LEUPHANA
UNIVERSITÄT LÜNEBURG

Bachelorarbeit

Marcel Schlegel

Entwicklung, Implementation und Integration einer Nachrichtenschnittstelle zur Kommunikation zwischen der GET.ON GesundheitsTraining.Online-Plattform und dem MobileCoach

Fakultät Wirtschaftswissenschaften
Institut für Wirtschaftsinformatik

Faculty of Business and Economics
Institute of Information Systems

Marcel Schlegel

Entwicklung, Implementation und Integration einer Nachrichtenschnittstelle zur Kommunikation zwischen der GET.ON GesundheitsTraining.Online-Plattform und dem MobileCoach

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Wirtschaftsinformatik (WI)
am Institut für Wirtschaftsinformatik
der Fakultät Wirtschaftswissenschaften
der Leuphana Universität Lüneburg

Betreuender Prüfer: Prof. Dr. Burkhardt Funk
Zweitgutachter: M.Sc. Psychologie Leif Boß

Verfasser: Marcel Schlegel
Matrikelnummer: 3027712
Kontakt: marcel.schlegel@stud.leuphana.de

Abgegeben am 05.06.2019

Inhaltsverzeichnis

Anhang

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

1 Einleitung

Die webbasierte GET.ON GesundheitsTraining.Online-Plattform wird u. a. für Studien zur Stressbewältigung eingesetzt. Der MobileCoach soll, für bestimmte Aufgaben, in die Studien der GET.ON-Plattform eingebunden werden. Dazu gehört z. B. der Empfang von Benachrichtigungen ausgehend von der GET.ON-Plattform.
Die Arbeit behandelt die Entwicklung, Implementation und Integration einer Schnittstelle, die die Einbindung und Kommunikation zwischen beiden Plattformen ermöglicht.

1.1 Motivation

Eine generelle Herausforderung von Maßnahmen der Gesundheitsförderung wie auch speziell von Online-Gesundheitstrainings besteht darin, die Studienteilnehmer zu unterstützen und zu motivieren, das Training komplett zu durchlaufen inkl. Bearbeitung aller Trainingseinheiten sowie das Erlernte möglichst oft im Alltag anzuwenden und zu üben.
Um eine hohe Adhärenz beim Studienteilnehmer zu erreichen, soll der MobileCoach bzw. die MobileCoach-App diesen dabei unterstützen. Dies geschieht, indem der Studienteilnehmer über die App z. B. motivierende Nachrichten empfängt.
Ein positiver Effekt solcher Nachrichten wurde bereits in einer Studie von (Adler et al., 2017), die den Einfluss von mobilen Textnachrichten zur Unterstützung der Medikamentenadhärenz von Herz-Kreislauf-Erkrankungen untersucht, nachgewiesen. Die Adhärenz war bei sechs von sieben Versuchsgruppen gestiegen.
Die Möglichkeit externe Systeme, wie die GET.ON-Plattform, in den MobileCoach einzubeziehen, gibt es nicht. Eine Kommunikation zwischen beiden Plattformen kann bis jetzt nicht stattfinden.

1.2 Zielsetzung

Damit der MobileCoach um die Nachrichtenschnittstelle erweitert werden kann, müssen die Abläufe und Funktionen der Software nachvollziehbar sein. Dies wird üblicherweise durch

eine entsprechende Dokumentation gewährleistet. Aktuell existiert solch eine Dokumentation für den MobileCoach nicht.
Das Ziel der Arbeit ist die technische Architekturbeschreibung des MobileCoach. Darauf aufbauend wird die Nachrichtenschnittstelle zwischen dem MobileCoach und der GET.ON-Plattform entworfen und implementiert.
Die Architekturbeschreibung umfasst dabei nicht den kompletten Umfang einer Architekturanalyse oder einer Bewertung. Sie soll lediglich einen Überblick über das System und die Komponenten vom MobileCoach bieten und zukünftige Erweiterungen vereinfachen. Der Entwurf beschreibt, wie eine Schnittstelle zum Austausch von Nachrichten zwischen zwei Systemen aufgebaut wird, sowie die eingesetzten Technologien. Die Implementation umfasst den Austausch von Daten bzw. Nachrichten zwischen beiden Plattformen.

1.3 Aufbau

Die Bachelorarbeit gliedert sich in 8 Kapitel. Das nachfolgende Kapitel 2 behandelt die Grundlagen zur Umsetzung der technischen Architekturbeschreibung des MobileCoach sowie eine Einführung in beide Plattformen. Kapitel 3 bietet einen technischen Überblick über den MobileCoach und dokumentiert die wichtigsten Abläufe. In Kapitel 4 werden die Anforderungen für die Entwicklung einer Nachrichtenschnittstelle für MobileCoach sowie die GET.ON-Seite definiert. Durch die in Kapitel 3 erstellte Dokumentation erfolgt darauf aufbauend in Kapitel 5 ein Entwurf der Nachrichtenschnittstelle sowie in Kapitel 6 deren Realisierung und Integration. Kapitel 7 beschreibt den Softwaretest der definierten Anforderungen. Abschließend wird in Kapitel 8 das Fazit gezogen und ein Ausblick auf weitere Entwicklungen gegeben.

2 Grundlagen

2.1 Softwarearchitektur

Diese Arbeit widmet sich zum Teil der Dokumentation von Software. Zuerst gilt zu klären, was Softwaredokumentation ist und in welchem Zusammenhang sie zur Softwarearchitektur steht.
Die Definitionen der Softwarearchitektur sind zahlreich. Das (Software Engineering Institute [SEI], 2010) beschreibt über 30 Definitionen. Jedoch haben alle Beschreibungen gemein, dass eine Architektur etwas Strukturelles abbildet. Der Kern solch einer Struktur definiert sich aus unterschiedlichen Komponenten, deren Beziehungen zueinander sowie Eigenschaften beider.
(Starke, 2017, S. 16–20) beschreibt Architektur als ein grundlegendes Mittel zur Definition von Bausteinen eines Systems. Diese Bausteine werden geprägt durch Entwurfsentscheidungen, abgebildet durch (dokumentierte) Komponenten, Schnittstellen und deren Relationen untereinander sowie visualisiert mit Hilfe von Sichten, die einen Überblick des Gesamtsystems aus verschiedenen Perspektiven ermöglichen.
Diese Aspekte der Softwarearchitektur werden in einer entsprechenden Softwaredokumentation festgehalten. Der Architekturprozess sollte von einer Dokumentation begleitet werden, um die Transparenz und Verständlichkeit des Gesamtsystems sicherzustellen.

2.1.1 Arc42

Arc42 ist ein Standard zur Erstellung von Dokumentation der Softwarearchitektur (Starke & Hruschka, 2016, S. 1). Als Vorlage bietet dieser Standard Methoden und Vorgehensweisen, die den Prozess der Architekturbeschreibung begleiten und vereinfachen. Da Softwaresysteme in ihrem Lebenszyklus einer ständigen Weiterentwicklung unterliegen, sind Systemerweiterungen und die Implementation neuer Funktionalitäten bei marginaler oder fehlender Dokumentation ein Risiko. Selbst kleine Änderungen verursachen einen hohen Änderungs- und Implementationsaufwand. Dieser Umstand kann das Softwaresystem im Extremfall unwirtschaftlich machen.
Dennoch ist die Dokumentation das erste, was bei Softwareprojekten vernachlässigt wird (Glass, 1989).
Arc42 verhindert solch eine Negativentwicklung, indem es sich u. a. an denen in Kapitel 2.1 erläuterten Aspekten der Softwarearchitektur orientiert.

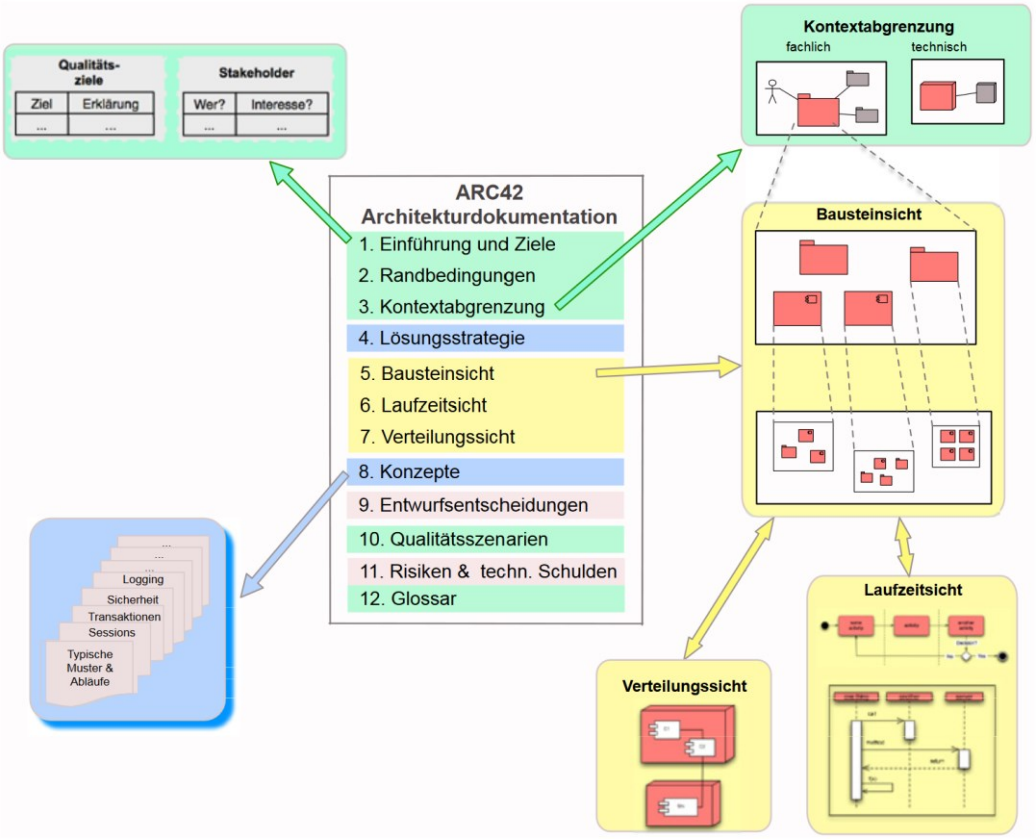


Abbildung 2.1 Gesamtstruktur von arc42 nach Starke & Hruschka (2016, S. 2)

Arc42 gliedert sich in zwölf Kapitel, die wie ein Schubladensystem funktionieren (Starke & Hruschka, 2016, S. 2–3). Jede Schublade enthält zusammengehörige Informationen. Die Reihenfolge, in der die Schubladen mit Informationen befüllt werden, ist nicht vorgegeben. Bei der Dokumentation eines vorhanden Softwaresystems, kann z. B. ein initialer Sprung in das Kapitel Bausteinsicht erfolgen. Alle Kapitel sind optional und werden je nach Bedarf ausgefüllt oder ausgelassen.
Arc42 sieht sich nicht als ein „Alles-ausfüllen-Formular“, sondern als flexible Lösung, die Zusammenhänge im Softwaresystem verständlich, für alle beteiligten Stakeholder, darstellt.
Die in Abbildung 2.1 dargestellte Gesamtstruktur von arc42, wird in folgender Tabelle beschrieben:

Überschrift	Erläuterung
1. Einführung und Ziele	Dieses Kapitel enthält maßgebliche Forderungen der Auftraggeber. Es werden wichtige Punkte der Anforderungsdokumentation zusammengefasst.
2. Randbedingungen	Technische und organisatorische Randbedingungen, die Auswirkungen auf Architekturentscheidungen besitzen, werden beschrieben. Einschränkungen der Entwurfsfreiheit werden aufgeführt.

	Darunter fallen technische, organisatorische oder juristische Randbedingungen sowie einzuhaltende Standards.
3. Kontextabgrenzung	Zeigt das Gesamtsystem aus der Vogelperspektive, als Blackbox sowie die Relationen zu Nachbarsystemen, wichtigen Stakeholdern und technischer Infrastruktur. Der fachliche und technische Kontext wird dargestellt.
4. Lösungsstrategie	Knappe Darstellung einer Lösung der Kernidee. Was sind die zentralen Lösungsideen, Gestaltungskriterien oder Herangehensweisen? Eine detaillierte Ausführung sollte auf die Bausteine bzw. technischen Konzepte verschoben werden.
5. Bausteinsicht	Zerlegung des Systems in Bausteine (Subsysteme, Module, Komponenten, Pakete, Klassen, Funktionen) sowie deren Relationen und Abhängigkeiten. Das Gesamtsystem wird als Whitebox dargestellt, weitere Ebenen werden wechselnd als Black- und Whitebox-Sicht abgebildet und die Granularität schrittweise verfeinert.
6. Laufzeitsicht	Das Zusammenspiel der Architekturbausteine wird in Laufzeitszenarien dargestellt. Erfolgsszenarien der zentralen Use-Cases sowie weitere relevante Abläufe sollten gezeigt werden.
7. Verteilungssicht	Zeigt in welcher Umgebung ein System läuft. Hardware (Rechner, Netze etc.) und Software (Betriebssysteme, Datenbanken, Middleware etc.) werden abgebildet.
8. Konzepte	Erklärung von Konzepten (Persistenz, Logging, Internationalisierung etc.), Technologien und fachlichen Modellen. Dazu können Verweise in die Bausteinsicht sowie den Quellcode erfolgen.
9. Entwurfsentscheidungen	Wichtige Entwurfs- oder Architekturentscheidungen werden erläutert sowie deren (mögliche) Alternativen. Entscheidungen über die Struktur von Bausteinen sollte in der jeweiligen Whitebox-Sicht der Bausteinsicht erfolgen.
10. Qualitätsszenarien	Qualitätsziele bzw. Qualitätsanforderungen können ausführlich beschrieben werden. Dazu können Qualitätsbäume mit den wichtigsten Zielen oder Mindmaps definiert werden.
11. Risiken und technische Schulden	Technische Risiken sowie deren Auswirkungen und Gegenmaßnahmen werden beschrieben. Dazu gehören u. a. Architektur- und Implementationsrisiken sowie Risiken der Infrastruktur.
12. Glossar	Definitionen domänenspezifischer Begriffe, verständlich für alle Stakeholder.

Tabelle 2.1 Gliederung der Architekturbeschreibung mit arc42 nach Starke & Hruschka (2016, S. 44–113; 2017, S. 151–152)

Arc42 kann als Vorlage für die gängigsten Textverarbeitungsprogramme zum Download bezogen werden (Starke & Hruschka, 2019b). Außerdem wird eine Vielzahl an webbasierten Textvorlagen angeboten, die eine einfache Integration in bestehende Plattformen zur Softwareentwicklung (Bitbucket, Github etc.) ermöglichen.
Arc42 steht unter der freien Lizenz „Creative-Commons Sharealike 4.0“ (Starke & Hruschka, 2019a).

