.

- `email` Eingabefeld für E-Mailadressen
  Patternmatching: `^[a-zA-Z0-9.!#$%&'*+/=?^_`{|}~-]+@[a-zA-Z0-9]`
  `↪ (?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9]`
  `↪ (?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$` [6]
  - `value` (*optional*) Vorgegebener, gültiger Wert

- `tel` Eingabefeld für Telefonnummern. `\r` und `\n` sind nicht erlaubt, sonst keine Validierung.
  Patternmatching: `^[^\r\n]+$`
  - `value` (*optional*) Vorgegebener, gültiger Wert

- `url` Eingabefeld für *absolute* URLs inkl. Protokoll
  Die Prüfung mit Patternmatching ist sehr komplex[7]. Der Server wird daher die PHP-Funktion `filter_var($str, FILTER_VALIDATE_URL)` und eine zusätzlichen Prüfung auf http(s) zur Validierung verwenden. Dabei werden Unicode-URLs leider nicht validiert[8]. Da diese kaum verbreitet sind, wird dieser Kompromiss eingegangen. Die Beschränkung auf http(s) wird vorgenommen, weil es für die

[5] `http://www.w3.org/TR/html5/forms.html#attr-input-step`
[6] `http://www.w3.org/TR/html5/forms.html#valid-e-mail-address`
[7] `https://mathiasbynens.be/demo/url-regex`
[8] `http://php.net/manual/de/filter.filters.validate.php`

meisten Anwendungsfälle ausreicht und dadurch einige Probleme umgangen werden können[9]

  - `value` (*optional*) Vorgegebener, gültiger Wert

- `date` Eingabemöglichkeit für Datum
  Format: `yyyy-mm-dd` oder `--mm-dd` oder `mm-dd`
  Patternmatching: `^(--|[0-9]{4,}-)?[0-9]{2}-[0-9]{2}$`
  Prüfung auf den Wertebereich, um sicherzustellen, dass das Datum existiert.

  - `value` (**optional**) Vorgegebener, gültiger Wert

- `time` Eingabemöglichkeit für Uhrzeiten
  Format: `hh:mm` oder `hh:mm:ss` oder `hh:mm:ss.s` oder `hh:mm:ss.ss` oder `hh:mm:ss.sss`
  Patternmatching: `^[0-9]{2}:[0-9]{2}(:[0-9]{2}(\.[0-9]{1,3})?)?$`
  Prüfung auf den Wertebereich, um sicherzustellen, dass die Uhrzeit existiert[10].

  - `value` (*optional*) Vorgegebener, gültiger Wert

- `datetime` Eingabemöglichkeit für einen Zeitpunkt bestehend aus Datum, Zeit und Zeitzonenverschiebung.
  Format: `{date}T{time}{timezone}`, wobei `timezone` aus der Zeitverschiebung zur UTC-Time besteht im Format `+hh:mm` oder `-hh:mm` oder `Z` als Alias für `+00:00`.
  Patternmatching: `^[0-9]{4,}(-[0-9]{2}){2}T[0-9]{2}:[0-9]{2}`
  ↪ `(:[0-9]{2}(\.[0-9]{1,3})?)?(Z|[+-][0-9]{2}:[0-9]{2})$`
  W3 Spezifikation[11] erweitert RFC 3339[12]: `T` und `Z` müssen Großbuchstaben sein; Die Jahreszahl hat *mindestens* vier Ziffern und ist größer `0`.

  - `value` (*optional*) Vorgegebener, gültiger Wert

[9] http://www.d-mueller.de/blog/warum-url-validierung-mit-filter_var-keine-gute-idee-ist/
[10] http://www.w3.org/TR/html5/infrastructure.html#valid-time-string
[11] http://www.w3.org/TR/html5/infrastructure.html#global-dates-and-times
[12] http://tools.ietf.org/html/rfc3339

- `select` Auswahl einer Option von mehreren vorgegebenen Möglichkeiten, am besten als Dropdown.
  - `options` (*required*) Array von Auswahlmöglichkeiten
    * `value` (*required*) Der fest vorgegebene Rückgabewert. Darf nicht verändert werden.
    * `label` (*required*) Die Beschriftung der Auswahlmöglichkeit. Ist i.d.R. die Repräsentation von `value` in der angeforderten Sprache.
- `multiselect` Auswahl einer oder mehrerer Optionen von mehreren vorgegebenen Möglichkeiten
  Analog zu `select`. Rückgabewert ist Array statt String. Gegebenenfalls andere Darstellung im Client.
- `checkbox` Binäres Eingabefeld zum Ein-/Ausschalten von Optionen.
  - `value` (*required*) Der fest vorgegebene Rückgabewert. Darf nicht verändert werden.
- `radio` Wie `select`, jedoch kein Dropdown sondern flache Auflistung der Optionen.