2.2 MobileCoach-Plattform

Der MobileCoach (Kowatsch, 2019) ist eine Softwareplattform für Verhaltensinterventionen und bietet die Möglichkeit, digitale Interventionen zu erstellen. Teilnehmer einer Intervention können diese auf einem Endgerät, z. B. einem Computer oder Smartphone, über den Webbrowser oder auf einer mobilen App abrufen.

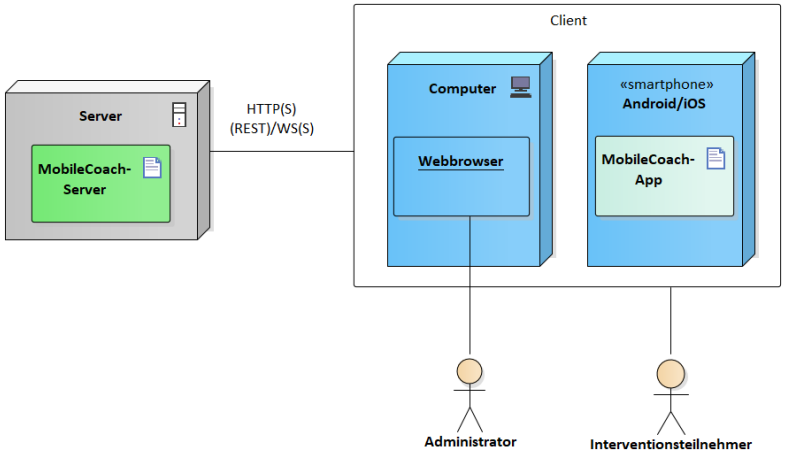


Abbildung 2.2 Vereinfachtes Kontextdiagramm der MobileCoach-Plattform

2.2.1 Netzwerkarchitektur

Die MobileCoach-Plattform baut auf einer Client-Server-Architektur auf. Der MobileCoach-Server ist dabei die zentrale Komponente, die die Logik einer Intervention abbildet.
Der MobileCoach-Client besitzt eine lose Kopplung zum Server und übernimmt die Darstellung der Intervention. Der Vorteil dieser verteilten Architektur besteht in der Unabhängigkeit beider Komponenten (Client, Server). Der Client kann in seiner Implementation leichtgewichtig und ressourcenschonend sein. Durch die Zentralität des Servers ist die Zugriffskontrolle auf Interventionsdaten einfach umsetzbar, da jeder Client authentifiziert wird. Interventionen können unabhängig auf dem Server geändert werden, ohne dass der Client aktualisiert werden muss. Die Zentralisierung des Servers hat allerdings auch Nachteile. So ist der Server als kritisch einzustufen, da bei einem Ausfall kein Zugriff auf eine Intervention stattfinden

kann. Dieses Risiko kann jedoch durch eine Redundanz des Servers verringert werden (Spanke, 2015).

2.2.2 MobileCoach-Client

Die MobileCoach-Plattform unterscheidet folgende Arten von Clients:

Verwalter einer Intervention Die Intervention wird über den Webbrowser durch eine Administrationsoberfläche verwaltet. Dies übernimmt z. B. ein Administrator des MobileCoach-Servers.

Interventionsteilnehmer Der Interventionsteilnehmer ist eine Person, die über eine Clientanwendung an einer vom MobileCoach-Server ausgehenden Intervention teilnimmt.
Für die Interventionsteilnahme werden unterschiedliche Clientanwendungen genutzt:

- **Webbrowser** Über einen Webbrowser kann der Interventionsteilnehmer an einer webbasierten Intervention (siehe WebSurvey Kapitel 2.2.4.1) teilnehmen. Dafür kommuniziert der Browser über das HTTP(S)-Protokoll mit dem MobileCoach-Server (Abbildung 2.2).

- **MobileCoach-App** Interventionen werden über die App chatbasiert, als multimedialer Dialog, durchgeführt. Ein Chatbot leitet den Interventionsteilnehmer durch einen Dialog, der regelbasiert einen Interventionsablauf erzeugt. Die Ablaufregeln sind nicht auf Clientseite festgelegt, sondern werden vom MobileCoach-Server vorgegeben. Die App ist mit Hilfe des mobilen Applikationsframework React Native umgesetzt (Behrends, 2018, S. 1–6). Das Framework unterstützt die App-Entwicklung für Android und iOS-basierte Geräte. JavaScript, kombiniert mit dem Webframework React, wird eingesetzt, um eine mobile plattformübergreifende Entwicklung nativer Apps zu ermöglichen. Der webbasierte Ansatz der App lässt eine leichte Integration webbasierter Inhalte zu. Diese können z. B. während einer Intervention, über den Interventionschat, abgerufen werden. Die Kommunikation zum Server findet über das HTTP(S)- als auch über das WebSocket-Protokoll (WS(S)) statt (Abbildung 2.2).

2.2.3 MobileCoach-Server

Der MobileCoach-Server verwaltet die Interventionslogik und ermöglicht deren Administrierung. Ein MobileCoach-Client verbindet sich zum Server, um eine Intervention zu starten. Eine detailliertere Dokumentation des Servers ist in Kapitel 3 beschrieben.

2.2.4 Interventionsadministration

Die Administration von Interventionen erfolgt über die vom MobileCoach-Server bereitgestellte, webbasierte Oberfläche. Für die Administrierung kann der Server mehrere Benutzer verwalten. Über ein Rechtesystem können verschiedene Benutzerrollen vergeben werden. Folgende Rollen werden unterschieden:

Administrator Der Administrator besitzt die höchsten Systemrechte. Er kann Benutzer anlegen und löschen. Weiter kann dieser Interventionen anlegen, bearbeiten und löschen. Ein Administratorkonto wird initial vom Server angelegt.

Autor Ein Autor hat Zugriff auf die Interventionen. Diese können von ihm verwaltet und bearbeitet werden.

Team Manager Der Team Manager kann auf das Dashboard einer Intervention zugreifen, um diese auszuwerten. Das Dashboard zeigt Statistiken und Ergebnisse einer Intervention an.

2.2.4.1 Interventionserstellung

Interventionen werden serverseitig, über die Administrationsoberfläche, erstellt und finden zwischen n Interventionsteilnehmern statt. Eine Intervention besitzt einen Namen, eine Beschreibung und optional ein Passwort. Weiter besteht diese aus folgenden Komponenten:

WebSurveys Die Intervention kann über einen WebSurvey bereitgestellt werden. Ein WebSurvey besteht aus mehreren Folien. Jede Folie repräsentiert eine HTML-Seite, mit einer Interventionsfragestellung. Die Folien bzw. HTML-Seiten können über die Administrationsoberfläche angepasst werden und über Interventionsvariablen Informationen bzw. Daten einer Intervention oder eines Interventionsteilnehmers beinhalten (z. B. Teilnehmername). Weiter sind regelbasierte Verzweigungen auf andere Folien möglich. Der Zugriff auf WebSurveys erfolgt durch einen Webbrowser über eine zuvor vom MobileCoach-Server erstellte URL.

Nachrichten Interventionsnachrichten werden während einer Intervention an Interventionsteilnehmer per Kurznachrichtendienst (SMS) oder der MobileCoach-App gesendet. Ähnlich wie die Folien eines WebSurveys können Nachrichten individuell angepasst werden. Eine Definition von Bedingungen, bei denen eine Nachricht gesendet wird, ist möglich. Weiter können mehrere Nachrichten gruppiert werden, wobei z. B. die Möglichkeit besteht, Nachrichten dieser Gruppe in einer zufälligen Reihenfolge zu versenden.

Dialoge Interventionsdialoge stellen den Kern einer Intervention über die MobileCoach-App dar. Diese bestehen aus einzelnen Interventionsnachrichten und können regelbasierte

Verzweigungen besitzen, um einen dynamischen Interventionsverlauf zu erzeugen. Eine Intervention kann aus mehreren Dialogen bestehen.

Variablen Interventionsvariablen werden zum Speichern von Werten, die während einer Intervention entstehen, genutzt. Das können z. B. Antwortwerte auf eine WebSurvey-Folie oder eine Interventionsnachricht eines Interventionsteilnehmers sein. Die Interventionsstatistik bezieht ihre Werte u. a. aus den Variablen. Weitere Details finden sich im Kapitel 3.5.4.

Entscheidungsbaum/Regeln Interventionsregeln sind primärer Bestandteil der Intervention. Diese werden als Wurzelknoten eines Entscheidungsbaums dargestellt. Jede Interventionsregel (Wurzelknoten) kann weitere, vom Administrator oder Autor definierte Subregeln (Kindknoten) aufnehmen. Regeln werden während einer Intervention von bestimmten Ereignissen ausgelöst. Zum Beispiel wenn der Interventionsteilnehmer zum ersten Mal eine Intervention über die MobileCoach-App aufruft. Dabei können Regeln auf bestimmten Ereignissen beruhen und bei einem erwarteten Wert bzw. einer auf Wahr ($True$) evaluierenden Regelbedingung eine Aktion ausführen. Dies kann z. B. das Verschicken einer Nachricht oder Starten eines Interventionsdialogs sein.

Teilnehmer Interventionsteilnehmer können sich zu einer erstellten Intervention per WebSurvey oder der MobileCoach-App verbinden. Dabei bekommt jeder Teilnehmer eine eindeutige Interventionsteilnehmer-ID vom MobileCoach-Server zugewiesen. Weiter kann von jedem Teilnehmer eine Interventionsstatistik erzeugt werden.

2.2.4.2 Interventionsablauf

Interventionen finden zwischen dem MobileCoach-Server und einem MobileCoach-Client statt. Dabei unterscheidet sich der Ablauf je nach Clientanwendung. Bei der Intervention über einen WebSurvey ruft der Interventionsteilnehmer die Startseite des Surveys über eine URL auf. Danach durchläuft dieser jede Folie bzw. HTML-Seite des Surveys. Der Übergang zwischen den Folien erfolgt regelbasiert und schließt die Antwort des Teilnehmers auf die jeweilige Interventionsfrage als Bedingung mit ein. Eine Intervention über die MobileCoach-App verläuft ebenfalls regelbasiert. Über den Entscheidungsbaum der Intervention werden Ereignisse (z. B. das Öffnen der Chatansicht in der MobileCoach-App) verarbeitet und es wird mit einer entsprechend definierten Aktion reagiert (z. B. der Versendung einer Interventionsnachricht).
Folgendes Ablaufdiagramm zeigt eine App-basierte Intervention:

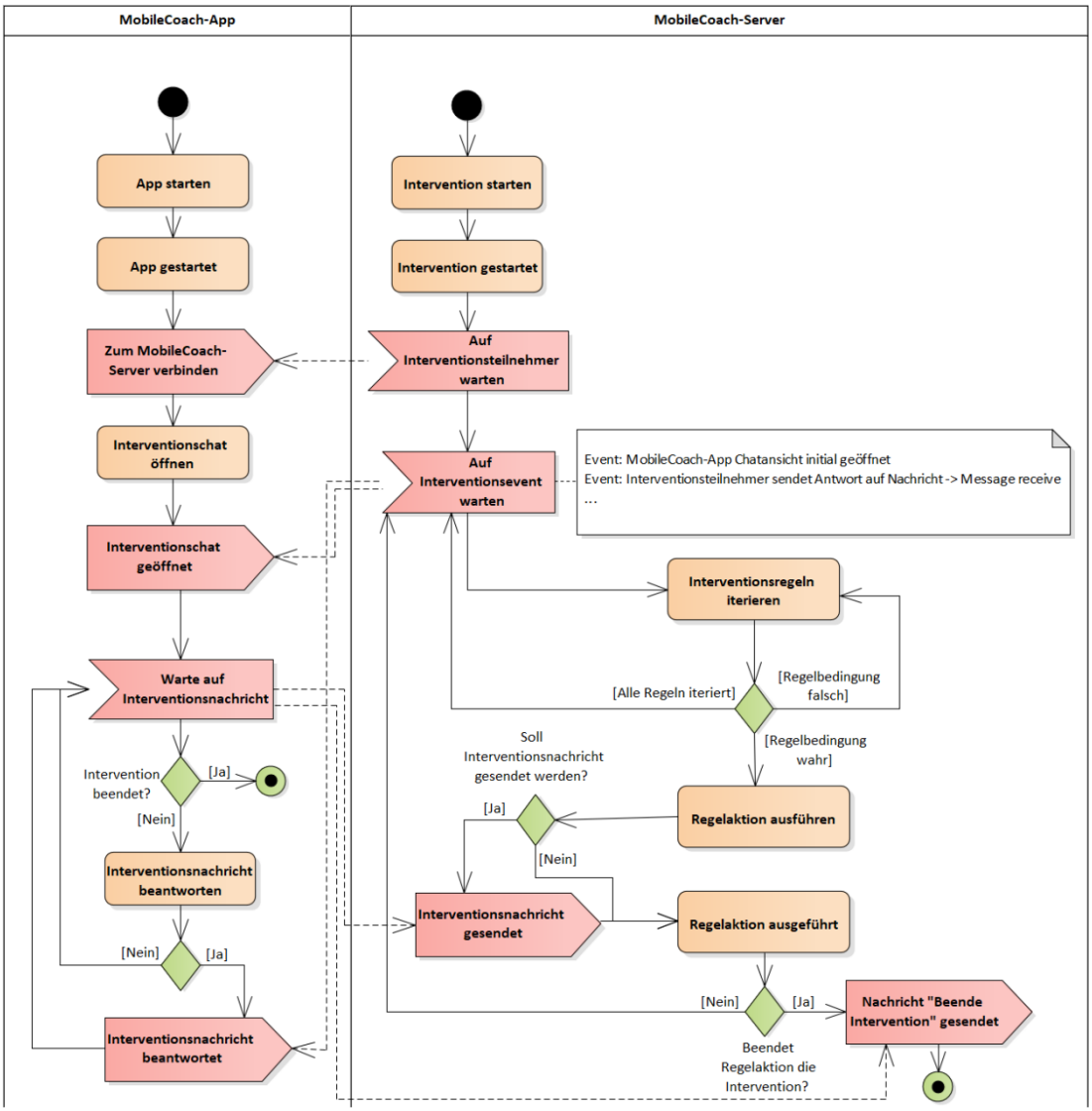


Abbildung 2.3 Ablaufdiagramm einer App-basierten Intervention

Folgende Schritte werden durchlaufen:

Schritt	Beschreibung
1	Der MobileCoach-Server wartet auf die Verbindung eines Interventionsteilnehmers.
2	Ist ein Teilnehmer verbunden, wird auf ein Interventionsevent gewartet.
3	Der Teilnehmer löst ein Event, z. B. durch das Öffnen des Interventionschats, in der MobileCoach-App aus.
4	Das Event wird zum Server gesendet und an den Entscheidungsbaum der Intervention weitergegeben.

5	Ist eine entsprechende Regel vorhanden, die auf das Event reagiert, und eine Regelbedingung evaluiert auf Wahr, wird diese ausgeführt.
6	Beinhaltet die Regel das Versenden einer Nachricht, wird diese gesendet.
7	Wird die Intervention durch die Regel nicht beendet, wird auf weitere Events gewartet.
8	Die MobileCoach-App wartet auf eingehende Nachrichten.
9	Wird eine Nachricht empfangen, die Intervention durch diese Nachricht nicht beendet und auf die Nachricht keine Antwort erwartet, wird auf weitere Nachrichten gewartet.
10	Muss eine Nachricht beantwortet werden, wird bei einer Antwort des Interventionsteilnehmers diese an den Server geschickt.
11	Der Server interpretiert die Nachrichtenantwort als weiteres Event.

Tabelle 2.2 Ablaufschritte einer App-basierten Intervention

Die Intervention kann durch eine Regelaktion auf Serverseite beendet werden. Wird die Intervention beendet, schickt der Server dem Client eine entsprechende, für den Teilnehmer nicht sichtbare, Nachricht. Der Client beendet daraufhin die Intervention.

2.3 GET.ON-Plattform

Die Abteilung für Gesundheitspsychologie des Instituts für Psychologie der Leuphana Universität Lüneburg setzt für Online-Gesundheitstrainings zur Förderung der psychischen Gesundheit und zum Umgang chronischer Erkrankungen die webbasierte Plattform GET.ON GesundheitsTraining.Online ein. GET.ON wird u. a. für Studien zur Stressbewältigung eingesetzt.
Die Studienteilnahme erfordert die Erstellung eines GET.ON-Profils für einen Studienteilnehmer. Dieses wird durch einen Administrator erstellt. Ein spezifisches Trainingsmodul wird für das Profil freigeschaltet. Die Logindaten werden dem Studienteilnehmer im Anschluss übermittelt. Dieser kann sich dann über einen Webbrowser mit den Logindaten in sein GET.ON-Profil einloggen und das für ihn bereitgestellte Trainingsmodul durchlaufen.

2.3.1 GET.ON-Server

Serverseitig basiert die GET.ON-Plattform auf dem Content-Management-System (CMS) WordPress. Über dieses wird die GET.ON-Plattform verwaltet. Dazu zählt die Benutzer-, Rechte-, Oberflächen- und Datenverwaltung. Für die Strukturierung der GET.ON-Webinhalte (u. a. Trainingsmodule) bietet WordPress die Erstellung von „Pages“ (Seiten) an (WordPress Foundation, 2019a). Diese beinhalten eine statische Oberflächenbeschreibung, die durch HTML definiert ist. Neben dieser können dynamische Inhalte durch PHP oder JavaScript eingebettet werden.

Für die Veröffentlichung der in WordPress erstellten Seiten werden diese in den Status „Öffentlich“ versetzt. Ein Abruf der Seiten über einen Webbrowser ist damit möglich.

Identifikation von Ressourcen Um eine Identifikation von Webinhalten zu ermöglichen, vergibt WordPress jeder über eine URL abrufbaren Ressource eine eindeutige ID (Sibley, 2018). Diese ist als `page_id`-Parameter in der URL angegeben.
Beispiel-URL: `https://coachstaging.geton-training.de/?page_id=45284`

Integration von Quellcode Neben der Möglichkeit Quellcode direkt in eine Seite einzubetten, nutzt der GET.ON-Server das WordPress-Plugin „Code Snippets“ (Bunge, 2019). Dieses ermöglicht die Separierung zwischen Oberflächenbeschreibung und Quellcode. WordPress wird um die Option erweitert, Snippets anzulegen. Diese beinhalten nur Quellcode in Form von PHP oder JavaScript. Über eine Referenz durch „Shortcodes“ (z. B. `[shortcode_name]`) können diese in „Pages“ eingebettet werden.

WordPress-API und Datenbank Als Datenbank kommt eine Standardinstallation von MySQL zum Einsatz (WordPress Foundation, 2019c). WordPress Datenbanktabellen lassen sich durch das Prefix `wp_` identifizieren. Über die PHP-basierte WordPress-API werden z. B. Datenbankmanipulationen vorgenommen. Dafür steht eine umfangreiche Sammlung von Funktionen zur Verfügung. Eine primäre Schnittstelle ist die `wpdb`-Klasse (WordPress Foundation, 2019b). Diese ermöglicht die WordPress-basierte Interaktion mit der Datenbank und stellt neben Abfragefunktionalitäten sicherheitsrelevante SQL-Validierungsfunktionen bereit. Diese verhindern z. B. SQL-Injection[1] Angriffe.

[1] „SQL Injection wird eingesetzt, um Content-Management-Systeme mit SQL-Datenbank-Anbindung anzugreifen. SQL Injection wird erst durch Ausnutzen einer Sicherheitslücke möglich, die durch mangelnde Maskierung bzw. Überprüfung in Benutzereingaben entsteht.“(Hochschule für angewandte Wissenschaften Augsburg (2019))

3 Dokumentation des MobileCoach-Servers

Um die Implementierung der Schnittstelle auf MobileCoach-Serverseite zu realisieren, wird eine Dokumentation der Software benötigt. Diese orientiert sich am Entwurf von arc42 (Kapitel 2.1.1) und beinhaltet alle Beschreibungen, die für die Realisierung erforderlich sind. Dabei liegt der Fokus auf der technischen Seite des Servers. Die fachlichen Aspekte sind für diese Arbeit zu vernachlässigen.

3.1 Randbedingungen

Randbedingung	Beschreibung
Implementation in Java	Der MobileCoach-Server wird mit einem zu Java SE (Standard Edition) 8 kompatiblen JDK (Java Development Kit) entwickelt.

Tabelle 3.1 Randbedingung des MobileCoach-Servers

3.2 Technischer Kontext

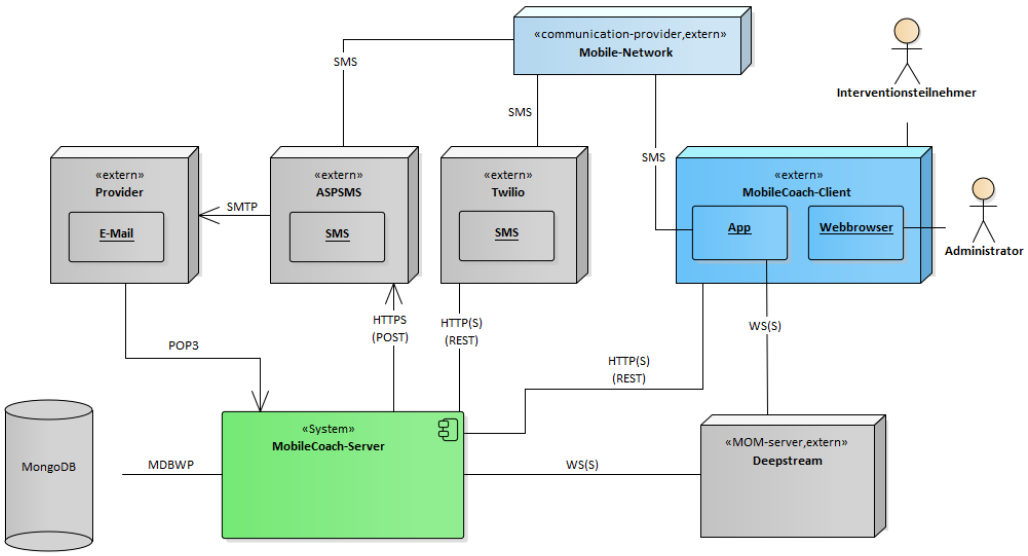


Abbildung 3.1 Technisches Kontextdiagramm der MobileCoach-Serverumgebung

MobileCoach-Client (*externes System*)
Interventionsteilnehmer werden über ein grafisches Frontend mit dem MobileCoach-Server verbunden. Dafür existieren unterschiedliche Arten von Clientanwendungen (Kapitel 2.2.2). Über den Client werden Nachrichten vom Server empfangen und gesendet.
Die bidirektionale Kommunikation erfolgt über das HTTP-Protokoll per REST-Schnittstelle und das WebSocket-Protokoll.

Twilio (*externes Fremdsystem*)
Twilio ist eine Cloud-Kommunikationsplattform, die Lösungen für einen multimedial basierten Nachrichtenaustausch ermöglicht (Twilio Inc., 2019). Es steht u. a. eine Schnittstelle für den Kurznachrichtendienst (SMS) zur Verfügung. Die bidirektionale Kommunikation erfolgt über das HTTP-Protokoll per REST-Schnittstelle.

ASPSMS (*externes Fremdsystem*)
ASPSMS ist eine Kommunikationsplattform, die Schnittstellen für einen Nachrichtenaustausch per SMS anbietet. Die Kommunikation erfolgt in Senderichtung über das HTTP-Protokoll per POST-Methode (VADIAN.NET AG, 2019b). In Empfangsrichtung erfolgt die Kommunikation E-Mail-basiert über das POP3-Protokoll. Dafür wird eine empfangene Nachricht an einen beliebigen E-Mail-Provider geleitet. Der MobileCoach-Server kann daraufhin das Postfach des Providers abrufen (VADIAN.NET AG, 2019a).

Deepstream (*externes System*)
Deepstream ist eine Message Oriented Middleware (MOM), die Dienste zum Speichern von Dokumenten, Senden von Nachrichten und das Synchronisieren von Daten anbietet (deepstreamHub GmbH, 2019b). Die bidirektionale Kommunikation erfolgt über das WebSocket-Protokoll.

MongoDB (*externes System*)
Als dokumentenorientierte Datenbank wird MongoDB zum persistenten Speichern von Entitäten des MobileCoach-Servers genutzt (MongoDB Inc., 2019). Die bidirektionale Kommunikation erfolgt über das proprietäre MongoDB-Wire-Protokoll (MDBWP).

3.3 Lösungsstrategie

3.3.1 Architektur des MobileCoach-Servers

Der MobileCoach-Server folgt dem Entwurf der Schichtenarchitektur (Richards, 2015, S. 1–2). Als eine der meist genutzten Architekturentwürfe ist die Schichtenarchitektur (auch *n-tier* Architektur) ein fester Standard für viele Java-EE-Anwendungen. Diese lässt eine grobe und erste Gliederung des Quelltextes zu. Die Architektur gliedert sich in einzelne Komponenten. Diese sind horizontal, auf Schichten, aufgeteilt. Die Anzahl der Schichten ist nicht vorgegeben. Jede Schicht übernimmt eine spezifische Rolle und ist für eine bestimmte Aufgabe verantwortlich. Das vereinfacht die Erweiterung und Wartung des Systems, da die Auffindung von Quelltext, durch Schichten als grobe Abstraktion, erleichtert wird.
Die Komponenten bilden die Java-Pakete (*packages*) des Servers ab.
Dieser ist in drei Schichten unterteilt (Filler et al., 2015):

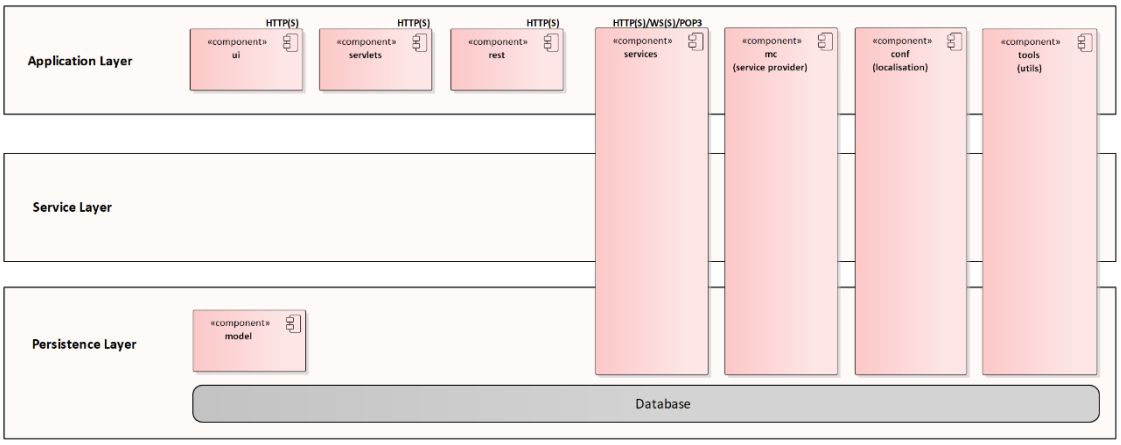


Abbildung 3.2 Schichtenarchitektur des MobileCoach-Servers

Die Komponenten `services`, `mc`, `conf` und `tools` bestehen aus Subkomponenten und Klassen, die nicht trennscharf definiert sind und sich auf mehrere Schichten aufteilen.

Application Layer Die Schicht übernimmt das Anzeigen der Administrationsoberfläche (`ui`-Komponente), die Kommunikation zu externen Fremdsystemen u. a. das Empfangen und Versenden von Nachrichten über verschiedene Kommunikationskanäle (`services-` und `rest`-Komponente), die Bereitstellung von WebSurveys sowie das Streamen und Bereitstellen von multimedialen Daten (Videos, Bilder) (`servlets`-Komponente).

Service Layer Der Service Layer ist für die Verwaltung der Geschäftslogiken zuständig. Administrationsdienste für WebSurveys und Interventionen delegieren CRUD[2]-Funktionalitäten an den Application Layer und sorgen für die Kommunikation mit der Persistenz. Laufzeitdienste überwachen WebSurveys und Interventionen. Dies umfasst die Nachrichtenkommunikation und die Bereitstellung der Webseite eines Surveys. Weiter werden Interventionsregeln abgebildet und durch Entscheidungsbäume evaluiert.

Persistence Layer Der Persistence Layer verwaltet *alle Domain-Objekte*. Dazu gehören persistente Entitäten sowie transiente Datentypen. CRUD-Methoden werden von einem `ModelObject`-Datentyp angeboten und über einen auf dem Service Layer liegenden `DatabaseManagerService` delegiert. Der `FileStorageManagerService` persistiert multimediale Daten (Videos, Bilder) auf dem lokalen Dateisystem des MobileCoach-Servers.