B.1.5 CompositeFields

Jedes Objekt im `compositeFields`-Array beschreibt eine Gruppe aus Formularfeldern, die unter `fields.{FIELD}.compositeField` referenziert werden kann. Die `CompositeFields` können als selbst definierter `type` angesehen werden, welcher aus mehreren `fields` zusammengesetzt wird. Dies erlaubt eine einfache Wiederverwendung, sowie das Hinzufügen einer Überschrift zu Feldgruppen.
`CompositeFields` können mehrfach referenziert werden. Im Antwort-JSON werden `CompositeFields` dereferenziert, wodurch die Eindeutigkeit der Felder gewährleistet wird.

- `name` Der eindeutige Name des Formularblocks.
  Wird von `fields.{FIELD}.compositeField` referenziert.

- `fields` Array von Formularfeldern, wie in Abschnitt B.1.4 beschrieben.

Verhalten von Required

- Ist an einem `CompositeField` das `required`-Attribut auf `true` gesetzt (oder per Default `true`), so müssen alle Felder innerhalb von `CompositeField` gemäß deren `required`-Attribute ausgefüllt werden.
  ⇒ Alles muss ausgefüllt werden entsprechend `required`.
- Ist an einem `CompositeField` das `required`-Attribut auf `false` gesetzt, so müssen alle Felder innerhalb von `CompositeField` gemäß deren `required`-Attribute ausgefüllt werden, sobald mindestens eines der Felder innerhalb von `CompositeField` ausgefüllt ist. Dabei spielt es keine Rolle, ob das `required`-Attribut des Feldes `true` oder `false` ist.
  ⇒ Alles oder nichts muss ausgefüllt werden entsprechend `required`.

Enthält ein `CompositeField` ausschließlich `required`-Attribute mit dem Wert `false`, macht es folglich keinen Unterschied, ob das `required`-Attribut des `CompositeField` selbst `true` oder `false` ist.

Einschränkung

Die Verschachtelungstiefe beträgt momentan 1, was bedeutet, dass ein Formularfeld vom Typ `compositeField` *eine* Ebene an Kindfeldern haben kann. Bei den Kindfeldern sind nur noch einfache Feldertypen erlaubt (kein `"type":"compositeField"`).

Ein Beispiel für Zusammengesetzte Felder wird in Listing B.3 gezeigt.

```
1 {"form": {
2   "version": "0.0.2",
3   "fields": [
4   { "name":"myFirstCompositeField",
5     "type":"compositeField",
6     "compositeField":"someCompositeField"},
```

```
7     { "name":"mySecondCompositeField",
8       "type":"compositeField",
9       "compositeField":"someCompositeField" }],
10
11    "compositeFields":[
12    { "name": "someCompositeField",
13      "fields": [
14      { "name": "myTextField",
15        "label": "Single line text",
16        "type": "text"},
17      { "name": "myTextArea",
18        "label": "Multi line text",
19        "type": "textarea"}]
20    }]
21  }}
```