Um den Zugriff von Schichten untereinander zu regulieren, wird ein weiteres Konzept der Schichtenarchitektur angewandt. Das Konzept *layers of isolation* unterscheidet zwischen offenen (*open*) und geschlossenen (*closed*) Schichten. Ist eine Schicht als geschlossen markiert, darf diese nur auf ihre nächste unterliegende Schicht zugreifen. Beispielsweise greift die `ui`-Komponente des Application Layer über den Service Layer auf die Persistenz zu.
Der Application Layer ist mit dem Persistence Layer nicht gekoppelt.
Wäre in diesem Fall der Service Layer als offen markiert, dürfte der Application Layer direkt auf den Persistence Layer zugreifen.

Die umgesetzte Schichtenarchitektur bietet folgende Vor- und Nachteile:

Vorteile *1)* Hoher Bekanntheitsgrad, geringe Komplexität → leicht verständlich für Entwickler*; 2)* Klare Trennung von Verantwortlichkeiten → Hohe Wart-, Test- und Erweiterbarkeit.

Nachteile *1)* Geringe Agilität durch enge Kopplung der Schichten → Hoher Änderungsaufwand*; 2)* Hoher Deployment-Aufwand durch monolithische Struktur → Separates Deployment von Schichten bzw. Komponenten nicht möglich.

[2] Im Zusammenhang mit der dokumentenorientierten Datenbank MongoDB kann anstelle von CRUD-Methoden (Create, Read, Update, Delete) auch von IFUR-Methoden (Insert, Find, Update, Remove) gesprochen werden (Jaspers (2012)). In dieser Arbeit wird, wegen dem höheren Bekanntheitsgrad und gleichbleibender Bedeutung, der Begriff CRUD benutzt.

3.4 Bausteinsicht

Der MobileCoach Server wird in einzelne Komponenten/Subkomponenten zergliedert.
Wie in Kapitel 3.3.1 erwähnt bilden die Komponenten/Subkomponenten die Java-Pakete des Quelltextes ab.

3.4.1 Ebene 1

Der MobileCoach-Server zerfällt in acht Komponenten. Blaue, gestrichelte Linien zeigen die Abhängigkeiten zu den, im technischen Kontext beschriebenen (Kapitel 3.2), externen Systemen. Für eine bessere Übersicht sind die Abhängigkeiten der Komponenten untereinander nicht als Linien, sondern textuell in den jeweiligen Komponenten beschrieben.

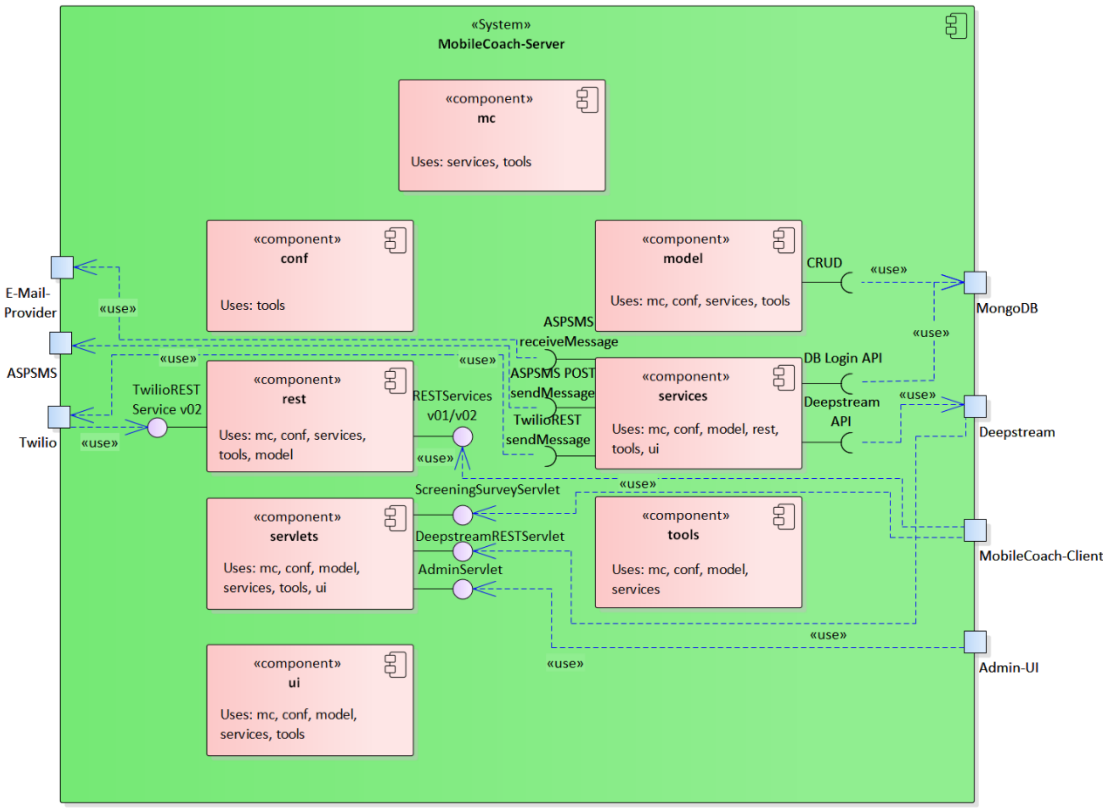


Abbildung 3.3 Bausteinsicht des MobileCoach-Servers

Komponente (Blackbox)	Kurzbeschreibung
mc	Initialisiert das System. Liest die Konfiguration ein und startet Systemdienste.

conf	Beinhaltet systemweite Konfigurations- und Implementationskonstanten und ist für die Internationalisierung verantwortlich.
model	Beinhaltet systemweite Entitäten und Datentypen.
rest	Stellt Dienste der REST-Schnittstelle bereit.
services	Stellt Systemdienste und Datentypen für eine fehlerfreie Systemausführung bereit.
servlets	Stellt Servlets zur Verfügung, die HTTP-Anfragen verarbeiten.
tools	Komponente, die systemweit Utility-Klassen bereitstellt.
ui	Realisiert Darstellung der webbasierten Administrationsoberfläche.

Tabelle 3.2 Bausteinsicht Ebene 1

3.4.2 Ebene 2

3.4.2.1 Model (Whitebox)

Die Komponente `model` zerfällt in die Subkomponenten `memory`, `persistent`, `rest` und `ui`.

Subkomponente	Kurzbeschreibung
memory	Enthält systemweite transiente Datentypen und Entitäten, die nur zur Laufzeit des Systems existieren.
persistent	Beinhaltet systemweite persistente Datentypen und Entitäten.
rest	Verwaltet Datentypen mit einem REST-Kontextbezug. Diese werden vorwiegend in der `rest-` sowie `servlets-`Komponente benutzt.
ui	Enthält Abbildungen von Attributen einer Entität, die in der Administrationsoberfläche über eine bestimmte Oberflächenkomponente (z. B. eine Tabelle) angezeigt werden sollen.

Tabelle 3.3 Model-Bausteinsicht Ebene 2

3.4.2.2 Services (Whitebox)

Die Komponente `services` zerfällt in die Subkomponenten `internal`, `threads` und `types`.

Subkomponente	Kurzbeschreibung
internal	Enthält interne Systemdienste, die Funktionen für den MobileCoach-Server bereitstellen.
threads	Verwaltet das Threading für den Empfang, das Senden sowie die Überwachung von Nachrichten.

types	Enthält Enumerationstypen, die Platzhalter und Felder einer Session sowie der WebSurveys abbilden. Weiter werden die Systemvariablen abgebildet.

Tabelle 3.4 Services-Bausteinsicht Ebene 2

3.4.2.3 UI (Whitebox)

Die Komponente `ui` zerfällt in die Subkomponenten `components`, `components/basics` und `components/main_view/interventions`. Durch den Schrägstrich (/) getrennte Komponenten sind weitere verschachtelte Subkomponenten. Diese werden zur Vereinfachung in der aktuellen Ebene (Ebene 2) angezeigt.

Subkomponente	Kurzbeschreibung
components	Enthält Basisklassen für die Erweiterung der Administrationsoberfläche. Diese werden von jeder Oberflächenkomponente abstrahiert.
components /basics	Enthält Oberflächenelemente, die die Standard-Oberflächenbibliothek erweitert.
compnents /main_view /interventions	Enthält alle Oberflächenkomponenten einer Intervention, die die Anzeige in der Administrationsoberfläche ermöglichen.

Tabelle 3.5 UI-Bausteinsicht Ebene 2

3.5 Konzepte

3.5.1 Interventionsregeln

Wie in Kapitel 2.2.4.1 bereits erwähnt stellt der Entscheidungsbaum des Servers mit seinen Interventionsregeln eine zentrale Funktionalität zur Verfügung. Dabei unterscheidet der Server zwischen folgenden Arten von Interventionsregeln:

Tagesbasis (*Daily basis*) Regeln, die in einem 24-Stunden-Intervall evaluiert werden.

Periodenbasis (*Periodic basis*) Regeln, die in einem 5-Minuten-Intervall evaluiert werden.

Unerwartete Nachricht (*Unexpected message*) Regel, die bei Nachrichten, die keine Nutzerintention oder keine Interventionsnachricht abbilden, ausgelöst wird.

Nutzerintention (*User intention*) Regeln, die bei einer Aktion des Interventionsteilnehmers in der MobileCoach-App ausgelöst werden. Darunter fallen z. B. Aktionen wie das Öffnen der Interventionschat-Ansicht oder das Schließen der Webansicht im Interventionschat.

3.5.2 Message Oriented Middleware

Für die asynchrone Verarbeitung von Interventionsnachrichten wird eine Message Oriented Middleware (MOM) eingesetzt (Oracle Corporation, 2010). Diese sorgt für eine lose Kopplung zwischen einem MobileCoach-Client (z. B. MobileCoach-App) und dem MobileCoach-Server, da eine direkte Kommunikation zwischen beiden entfällt. Weiter wird eine verbesserte Performance der Nachrichtenverarbeitung erreicht, da der MOM-Server für die Annahme simultaner Verbindungen optimiert ist und eine effiziente Implementierung eines Nachrichtenpuffers und -routings umsetzt. Der MobileCoach-Server setzt als MOM-Server Deepstream ein und nutzt folgende Funktionalitäten:

Remote Procedure Call Die Nachrichtenkommunikation zwischen MobileCoach-App und Server wird über Remote Procedure Calls (RPC) ausgeführt, die dem Request-Response-Nachrichtenmuster zuzuordnen sind. Der MobileCoach-Server registriert über die `provide`-Methode des Deepstream-Servers eine RPC-Methode. Diese wird durch einen Methodennamen, z. B. `user-message`, definiert. Die Methode kann weitere Parameter (z. B. Nachrichtentext, Zeitstempel etc.) als JSON-Objekt (JavaScript Object Notation) erwarten. Ein Client ruft über eine entsprechende Deepstream-Client-Programmbibliothek die Methode `make` auf, der der RPC-Methodenname `user-message` sowie eine JSON-Datenstruktur übergeben wird. Abschließend kann über einen Rückgabewert die erfolgreiche oder nicht erfolgreiche Methodenausführung ausgewertet werden (deepstreamHub GmbH, 2019f).

Records Der Deepstream-Server bietet neben der Nachrichtenkommunikation die Möglichkeit, dokumentenbasierte Objekte als JSON-Datenstruktur transient zu speichern. Der MobileCoach-Server nutzt zusätzlich die Möglichkeit, diese Objekte persistent in der MongoDB-Datenbank zu halten. Über die Methode `getRecord` des Deepstream-Servers wird ein JSON-Objekt angelegt sowie bezogen. Dafür wird der `getRecord`-Methode ein eindeutiger Name übergeben. Dieser dient für das JSON-Objekt als ID. Anschließend wird ein Record-Objekt erzeugt und zurückgegeben, das `Set/Get`-Methoden für ein JSON-Objekt bereitstellt. Der MobileCoach-Server nutzt Records für die Persistierung von Authentifizierungsdaten (z. B. Login-Tokens), auf die während eines Deepstream-Logins unkompliziert zugegriffen werden kann. Ein direkter Zugriff auf die Datenbank kann somit vermieden werden (deepstreamHub GmbH, 2019d).

Berechtigungen Deepstream ermöglicht es für seine Funktionalitäten, wie RPC-Methoden oder Records, Berechtigungen zu definieren. Für RPC-Methoden existieren *publish*- und *subscribe*-Berechtigungsregeln. Für Records existieren *create*-, *write*-, *read*- und *delete*-Berechtigungsregeln. Diese lassen sich u. a. durch Bedingungen mit Hilfe der Programmiersprache JavaScript weiter einschränken (deepstreamHub GmbH, 2019g). Berechtigungen werden in einer YAML-Datei (Yet Another Multicolumn Layout) folgendermaßen definiert:

```
1 rpc:
2   "user-message":
3     request: "user.data.role === 'participant'"
4 record:
5   "record-name":
6     read: "user.data.role === 'server'"
```

Listing 3.1 Beispielhafte YAML-Berechtigungsdatei

In diesem Beispiel kann die RPC-Methode `user-message` nur von einem Deepstream-Client, der als Rollen-Parameter „participant“ angibt bzw. ein Interventionsteilnehmer ist, aufgerufen werden.
Der Record kann nur vom MobileCoach-Server bzw. nur mit der Rollen-Parameterangabe „server“ gelesen werden.