Listing B.3: Beispiel für `"type":"compositeField"`

B.1.6 Gesamtbeispiel

In Listing B.4 wird ein Gesamtbeispiel des Formularschemas gezeigt. Dabei wird nicht auf alle Datentypen eingegangen, da sich diese analog verhalten und die Übersichtlichkeit des Beispiels weiter einschränken würden. In Listing B.5 wird der HTML-Code dargestellt, der aus dem beschriebenen JSON-Schema generiert wird.

```
1   {"form": {
2       "version": "0.0.2",
3       "fields": [
4       { "name": "fancyInput",
5         "label": "Fancy input",
6         "type": "text",
7         "value": "sample text"},
```

```
8       { "name": "fancyInputAgain",
9         "label": "One more fancy input",
10        "type": "text",
11        "required": false },
12      { "name": "someOption",
13        "label": "Gimme an option",
14        "type": "multiselect",
15        "options": [
16        { "label": "one", "value": "o1" },
17        { "label": "two", "value": "o2" },
18        { "label": "three", "value": "o3" }],
19        "value": ["o2"]},
20      { "label": "text field",
21        "name": "fancyText",
22        "type": "textarea" },
23      { "name": "fancyAddress",
24        "type": "compositeField",
25        "compositeField": "address",
26        "label": "First address" },
27      { "name": "AddressAgain",
28        "type": "compositeField",
29        "compositeField": "address",
30        "requiered": false },
31      { "name": "fancyBanking",
32        "type": "compositeField",
33        "compositeField": "banking",
34        "requiered": false }],
35
36      "compositeFields": [
37      { "name": "address",
38        "fields": [
```

```
      { "name": "givenName",
        "label": "Given Name",
        "type": "text" },
      { "name": "surname",
        "label": "Surname",
        "type": "text" },
      { "name": "street",
        "label": "Street",
        "type": "text" },
      { "name": "streetNumber",
        "label": "Street Number",
        "type": "text",
        "pattern": "[0-9]+(/[0-9]*[a-z]?)?" },
      { "name": "tel",
        "label": "TelNr",
        "type": "tel",
        "required": false }
    ]},
    { "name": "banking",
      "fields": [
      { "name": "ownerName",
        "label": "Owner name",
        "type": "text",
        "maxlength": 100,},
      { "name": "iban",
        "label": "IBAN",
        "type": "text",
        "pattern": "[a-zA-Z]{2}[0-9]{2}[a-zA-Z0-9]{4}[0-9]{7}([
           a-zA-Z0-9]?){0,16}" },
      { "name": "bic",
        "label": "BIC/SWIFT",
```

```
69          "type": "text",
70          "pattern": "([a-zA-Z]{4}[a-zA-Z]{2}[a-zA-Z0-9]{2}([a-zA
                -Z0-9]{3})?)" },
71        { "name": "bank",
72          "label": "Bank Name",
73          "type": "text" }]
74      }]
75  }}
```

Listing B.4: Gesamtbeispiel des Formularschemas

Das Formularschema aus Listing B.4 wird in einem HTML5-Client in den HTML-Code aus Listing B.5 übersetzt. Dabei ist die Besonderheit zu beachten, dass bei Feldgruppen, die auf `"required": false` gesetzt sind, das Attribut `data-opt` eingeführt wird. Da HTML selbst keine Abhängigkeiten des `required`-Attributs unterstützt, kann diese Funktionalität anhand des `data-opt`-Attributs mithilfe von JavaScript hinzugefügt werden.

```
1  <form>
2  Fancy input:
3  <input type="text" name="fancyInput" required
4    value="smaple text"><br>
5  One more fancy input:
6  <input type="text" name="fancyInputAgain"><br>
7  Gimme some option:
8  <select name="someOption" multiple>
9  <option value="o1">one</option>
10 <option value="o2" selected>two</option>
11 <option value="o3">three</option>
12 </select>
13 text field:<br>
14 <textarea name="fancyText" required></textarea><br>
15
```

```
<fieldset>
<legend>First address</legend>
Given Name:
<input type="text" name="fancyAddress_givenName" required><br>
Surname:
<input type="text" name="fancyAddress_surname" required><br>
Street:
<input type="text" name="fancyAddress_street" required><br>
Street Number:
<input type="text" name="fancyAddress_streetNumber" required
  pattern="[0-9]+(\\[0-9]*[a-z]?)?" ><br>
TelNr:
<input type="tel" name="fancyAddress_tel">
</fieldset><br>

<fieldset>
Given Name:
<input type="text" name="AddressAgain_givenName"
  data-opt="required"><br>
Surname:
<input type="text" name="AddressAgain_surname"
  data-opt="required"><br>
Street:
<input type="text" name="AddressAgain_street"
  data-opt="required"><br>
Street Number:
<input type="text" name="AddressAgain_streetNumber"
  data-opt="required"
  pattern="[0-9]+(\\[0-9]*[a-z]?)?" ><br>
TelNr:
<input type="tel" name="AddressAgain_tel">
```

```
47 </fieldset><br>
48
49 <fieldset>
50 Owner name:
51 <input type="text" name="fancyBanking_ownerName"
52   maxlength="100" data-opt="required"><br>
53 IBAN:
54 <input type="text" name="fancyBanking_iban" data-opt="required"
55   pattern=
56  "[a-zA-Z]{2}[0-9]{2}[a-zA-Z0-9]{4}[0-9]{7}([a-zA-Z0-9]?){0,16}"
57  ><br>
58 BIC/SWIFT:
59 <input type="text" name="fancyBanking_bic" data-opt="required"
60   pattern=
61   "([a-zA-Z]{4}[a-zA-Z]{2}[a-zA-Z0-9]{2}([a-zA-Z0-9]{3})?)" >
62 <br>
63 Bank Name:
64 <input type="text" name="fancyBanking_bank" data-opt="required">
65 </fieldset><br>
66 </form>
```