3.5.3 Threading

Während der Laufzeit einer Intervention muss der Server eine Vielzahl an eingehenden und ausgehenden Daten verarbeiten. Der Hauptteil dieser Daten besteht aus Interventionsnachrichten, die während einer Intervention zwischen Client und Server ausgetauscht werden. Weiter werden Interventionsnachrichten- und Systemzustände überwacht. Darunter fallen z. B. Interventionsnachrichten, die in einem bestimmten Zeitfenster von einem Interventionsteilnehmer beantwortet werden müssen. Läuft das Zeitfenster ab, reagiert der Server mit einem vorher definierten Verhalten (z. B. dem Senden einer Erinnerungsnachricht an den Interventionsteilnehmer). Interventionsaktivitäten werden vom Server gesammelt und protokolliert (z. B. für eine anschließende Ausgabe von Statistiken). Für den Server-Administrator werden System-Metriken (z. B. angemeldete Benutzer, Anzahl versendeter Interventionsnachrichten etc.) gesammelt und in Protokolldateien (Logs) geschrieben.
Ein synchroner Ablauf dieser Prozesse wäre nicht sinnvoll, da das System regelmäßig blockiert wäre. Würde z. B. das Warten auf eine Antwort einer Interventionsnachricht synchron umgesetzt, wäre das System während dieser Wartezeit blockiert. Eine weitere Ausführung von Interventionsprozessen könnte nicht stattfinden. Daher setzt der Server ein Threading um, das Aufgaben asynchron verwaltet. Dafür beinhaltet die Subkomponente `services/threads` entsprechende Thread-Klassen:

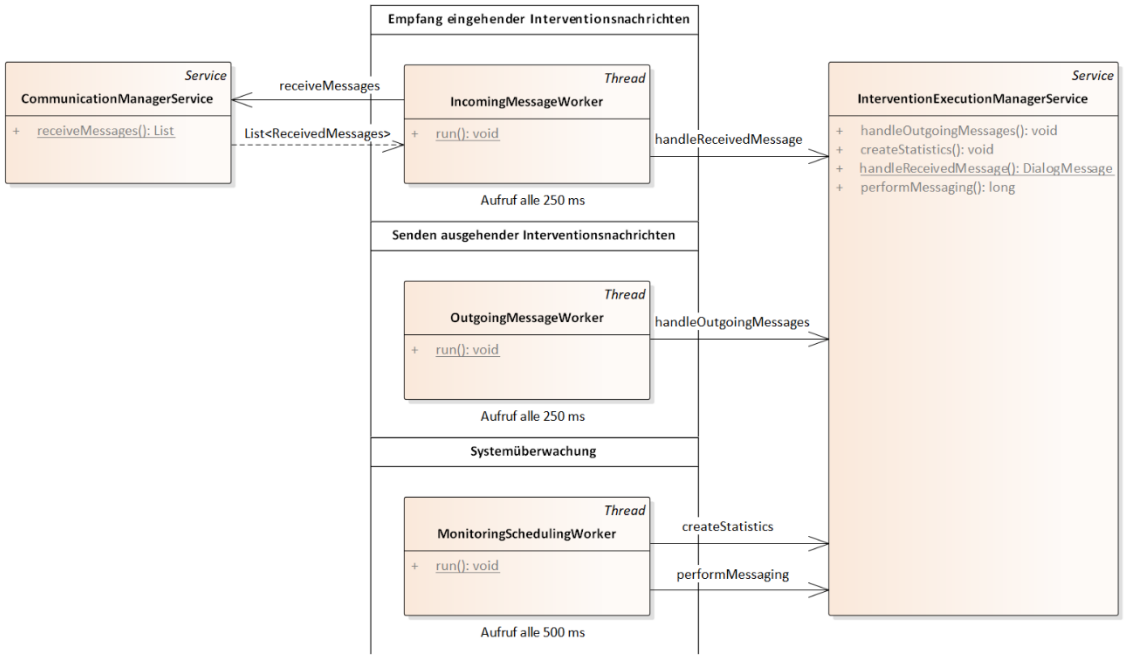


Abbildung 3.4 Threading des MobileCoach-Servers

Empfang eingehender Interventionsnachrichten (`IncomingMessageWorker`)
Die Klasse `IncomingMessageWorker` sammelt alle 250 Millisekunden die empfangenen Interventionsnachrichten über die Methode `receiveMessages` vom `CommunicationManagerService` ein. Im Anschluss werden diese iteriert und über die Methode `handleReceivedMessage` an den `InterventionExecutionManagerService` übergeben sowie verarbeitet.

Senden ausgehender Interventionsnachrichten (`OutgoingMessageWorker`)
Die Klasse `OutgoingMessageWorker` ruft alle 250 Millisekunden die Methode `handleOutgoingMessages` vom `InterventionExecutionManagerService` auf. Dieser verwaltet und versendet alle ausgehenden Interventionsnachrichten.

Systemüberwachung (`MonitoringSchedulingWorker`)
Die Klasse `MonitoringSchedulingWorker` protokolliert alle 30 Sekunden die System-Metriken. Falls aktiviert, wird über die Methode `createStatistics` eine tägliche Interventionsstatistik angelegt. Alle 7 Tage werden nicht abgeschlossene WebSurveys beendet. Alle 500 Millisekunden wird die Methode `performMessaging` vom `InterventionExecutionManagerService` aufgerufen. Diese prüft die auf Tages- sowie Periodenbasis laufenden Interventionsregeln, um diese bei Bedarf zu evaluieren und auszuführen. Weiter werden Interventionsnachrichten, die eine Antwort in einem bestimmten Zeitfenster erwarten, verarbeitet. Nach Ablauf des Zeitfensters wird eine entsprechend definierte Aktion (z. B. Schicken einer Erinnerungsnachricht an den Interventionsteilnehmer) ausgeführt.

3.5.4 Variablen

Die MobileCoach-Plattform verarbeitet zur Laufzeit sämtliche Interventionsdaten. Dazu gehören nicht nur Stammdaten, die eine weniger hohe Änderungsfrequenz aufweisen (z. B. Backend-Benutzer oder Interventionseigenschaften (u. a. Interventionsname)), sondern vor allem Bewegungsdaten einer laufenden Intervention. Das können z. B. Informationen über den Interventionsteilnehmer sein (u. a. Name, Geschlecht oder Alter) oder die Antwort eines Interventionsteilnehmers auf eine Interventionsnachricht.
Diese dynamisch anfallenden Daten lassen sich im MobileCoach-Server als Variablen anlegen. Eine Variable besteht aus einem Namen und einem Wert und bildet somit ein Schlüssel-Wert-Paar ab. Das Namensschema einer Variable definiert sich durch ein führendes Dollarzeichen und einen nachfolgenden Namen, der keine Leerzeichen enthält (z. B. `$meineVariable`). Variablennamen sind systemweit eindeutig.
Der Autor bzw. Administrator einer Intervention kann Variablen beim Erstellen von Interventionsnachrichten und Dialogen in der Administrationsoberfläche referenzieren und diese in Interventionsnachrichten einbetten. Serverseitig werden fachlich folgende Arten von Variablen unterschieden:

Systemvariablen (*System Variables*) Diese Variablen werden automatisch vom System angelegt und verwaltet. Es kann nur ein lesender Zugriff erfolgen. Weiter können die Variablen nicht überschrieben werden.
Beispielvariablen: `$systemYear`, `$systemMonth`

Interventionsteilnehmervariablen (*Participant Variables*) Diese Variablen werden während einer Intervention definiert und vom System verwaltet. Sie enthalten interventionsteilnehmerbezogene Informationen. Es existieren sowohl schreibgeschützte als auch schreibbare Variablen. Der Autor bzw. Administrator einer Intervention kann Interventionsteilnehmervariablen beim Erstellen von Interventionsnachrichten und Dialogen in der Administrationsoberfläche referenzieren. Weiter können mögliche Rückgabewerte einer Interventionsnachrichtenantwort diesen Variablen zugewiesen werden.
Beispielvariablen: `$participantName`, `$participantLastLoginDate`

Interventionsvariablen (*Intervention Variables*) Interventionsvariablen werden manuell vom Autor bzw. Administrator über die Administrationsoberfläche angelegt. Diese können einen Standardwert besitzen sowie folgende Zugriffslevel annehmen (kleinste Level → höchste Restriktion, größte Level → niedrigste Restriktion):

- **Intern** *(Internal → Level 1)*
  Das *interne* Level ist die restriktivste Zugriffsstufe. Variablen können nur über die Administrationsoberfläche geschrieben werden. Ein lesender Zugriff erfolgt z. B. in Interventionsnachrichten.

- **Verwaltbar von Systemdiensten** *(Manageable by service → Level 2)*
  Ist eine Variable von Systemdiensten verwaltbar, können diese Werte der Variable schreiben und lesen. Weiter besitzt die Variable alle Rechte und Funktionen der Zugriffsstufe 1.

- **Extern lesbar** *(Externally readable → Level 3)*
  Ist die Variable extern lesbar, kann ein lesender Zugriff z. B. über die REST-Schnittstelle des Servers erfolgen. Weiter besitzt die Variable alle Rechte und Funktionen der Zugriffstufe 2.

- **Extern les- und schreibbar** *(Externally read and writable → Level 4)*
  Ein lesender wie auch schreibender Zugriff erfolgt z. B. über die REST-Schnittstelle des Servers. Weiter besitzt die Variable alle Rechte und Funktionen der Zugriffstufe 3.

Wird einer Variable während einer Intervention ein Wert zugewiesen und geschieht dies im Kontext eines Interventionsteilnehmers (z. B. bei Beantwortung einer Frage im Interventionsdialog über die MobileCoach-App), wird für genau diesen Teilnehmer eine eigene Version der Variable mit dem entsprechenden Wert angelegt. Jeder Interventionsteilnehmer besitzt somit eine eigene Versionierung von Interventionsvariablen.

3.5.5 Persistenz

Persistente Objekte werden innerhalb der `model`-Komponente verwaltet und durch die Klasse `ModelObject` abgebildet. Entitäten sind nach dem *Active-Record*-Entwurfsmuster definiert (Fowler & Rice, 2011, S. 160–162; Fowler, 2019a). Dies bedeutet, dass Entitäten neben der Datenhaltung durch Attribute auch die Zugriffslogik für Schnittstellen zur Datenbank halten und kapseln. Die Zugriffslogik wird von der `ModelObject`-Klasse abgeleitet, die zusammen mit dem *Active Record* das *Layer-Supertype*-Entwurfsmuster abbildet (Fowler & Rice, 2011, S. 475; Fowler, 2019c). Dabei agiert die `ModelObject`-Klasse als *Supertype* und stellt die Schnittstellen mit Hilfe von CRUD-Methoden bereit. Diese können in jeder Entität aufgerufen werden.

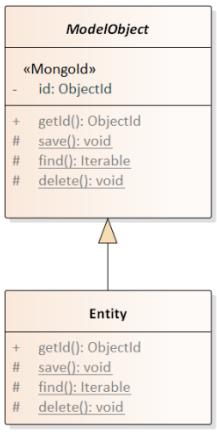


Abbildung 3.5 Basisklasse ModelObject für die MobileCoach-Server-Persistenz.

Wird die Objektinstanz einer Entität verändert und persistiert, werden diese Änderungen ebenfalls auf Datenbankseite angewandt.
Eine Entität repräsentiert einen Datensatz der Datenbank.

3.5.5.1 Datenbank

Als Datenbank verwendet der MobileCoach-Server eine MongoDB-Instanz. MongoDB ist eine dokumentenorientierte NoSQL-Datenbank. Diese besteht nicht aus Tabellen und Tabellenzeilen, wie z. B. relationale Datenbanken, sondern aus Collections und Dokumenten (Trelle, 2014). Eine Collection repräsentiert dabei eine Tabelle, die n Dokumente enthalten kann. Jedes Dokument repräsentiert einen Datensatz, ähnlich der Zeile einer relationalen Datenbanktabelle. Im Gegensatz dazu besitzen Dokumente jedoch kein einheitliches Schema, sodass Redundanzen oder nicht atomare Attribute als gewünschtes Verhalten auftreten können. Ausgenommen ist ein eindeutiges `_id`-Feld, das als Primärschlüssel in jedes Dokument geschrieben wird. Dokumente werden als einfache Schlüssel-Wert-Paare behandelt, die in einem JSON-Format dargestellt und abgerufen werden. Ein Schlüssel besteht aus einer Zeichenkette, die einen eindeutigen Namen darstellt (z. B. `SchlüsselA`). Ein Wert stellt einen Datentyp dar, der ebenfalls als Zeichenkette angegeben und einem Schlüssel zugeordnet wird (z. B. `SchlüsselA: WertA`). Jedes Dokument besteht aus n Schlüssel-Wert-Paaren, die alle spezifizierten Datentypen (z. B. Boolean → `true`, `false` oder Array → `[ElementA, ElementB]`) einer JSON-Datenstruktur enthalten können. Dadurch ist die Erstellung von verschachtelten Dokumenten, bestehend aus Arrays und weiteren Subdokumenten, möglich. Intern wird ein MongoDB-Dokument im sogenannten BSON-Format (Binary JSON), einer effizienteren JSON-Datenstruktur, abgebildet.
Für Datenbankabfragen existiert keine domänenbezogene Abfragesprache, wie z. B. für relationale Datenbanken SQL. MongoDB bietet drei Abfragemöglichkeiten: *Query-by-Example*, *Aggregation-Framework* und *MapReduce* (Trelle, 2015). Im MobileCoach-Server werden Abfragen nach dem *Query-by-Example*-Prinzip ausgeführt. Dabei sind Suchkriterien als Dokument beschrieben, die die abfragende Collection durchsuchen. In diesen können logische

sowie Vergleichsoperatoren genutzt werden. Eine Abfrage gibt immer einen Cursor zurück, der als Iterator dient. Dieser gibt die einzelnen Elemente einer Abfrage zurück.

3.5.5.2 Objekt-Dokumenten-Abbildung

Um die Abbildung von Entitäten in der Datenbank zu realisieren, übernimmt das ODM[3]-Framework (object-document mapping → Objekt-Dokumenten-Abbildung) Jongo diese Aufgabe (Guérout, 2016).
Beim Persistieren werden die Attribute einer Entität in das JSON-Format als `String` serialisiert. Der JSON-`String` repräsentiert ein Dokument der MongoDB. Existiert keine Collection, in die das Dokument geschrieben werden kann, wird diese mit dem Klassennamen der Entität angelegt. Anschließend wird das Dokument in die MongoDB persistiert.
Damit keine Dokumente redundant in die Datenbank geschrieben werden, wird das *Identity-Field*-Entwurfsmuster umgesetzt (Fowler & Rice, 2011, S. 216–220; Fowler, 2019b). Dafür besitzt jede Entität, abgeleitet von der `ModelObject`-Klasse, ein ID-Attribut vom Typ `ObjectId`. Das Attribut ist als `@MongoId` annotiert. Diesem wird bei der Persistenz automatisch ein eindeutiger Wert zugewiesen.
Weiter übernimmt Jongo das Laden von Entitäten. Zurückgegebene Dokumente einer Datenbankabfrage werden anhand des Namens ihrer Collection der gleichnamigen Entität zugeordnet und deserialisiert. Dafür wird eine Instanz der zugeordneten Entität erzeugt. Die Werte der JSON-Felder des Dokuments werden den Attributen der Instanz zugeordnet und geschrieben.

3.5.6 Systemdienste

Systemdienste werden innerhalb der Komponente `services` verwaltet und bilden die Geschäftslogik des MobileCoach-Servers ab. Es existieren Administrationsdienste, die die *Fassade*[4] zwischen `ui-` und `model`-Komponente bilden und die Kommunikation zwischen diesen ermöglichen. Laufzeitdienste beinhalten die Logik zur Ausführung einer Intervention.

---

[3] ODM ist gleichzusetzen mit dem von relationalen Datenbanken bekannten ORM (object-relational mapping → Objektrelationale Abbildung) (Bhusal (2015)).

[4] Das Fassaden Entwurfsmuster entkoppelt eine Komponente, indem es als Schnittstelle Funktionalitäten kapselt und aus dieser delegiert (Eilebrecht und Starke (2019, S. 87)).

Dienste werden nach dem *Singelton*[5]-Entwurfsmuster realisiert. Für weitere Abhängigkeiten zu anderen Diensten wird das *Dependency-Injection*[6]-Entwurfsmuster eingesetzt. Wird ein neuer Dienst implementiert, sollte dieser folgende Struktur besitzen:

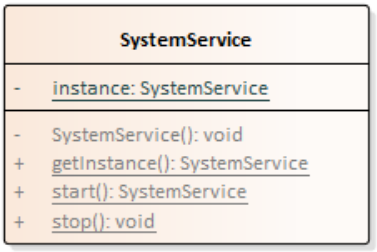


Abbildung 3.6 Struktur einer SystemService-Klasse

Die statische Methode `start` erzeugt die neue Instanz der Dienstklasse und gibt diese zurück. Der Methode können weitere Abhängigkeiten übergeben bzw. injiziert werden. In der `stop`-Methode sind Funktionalitäten zum regelkonformen Beenden des Dienstes implementiert.
Alle Dienste des MobileCoach-Servers werden in der `mc`-Komponente initialisiert und komponiert. Die Klasse `MC` übernimmt die Aufgabe des *Composition Root*[7]. Ein neuer Dienst wird dieser Klasse in der Methode `contextInitialized` hinzugefügt. Dabei ist zu beachten, dass die Reihenfolge der Initialisierung von Systemdiensten von ihren Abhängigkeiten untereinander abhängt z. B.: *Dienst A* ist von *Dienst B* abhängig, *B* hat keine Abhängigkeiten = *Dienst B* wird vor *A* instanziiert/initialisiert. *Zyklische Abhängigkeiten (A ↔ B)* müssen vermieden werden.
Die `stop`-Methode des Dienstes wird der Methode `contextDestroyed` in der `MC`-Klasse hinzugefügt. Wird der MobileCoach-Server angehalten, werden alle Dienste regelkonform beendet.

3.5.7 Oberfläche (UI)

Zur Anzeige der Administrationsoberfläche benutzt der MobileCoach-Server das Webframework Vaadin. Dieses besteht aus einem serverseitigem Framework, das die Oberflächenentwicklung in Java, ähnlich einer Desktop Applikation, ermöglicht. Die clientseitige Ausführung

[5] Das Singelton Entwurfsmuster übernimmt die Erstellung einer Klasseninstanz. Dabei stellt es eine statische Instanzmethode zur Verfügung, die die einmalige Instanziierung übernimmt. Bei nochmaligem Aufruf wird die schon erstellte Instanz zurückgegeben und keine neue instanziiert (Eilebrecht und Starke (2019, S. 38)).

[6] Bei der Dependency Injection wird eine Instanz der anderen bei der Instanziierung übergeben. So kann die neu erstellte Instanz die Funktionen der anderen nutzen, ohne diese zu erzeugen bzw. wissen über die konkrete Klasse zu besitzen. Es entsteht eine lose Kopplung zwischen beiden Instanzen (Seemann (2012)).

[7] Das Compositon Root Entwurfsmuster dient als Startpunkt bei der Initialisierung einer Anwendung und komponiert die Abhängigkeiten von Klassen über Dependency Injection (Seemann (2012, S. 76)).

der UI bzw. die Kommunikation zum Server für die Übertragung von u. a. Oberflächenevents (z. B. das Klicken auf einen Button) wird durch JavaScript implementiert (Grönroos, 2016, S. 1–3).
Der MobileCoach-Server realisiert die Administrationsoberfläche innerhalb der `ui`-Komponente. Diese bietet Schnittstellen in Form von UI-Komponentenklassen, die die Standardkomponenten des Vaadin-Frameworks abstrahieren. Die systemspezifischen UI-Komponentenklassen setzen sich aus Layout-, Basic-, Field- sowie Selection-Komponenten des Vaadin-Frameworks zusammen.
Soll z. B. ein neuer Tab in der Administrationsoberfläche erstellt werden, wird eine neue `TabComponent`-Klasse erzeugt, die die Oberflächenbeschreibung für einen Tab definiert. Diese abstrahiert von der vom MobileCoach-Server bereitgestellten `AbstractCustomComponent`-Klasse, die die Basis jeder Oberflächenkomponente darstellt. Die Oberflächenbeschreibung für z. B. eine tabellarische Darstellung und Funktionalitäten zur Verwaltung einer Entität wird in einer `EditComponent`-Klasse erstellt. Diese wird in der `TabComponent`-Klasse instanziiert und delegiert.
Für die spezifische Oberflächenlogik wird eine `TabComponentWithController`-Klasse erstellt, die von der zuvor erstellten `TabComponent`-Klasse ableitet, um auf die Oberflächenkomponenten zuzugreifen. Die Controller-Klasse übernimmt die Delegation von CRUD-Funktionalitäten zum Verwalten von Entitäten über den `InterventionAdministrationManagerService`. Soll eine Entität in einer Oberfläche, z. B. als Tabelle, abgebildet werden, muss für diese ein entsprechendes `UIModelObject` in der Subkomponente `model/ui` definiert sein.

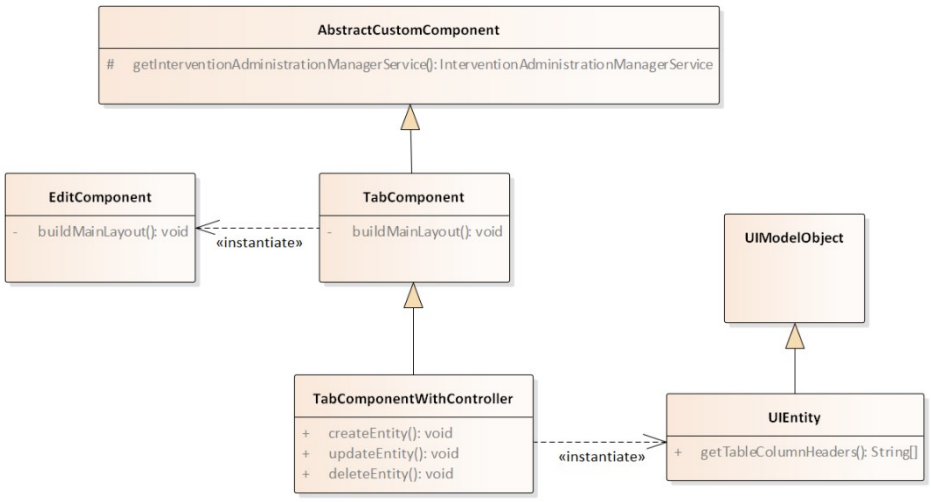


Abbildung 3.7 Erweiterung der Administrationsoberfläche

Um die erstellte `TabComponent`-Klasse in der Administrationsoberfläche anzuzeigen, wird diese in der `InterventionEditingContainerComponentWithController`-Klasse registriert.

4 Anforderungsanalyse

4.1 GET.ON GesundheitsTraining.Online-Plattform

4.1.1 Ist-Zustand

Studienteilnehmer der GET.ON-Plattform durchlaufen Trainingsmodule ausschließlich webbasiert.
Eine Begleitung durch weitere externe Software, z. B. Mobile Apps wie die MobileCoach-App, findet nicht statt. Benötigt ein Teilnehmer Unterstützung oder sinkt dessen Trainingsadhärenz, kann dies nur durch den Betreuer der Studie festgestellt werden. Dafür erfolgt eine manuelle Durchsicht der Trainingsstatistiken. Anhand von Zeitstempeln des letzten Teilnehmerlogins wird festgestellt, ob Trainingseinheiten nicht oder nicht komplett durchlaufen wurden. Die erforderliche Kontaktaufnahme zum Teilnehmer erfolgt in jedem Fall manuell durch den Studienbetreuer.

4.1.2 Soll-Zustand

Die MobileCoach-App soll Studienteilnehmer beim Durchlaufen der Trainingsmodule unterstützen. Wird eine Kontaktaufnahme zum Teilnehmer erforderlich, erfolgt diese automatisiert über die App.
Mögliche Szenarien wären: *1)* Eine Interaktion in Form einer Benachrichtigung in einer Stresssituation, wobei der Nutzer an bestimmte Kraftquellen erinnert wird, die er zuvor innerhalb des Online-Trainings festgelegt hat und die ihm helfen, sich zu erholen und ein vorhandenes Stressniveau abzubauen*; 2)* Eine Interaktion in Form einer Benachrichtigung, die den Nutzer daran erinnert, bei längerer Abwesenheit (z. B. 7 Tage) sein Training durch Unterstützung der App fortzuführen.
Damit eine Kommunikation zur MobileCoach-App stattfinden kann, bietet diese die Möglichkeit, sich mit der GET.ON-Plattform zu verknüpfen.
Innerhalb der App soll eine Webansicht gezeigt werden, die eine Verknüpfung mit der GET.ON-Plattform zulässt. Dafür wird, ähnlich einer Login-Ansicht, der Benutzername und das Passwort für die GET.ON-Plattform abgefragt. Bei einer erfolgreichen Authentifizierung ist die App mit dem GET.ON-Profil des Studienteilnehmers verbunden. Die Verbindung zur jeweiligen App wird über die Interventionsteilnehmer-ID, die über die App vom MobileCoach-Server bezogen wird, hergestellt.
Daraus ergeben sich folgende Aktivitäten:

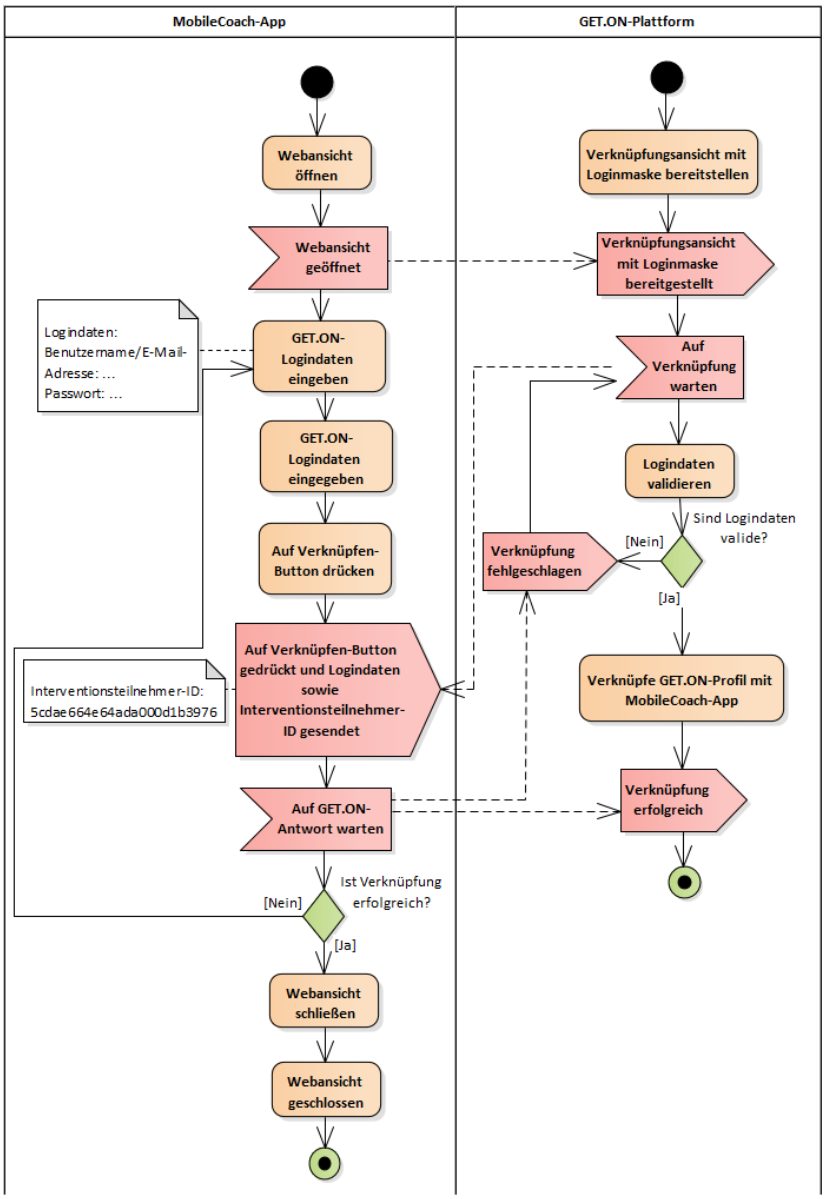


Abbildung 4.1 Aktivitätsdiagramm – Verknüpfung der MobileCoach-App mit der GET.ON-Plattform

4.2 MobileCoach-Plattform

4.2.1 Ist-Zustand

Für externe Systeme, wie z. B. die GET.ON-Plattform, existiert keine Integrationsmöglichkeit in den MobileCoach-Server. Interventionsnachrichten werden nur zwischen dem Mobile-Coach-Server und einem MobileCoach-Client ausgetauscht (z. B. über die MobileCoach-App).

Die Kommunikation zwischen Server und Client findet über den Deepstream-Server (Kapitel 3.5.2) statt. Die REST-API wird benutzt, um z. B. Werte von Interventionsvariablen zu setzen, und stellt eine weitere Schnittstelle des MobileCoach-Servers dar.
Die Schnittstellen erlauben jedoch nur die Kommunikation mit der MobileCoach-App. Fremde externe Systeme können keine Verbindung zum MobileCoach-Server herstellen.