Listing B.5: HTML-Darstellung des Schemas aus Listing B.4

B.1.7 Antwort-Schema

Die ausgefüllten Formulare werden mit einem `PUT` bzw. `POST` Request als JSON an den Server geschickt. Das Schema ist analog zum Formularschema, wobei die Daten der zusammengesetzten Felder direkt in die `fields` eingesetzt werden. Entgegen der Formularbeschreibung sind die Antworten für jede Instanz eines `compositeFields` individuell und können daher dereferenziert und direkt eingefügt werden. Die Zusätzlichen Informationen, wie `label`, sind in der Antwort nicht relevant, da anhand von

`name` eine eindeutige Zuordnung möglich ist. Einfache `field`-Elemente, die nur einem Rückgabewert haben, geben als Antwort einen `String` zurück, Mehrfachauswahlen ein `Array`, welches die ausgewählten `option`-Elemente enthält. `CompositeField`-Elemente geben ein `Array` zurück, dessen Elemente wiederum aufgebaut sind wie die Antworten der `field`-Elemente. In Listing B.6 wird das Antwort-JSON zu dem Formular aus Listing B.4 abgebildet.

```
{ "formreply": {
  "version": "0.0.2",
  "fields": [
  { "name": "fancyInput",
    "value": "i wrote some fancy stuff" },
  { "name": "fancyInputAgain",
    "value": "i wrote even more fancy stuff, 'cause i can" },
  { "name": "someOption", "value": ["o1, o2] },
  { "name": "fancyText",
    "value": "so text\nmuch wow!\ncan even write multiple lines
        here" },
  { "name": "fancyAddress",
    "value": [
     { "name": "givenName", "value":" Sherlock" },
     { "name": "surname", "value": "Holmes" },
     { "name": "street",  "value": "Baker Street" },
     { "name": "streetNumber", "value": "221b" } ]
    }]
}}
```

Listing B.6: Antwortschema eines Formulars

B.1.8 Fehlermeldungen

Werden fehlerhafte Daten übertragen, antwortet der Server mit dem HTTP-Statuscode `400 Bad Request`. Dieser gibt ggf. zusätzliche Informationen als JSON zurück. Da es auch zu Fehlern kommen kann, die schon vom Server abgefangen werden, bevor die Antwort mit JSON angereichert wird, darf sich nicht darauf verlassen werden, dass die Antwort immer JSON enthält.

Listing B.7 Zeigt die Fehlermeldung zu einer Anfrage, bei der das Feld `fancyInput`, sowie die Felder `iban` und `swift` innerhalb der Feldgruppe `fancyBanking` falsch ausgefüllt wurden. Dies kann sowohl bedeuten, dass das `required`-Attribut verletzt wurde, als auch, dass der Inhalt nicht dem gewünschten Format entspricht.

```
{ "formreply": {
  "errors": {
    "inputValidation": [
    { "name": "fancyInput" },
    { "name": "fancyBanking",
      "fields": [
      {"name":"iban"},
      {"name":"swift"} ]}
    ]}
}}
```

Listing B.7: Fehlermeldung einer verletzten Eingabevalidierung

Wird eine Anfrage an den Server geschickt, obwohl deren Transaktion schon abgelaufen ist, wird dieser mit einer Fehlermeldung antworten, wie sie in Listing B.8 gezeigt wird.

```
{ "formreply": {
  "errors": {
    "transaction":"expired" }
}}
```

Listing B.8: Fehlermeldung bei abgelaufener Transaktion

Abbildungsverzeichnis

Name: Arnim Schindler Matrikelnummer: 650432

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Arnim Schindler