4.2.2 Soll-Zustand

Der MobileCoach-Server bietet Funktionalitäten für die Integration externer Systeme (z. B. GET.ON-Plattform). Dabei besteht die Möglichkeit, externe Systeme über die Administrationsoberfläche des MobileCoach-Servers zu erstellen bzw. abzubilden. Während der Erstellung wird ein *Systemname* festgelegt, eine eindeutige *System-ID* und ein eindeutiger *Schlüssel* (*Token*) für das externe System erzeugt. Mit Hilfe dieser Daten kann sich ein externes System am MobileCoach-Server authentifizieren und eine Kommunikation starten. Während der Kommunikation werden Nachrichten vom externen System zum Server übertragen. Diese beinhalten textbasierte Daten in Form von Schlüssel-Wert-Paaren (z. B. `SchlüsselA: WertA`). Um die übertragenen Daten auf MobileCoach-Serverseite flexibel zu nutzen, werden diesen Interventionsvariablen (Kapitel 3.5.4) zugewiesen. Dafür wird in der Administrationsoberfläche eine entsprechende Abbildung eines Schlüssel-Wert-Paares auf eine Interventionsvariable ermöglicht. Die Abbildungen gelten wiederum für ein spezifisches externes System.
Um auf eine empfangene externe Nachricht zu reagieren, soll eine neue Interventionsregel erstellt werden. Wird die Nachricht eines externen Systems empfangen, soll z. B. auf den Systemnamen (ist `Systemname == „GET.ON Plattform“`) des externen Systems, von dem die Nachricht gesendet wurde, geprüft werden können. Ist die Bedingung wahr, kann eine weitere Aktion, z. B. das Senden einer Interventionsnachricht an einen Interventionsteilnehmer, erfolgen.
Daraus ergeben sich folgende Aktivitäten:

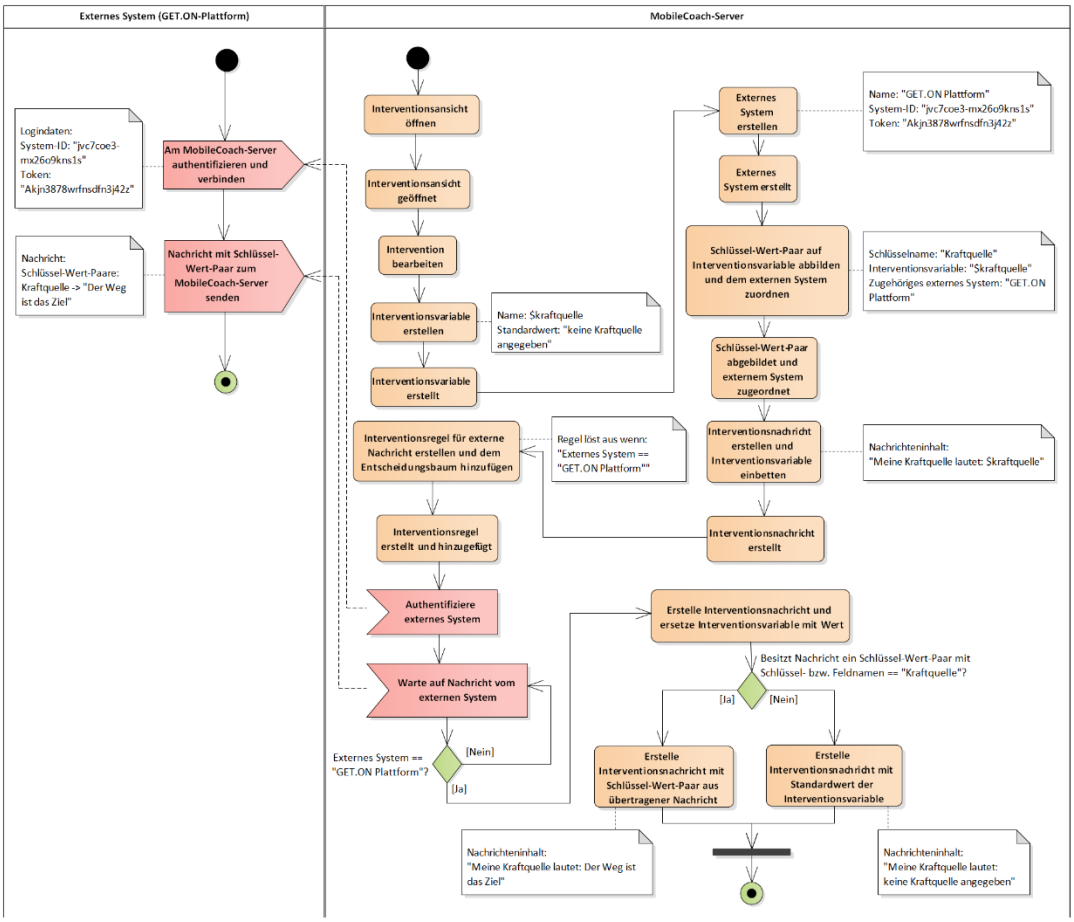


Abbildung 4.2 Aktivitätsdiagramm – Integration externer Systeme in den MobileCoach-Server

4.3 Anwendungsfälle

Aus der zuvor erstellten Ist-/Soll-Analyse ergeben sich vier Anwendungsfälle. Die Anwendungsfälle finden in einem jeweils spezifischen Systemkontext statt:

Anwendungsfall	Systemkontext	Beschreibung
AWF01	MobileCoach-Server	Erstellung der GET.ON-Plattform als externes System.
AWF02	MobileCoach-Server	Abbildung von Nachrichtenwerten eines externen Systems auf Interventionsvariablen.
AWF03	MobileCoach-App	Verknüpfung der MobileCoach-App mit der GET.ON-Plattform.
AWF04	GET.ON-Plattform	Senden von Nachrichten aus der GET.ON-Plattform an die MobileCoach-App.

Tabelle 4.1 Anwendungsfälle

4.3.1 AWF01 – Externes System erstellen

Bevor ein externes System mit dem MobileCoach-Server kommunizieren kann, muss dieses dem Server bekannt sein. Ein Zugriff von nicht autorisierten externen Systemen wird damit unterbunden.

Auslöser	**Ein externes System soll über die Administrationsoberfläche des MobileCoach-Servers erstellt werden.**
Vorbedingungen	-
Nachbedingungen	- Das externe System ist persistiert und kann über die Administrationsoberfläche angezeigt werden.
Erfolgsszenario	1. Administrator/Autor einer Intervention ruft webbasierte Administrationsoberfläche des MobileCoach-Servers auf und meldet sich mit Benutzernamen und Passwort an. 2. Administrator/Autor navigiert zur Interventionsübersicht, wählt eine Intervention aus und bearbeitet diese. 3. Administrator/Autor navigiert zur Ansicht „Externe Systeme“. 4. Eine tabellarische Ansicht wird geöffnet, in der externe Systeme angezeigt und verwaltet werden. 5. Administrator/Autor wählt die Funktion „Neu“ und gibt einen Namen für das externe System an. 6. Administrator/Autor bestätigt mit „OK“. 7. Das System prüft den angegeben Namen des externen Systems auf Eindeutigkeit. 8. Das System legt einen Datensatz mit folgenden Daten an: Externer Systemname, Externe System-ID, Token, Status (aktiv, inaktiv). Das externe System wird der momentan in Bearbeitung befindlichen Intervention zugeordnet.
Erweiterung	-
Fehlerfälle	7.a Falls der externe Systemname nicht eindeutig ist, wird ein Fehler ausgegeben („Der gewählte Systemname wird bereits benutzt. Wähle einen anderen Namen.“). Der Datensatz wird nicht angelegt.

Tabelle 4.2 Anwendungsfall01 – Externes System erstellen

4.3.2 AWF02 – Schlüssel-Wert-Paare auf Interventionsvariablen abbilden

Um übertragene Daten zwischen einem externen System und dem MobileCoach-Server zu verarbeiten, müssen diese innerhalb einer Intervention abgebildet werden. Eine Abbildung erfolgt auf Interventionsvariablen (Kapitel 3.5.4), da diese in den Interventionsablauf eingebunden werden und sich in Interventionsnachrichten sowie Interventionsdialogen referenzieren lassen.

Auslöser	**Über die Administrationsoberfläche sollen Abbildungen von Schlüssel-Wert-Paaren auf Interventionsvariablen für ein externes System angelegt werden.**
Vorbedingungen	- Administrator/Autor hat bereits ein externes System über die Administrationsoberfläche im MobileCoach-Server erstellt.
Nachbedingungen	- Eine Abbildung von Schlüssel-Wert-Paaren auf Interventionsvariablen ist einem bestehenden externen System zugeordnet.
Erfolgsszenario	1. Administrator/Autor einer Intervention ruft webbasierte Administrationsoberfläche des MobileCoach-Servers auf und meldet sich mit Benutzernamen und Passwort an. 2. Administrator/Autor navigiert zur Interventionsübersicht, wählt eine Intervention aus und bearbeitet diese. 3. Administrator/Autor navigiert zur Ansicht „Externe Systeme“. 4. Eine tabellarische Ansicht wird geöffnet, in der externe Systeme angezeigt und verwaltet werden. 5. Administrator/Autor wählt den Datensatz eines bestehenden externen Systems aus. 6. Administrator/Autor wählt die Funktion „Schlüssel-Wert-Paare auf Variablen abbilden“. 7. Eine modale tabellarische Ansicht wird geöffnet, in der Abbildungen von Schlüssel-Wert-Paaren auf Variablen angezeigt und verwaltet werden. 8. Administrator/Autor wählt die Funktion „Neu“, gibt den Schlüsselnamen eines Schlüssel-Wert-Paares an und wählt aus einer Dropdown-Liste eine Interventionsvariable aus. 9. Das System prüft den angegeben Schlüsselnamen des Schlüssel-Wert Paares auf Eindeutigkeit und ob eine Interventionsvariable ausgewählt wurde.

	10. Das System legt einen Datensatz mit folgenden Daten an: Schlüsselname des Schlüssel-Wert-Paares, Interventionsvariablenname.
Erweiterung	5.a Falls kein externes System existiert → Ausführung AWF01.
Fehlerfälle	8.a Es werden nur Interventionsvariablen mit dem Zugriffslevel 2 bis 4 angezeigt (Kapitel 3.5.4). Variablen mit Zugriffslevel 1 werden nicht angezeigt, da diese keine Zugriffsberechtigung besitzen. 9.a Falls der Schlüsselname nicht eindeutig ist, wird ein Fehler ausgegeben („Der gewählte Schlüsselname wird bereits benutzt. Wähle einen anderen Namen."). Der Datensatz wird nicht angelegt. 9.b Falls keine Interventionsvariable ausgewählt ist, wird ein Fehler ausgegeben („Die Variable wurde nicht gefunden."). Der Datensatz wird nicht angelegt. 10.a Falls sich das Zugriffslevel der Interventionsvariable auf Stufe 1 ändert, wird ein Fehler („Auf die Variable kann nicht zugegriffen werden.") bei der Anzeige des Datensatzes ausgegeben. 10.b Falls die referenzierte Interventionsvariable gelöscht ist, wird ein Fehler („Unbekannte Variable.") bei der Anzeige des Datensatzes ausgegeben.

Tabelle 4.3 Anwendungsfall02 – Schlüssel-Wert-Paare auf Interventionsvariablen abbilden

4.3.3 AWF03 – MobileCoach-App mit GET.ON-Plattform verknüpfen

Um Interventionsnachrichten ausgehend von der GET.ON-Plattform an Studienteilnehmer über die MobileCoach-App zu senden, muss eine Verknüpfung der App zur GET.ON-Plattform bestehen. Dabei wird die Interventionsteilnehmer-ID an die GET.ON-Plattform übertragen. Über diese lässt sich der Interventionsteilnehmer eindeutig identifizieren.

Auslöser	**Ein Studienteilnehmer will die MobileCoach-App mit seinem GET.ON-Profil verknüpfen.**
Vorbedingungen	- Der Studienteilnehmer besitzt ein Profil auf der GET.ON-Plattform. - Der Studienteilnehmer hat die MobileCoach-App auf seinem Smartphone installiert.
Nachbedingungen	- Die MobileCoach-App des Studienteilnehmers ist mit seinem GET.ON-Profil verknüpft.
Erfolgsszenario	1. Der Studienteilnehmer öffnet die MobileCoach-App auf seinem Smartphone.

	2. Der Studienteilnehmer navigiert zur Interventionschatansicht. 3. Der Studienteilnehmer wählt die Funktion „Mit GET.ON-Plattform verbinden“. 4. Eine Webansicht mit einem Formular zum Verknüpfen der GET.ON-Plattform öffnet sich. 5. Der Studienteilnehmer befüllt das Formular mit seinen GET.ON-Logindaten: Nutzername/E-Mail-Adresse, Passwort. 6. Das System prüft, ob Nutzername/E-Mail-Adresse und Passwort angegeben sind. 7. Der Studienteilnehmer wählt die Funktion „Mit GET.ON verbinden“. 8. Das System prüft, ob die Logindaten gültig sind bzw. ob ein GET.ON-Profil mit diesen Daten existiert und eine Verknüpfung mit der GET.ON-Plattform möglich ist. 9. Die Webansicht schließt sich und die MobileCoach-App des Studienteilnehmers ist mit seinem GET.ON-Profil verknüpft.
Erweiterung	-
Fehlerfälle	6.a Falls kein Nutzername/E-Mail-Adresse angegeben ist, wird ein Fehler ausgegeben („Bitte einen Benutzernamen oder E-Mail-Adresse angeben.“) und zur Korrektur aufgefordert. 6.b Falls kein Passwort angegeben ist, wird ein Fehler ausgegeben („Bitte ein Passwort angeben.“) und zur Korrektur aufgefordert. 8.a Sind die Logindaten ungültig, wird ein Anmeldungsfehler ausgegeben („Benutzername/E-Mail-Adresse oder Passwort ist ungültig. Probiere es nochmal.“) und zur nochmaligen Eingabe aufgefordert. 8.b Ist die MobileCoach-App bereits mit einem GET.ON-Profil verbunden, wird die Meldung („Du bist bereits mit der GET.ON-Plattform verbunden. Es sind keine weiteren Schritte erforderlich.“) ausgegeben. 8.c Tritt ein unbekannter Fehler auf, wird eine spezifische Fehlermeldung („*[Spezifische Fehlermeldung]*. Bitte wende dich an den Administrator.“) ausgegeben.

Tabelle 4.4 Anwendungsfall03 – MobileCoach-App mit GET.ON-Plattform verknüpfen

4.3.4 AWF04 – Kommunikation zwischen GET.ON-Plattform und MobileCoach

Entwickler externer Systeme sollen die Möglichkeit haben mit dem MobileCoach-Server zu kommunizieren. Über eine Schnittstelle werden Nachrichten in Form einer Schlüssel-Wert Datenstruktur übertragen. Der MobileCoach-Server kann auf eine vom externen System gesendete bzw. empfangene Nachricht reagieren und eine Interventionsnachricht an die MobileCoach-App eines Studienteilnehmers senden.

Auslöser	**Eine Nachricht soll von der GET.ON-Plattform an den MobileCoach-Server gesendet werden, worauf dieser eine Interventionsnachricht an die MobileCoach-App eines Studienteilnehmers sendet.**
Vorbedingungen	- Der Administrator/Autor hat bereits ein externes System auf MobileCoach-Serverseite erstellt. - Der Administrator/Autor hat eine Abbildung von Schlüssel-Wert-Paaren auf Interventionsvariablen, die die Nachrichteninformationen transportieren, angelegt. - Der Studienteilnehmer ist über die MobileCoach-App mit seinem GET.ON-Profil verbunden.
Nachbedingungen	- Die GET.ON-Plattform sendet eine Nachricht, die als Interventionsnachricht in der MobileCoach-App des Studienteilnehmers (Nachrichtenempfänger) angezeigt wird.
Erfolgsszenario	1. Die GET.ON-Plattform ruft eine Funktion für den Login am MobileCoach-Server mit folgenden Daten auf: Externe System-ID, Token. 2. Der MobileCoach-Server prüft, ob die Logindaten gültig sind und sich einem bestehenden externen System zuordnen lassen. 3. Die GET.ON-Plattform erstellt eine Nachricht mit Schlüssel-Wert-Paaren, die der erstellten Abbildung im MobileCoach-Server entspricht. 4. Die GET.ON-Plattform ruft eine Funktion für die Nachrichtenübertragung mit folgenden Daten auf: Externe System-ID, Interventionsteilnehmer-ID-Liste (Empfängerliste), verschachtelte Datenstruktur mit Schlüssel-Wert-Paaren. 5. Der MobileCoach-Server verarbeitet die übertragene Nachricht und erstellt eine entsprechend formatierte Interventionsnachricht mit den übertragenen Daten der Schlüssel-Wert-Paar-Datenstruktur und versendet diese.

	6. Die GET.ON-Plattform erhält eine positive Rückmeldung der Nachrichtenverarbeitung. 7. Der Studienteilnehmer empfängt eine Interventionsnachricht mit den übertragenen Daten von der GET.ON-Plattform, die in seiner MobileCoach-App angezeigt wird.
Erweiterung	2.a Falls kein externes System existiert → Ausführung AWF01. 3.a Falls keine Abbildung von Schlüssel-Wert-Paaren auf Interventionsvariablen existiert → Ausführung AWF02. 7.a Falls keine Verknüpfung mit der GET.ON-Plattform besteht → Ausführung AWF03.
Fehlerfälle	2.a Sind die Logindaten ungültig bzw. lässt sich kein externes System den angegebenen Logindaten zuordnen, wird ein Fehler zurückgegeben. 5.a Wird keine externe System-ID übertragen, wird ein Fehler zurückgegeben. 5.b Falls die übertragene externe System-ID ungültig ist bzw. sich dieser kein externes System zuordnen lässt, wird eine Warnung auf MobileCoach-Serverseite protokolliert und keine Nachricht versendet. 5.c Ist eine Interventionsteilnehmer-ID in der Empfängerliste ungültig oder kann der Intervention, zu der das externe System zugehörig ist, nicht zugeordnet werden, wird diese übersprungen, eine Warnung auf MobileCoach-Serverseite protokolliert und keine Nachricht versendet.

Tabelle 4.5 Anwendungsfall04 – Kommunikation zwischen GET.ON-Plattform und MobileCoach

4.4 Anforderungen

4.4.1 Funktionale Anforderungen

Aus den Anwendungsfällen resultieren folgende funktionale Anforderungen:

Anforderung	Beschreibung	Anwendungsfallreferenz
FA01	**Externes System erstellen** Ein externes System wird im MobileCoach-Server erstellt und persistiert.	AWF01
FA02	**Externen Systemnamen umbenennen** Der beschreibende und eindeutige Name eines externen Systems kann geändert werden.	AWF01
FA03	**Token für externes System neu erzeugen** Falls das Token eines externen Systems kompromittiert wurde bzw. unsicher ist, kann ein neuer erzeugt werden.	AWF01
FA04	**Externes System aktivieren/deaktivieren** Soll der Zugriff eines externen Systems (z. B. GET-ON-Plattform) auf den MobileCoach-Server vorübergehend unterbunden werden, wird das im Server angelegte externe System inaktiv geschaltet.	AWF01
FA05	**Externes System löschen** Es besteht die Möglichkeit, ein externes System permanent von der MobileCoach-Serverseite zu entfernen.	AWF01
FA06	**Abbildung von Schlüssel-Wert-Paaren auf Interventionsvariable erzeugen** Bei der Übertragung von Werten (Daten) zwischen einem externen System und dem MobileCoach-Server ist ein Wert einem eindeutigen Schlüssel zugeordnet (z. B. `SchlüsselA: WertA`). Dieser wiederum wird auf Serverseite einer Interventionsvariable zugeordnet (z. B. `VariableA = WertA`), was eine flexible Nutzung des Wertes innerhalb der Intervention ermöglicht.	AWF01, AWF02
FA07	**Schlüsselname der Abbildung umbenennen** Der Schlüsselname eines Schlüssel-Wert-Paares muss auf MobileCoach-Serverseite anpassbar sein, falls sich dieser auf Seite des externen Systems ändert oder ein anderer Name gewählt werden soll.	AWF01, AWF02

FA08	**Interventionsvariable der Abbildung ändern** Falls eine andere Interventionsvariable für einen übertragenen Wert genutzt werden soll, muss diese innerhalb der Abbildung änderbar sein.	AWF01, AWF02
FA09	**Abbildung von Schlüssel-Wert-Paaren auf Interventionsvariable löschen** Es besteht die Möglichkeit, eine Abbildung permanent von der MobileCoach-Serverseite zu entfernen.	AWF01, AWF02
FA10	**Oberfläche (Formular) für Verknüpfung der MobileCoach-App mit GET.ON-Plattform erstellen** Ein Studienteilnehmer der GET.ON-Plattform soll die MobileCoach-App mit seinem GET.ON-Profil verbinden können.	AWF03
FA11	**Funktion zur Authentifizierung über die MobileCoach-App an der GET.ON-Plattform** Damit eine Verbindung von der MobileCoach-App zur GET.ON-Plattform erfolgen kann, wird eine Funktion für die Authentifizierung auf GET.ON-Serverseite benötigt.	AWF03
FA12	**Funktion zur Authentifizierung externer Systeme am MobileCoach-Server** Externe Systeme müssen sich, um Daten an den MobileCoach-Server zu übertragen, über diesen authentifizieren können.	AWF04
FA13	**Funktion für die Kommunikation und den Austausch von Daten zwischen einem externen System und dem MobileCoach-Server** Daten werden über diese Funktion als Schlüssel-Wert-Paar-Datenstruktur ausgetauscht.	AWF04
FA14	**Interventionsregel für den Empfang von Nachrichten eines externen Systems in den Entscheidungsbaum einer Intervention integrieren** Der MobileCoach-Server muss auf eine übertragene Nachricht eines externen Systems reagieren können. Dies ist nötig, um z. B. Interventionsnachrichten mit den übertragenen Daten der externen Nachricht an Interventionsteilnehmer zu versenden. Dafür wird ein Event im Entscheidungsbaum der Intervention ausgelöst.	AWF04

Tabelle 4.6 Funktionale Anforderungen

4.4.2 Nicht funktionale Anforderungen

Anforderung	Beschreibung	Anwendungsfallreferenz
NFA01	**Performance der Nachrichtenverarbeitung** Die Kommunikationsschnittstelle zwischen GET.ON-Plattform und MobileCoach-Server kann 20 Nachrichten parallel/sequentiell verarbeiten.	AWF04

Tabelle 4.7 Nicht funktionale Anforderungen

5 Entwurf

5.1 MobileCoach-Server

5.1.1 Anzeige externer Systeme in der Administrationsoberfläche

Für die Anzeige externer Systeme in der Administrationsoberfläche (AWF01) wird eine eigene Oberflächensubkomponente `external_systems` erstellt. Diese wird in die `ui`-Komponente `compnents/main_view/interventions` (Kapitel 3.4.2.3) integriert. Die Anzeige ermöglicht eine tabellarische Übersicht aller angelegten externen Systeme. Es werden der *Systemname*, *System-ID*, *Token* und *Status* des externen Systems angezeigt. Ein Mockup der Oberfläche befindet sich im Anhang 2.1. Folgende Funktionalitäten werden über die Oberfläche angeboten:

Externes System erzeugen (*New*, FA01, Anhang – Sequenzdiagramm 1.1) Ein modaler Dialog öffnet sich, der den Namen des zu erstellenden externen Systems abfragt. Der externe Systemname wird in einem Textfeld eingegeben und bestätigt. Existiert bereits ein externes System mit gleichem Namen, wird eine Fehlermeldung ausgegeben und kein externes System erzeugt. Andernfalls wird eine Erfolgsmeldung angezeigt.

Externes System umbenennen (*Rename*, FA02, Anhang – Sequenzdiagramm 1.2) Ein modaler Dialog öffnet sich, der den neuen Namen des bestehenden externen Systems abfragt. Der neue externe Systemname wird in einem Textfeld eingegeben und bestätigt. Existiert

bereits ein externes System mit gleichem Namen, wird eine Fehlermeldung ausgegeben und das externe System nicht umbenannt.

Externes System löschen (*Delete*, FA05, Anhang – Sequenzdiagramm 1.3) Ein modaler Bestätigungsdialog öffnet sich. Wird mit OK bestätigt, wird das externe System aus der Datenbank gelöscht. Alle referenzierten Abbildungen auf Interventionsvariablen werden ebenfalls gelöscht.

Token erneuern (*Renew Token*, FA03, Anhang – Sequenzdiagramm 1.4) Ein modaler Bestätigungsdialog öffnet sich. Wird mit OK bestätigt, wird ein neuer Token erzeugt und gespeichert.

Schlüssel-Wert-Paare auf Interventionsvariablen abbilden (*Field → Variable Mapping*, AWF02) Ein modaler erweiterter Dialog öffnet sich. Dieser besitzt die Funktionalität, Schlüsselwerte auf eine Interventionsvariable festzulegen.

Externes System aktivieren/deaktivieren (*Activate/Deactivate System*, FA04, Anhang – Sequenzdiagramm 1.5) Ein modaler Bestätigungsdialog öffnet sich. Wird mit OK bestätigt, wird das aktuell selektierte externe System aktiviert/deaktiviert. Eine Erfolgsmeldung wird angezeigt.

5.1.1.1 Abbildung der Schlüssel-Wert-Paare auf Interventionsvariablen

Um die Abbildung von Schlüssel-Wert-Paaren auf Interventionsvariablen über die Oberfläche zu ermöglichen (AWF02), wird eine eigene tabellarische Ansicht erstellt, die sich für ein ausgewähltes externes System aufrufen lässt und alle erstellten Abbildungen des externen Systems anzeigt. Die Ansicht wird in einem modalen Dialog innerhalb der Anzeige externer Systeme aufgerufen (Mockup 2.2). Für die Zuordnung von einem Schlüsselnamen auf eine Interventionsvariable wird ein eigener Zuordnungsdialog innerhalb der Komponente `components/basics` erstellt (Mockup 2.3). Folgende Funktionalitäten werden angeboten:

Abbildung erzeugen (*New*, FA06, Anhang – Sequenzdiagramm 1.6) Der modale Zuordnungsdialog öffnet sich, der den Schlüsselnamen abfragt und eine Auswahl von Interventionsvariablen zulässt. Tritt ein Fehler auf, wird eine Fehlermeldung ausgegeben und die Abbildung nicht erzeugt.

Abbildung bearbeiten (*Edit*, FA07/FA08, Anhang – Sequenzdiagramm 1.7) Der modale Zuordnungsdialog öffnet sich, der einen neuen Schlüsselnamen abfragt und eine neue Auswahl von Interventionsvariablen zulässt. Tritt ein Fehler auf, wird eine Fehlermeldung ausgegeben und die Abbildung nicht verändert.

Abbildung löschen (*Delete*, FA09, Anhang – Sequenzdiagramm 1.8) Ein modaler Bestätigungsdialog öffnet sich. Wird mit OK bestätigt, wird die Abbildung aus der Datenbank gelöscht.

5.1.2 Datenmodell externer Systeme

Um die Anzeige externer Systeme in der Administrationsoberfläche zu ermöglichen, werden entsprechende Entitätsklassen erstellt. Die Entitäten werden auf Serverseite persistiert und sind somit von der Oberfläche dauerhaft abrufbar. Ein externes System unterteilt sich dabei in die Entitäten `InterventionExternalSystem` und `InterventionExternalSystemFieldVariableMapping`. Letztere Entität stellt die Abbildung von Schlüssel-Wert-Paaren auf Interventionsvariablen dar und ist jeweils einer `InterventionExternalSystem`-Entität zugeordnet ($1:n$). Folgendes fachliches Datenmodell wird erstellt (grüne Entitäten sind bereits im MobileCoach-Server vorhanden):

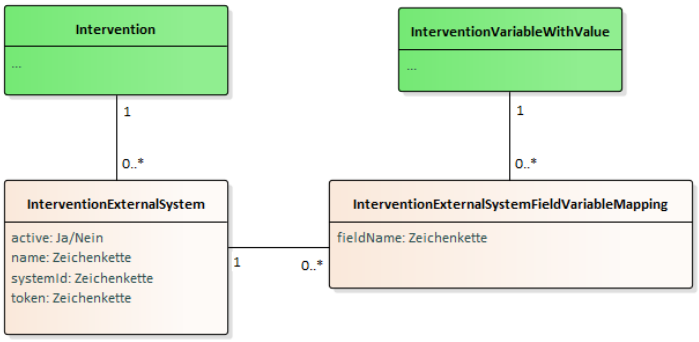


Abbildung 5.1 Fachliches Datenmodell der Subkomponente external_systems

Das Attribut `fieldName` der `InterventionExternalSystemFieldVariableMapping`-Entität stellt den Schlüssel eines Schlüssel-Wert-Paares dar.

Das Token eines neu erstellten externen Systems wird nicht nur in die Datenbank persistiert, sondern auch als Deepstream-Record (Kapitel 3.5.2) angelegt (siehe Sequenzdiagramm 1.1). Der Record-Pfad ergibt sich aus der Zeichenkette „`external-systems/[externe System-ID]`“. Dadurch lässt sich bei einem späteren Authentifizierungsversuch das Token effizienter abrufen, da ein direkter Zugriff auf die Datenbank entfällt.

5.1.3 Erstellung der Authentifizierungsdaten eines externen Systems

Ein externes System wird durch die Angabe der System-ID und des Tokens am MobileCoach-Server authentifiziert. Die System-ID muss dabei für jedes erstellte externe System eindeutig sein. Aus diesem Grund wird eine UID (*Unique-ID*) als System-ID erzeugt. Die Erzeugung kann von einer Funktion des Deepstream-Servers übernommen werden (`getUid()`). Die

Wahrscheinlichkeit der Kollision (mehrfache Generierung der gleichen UID) liegt bei $1\ zu\ 1 * 10^{16}$ und ist somit ein vertretbares Risiko (deepstreamHub GmbH, 2019e).
Das Token fungiert bei der Authentifizierung als Passwort des externen Systems. Ein Faktor bei der Generierung des Tokens kann die Möglichkeit einer Kollision sein. Dadurch, dass sich die Eindeutigkeit eines externen Systems aus System-ID und Token zusammensetzt, wird dieser Faktor vernachlässigt, da nur eine geringe Wahrscheinlichkeit der Kollision besteht.
Ein weitaus bedeutenderer Faktor ist die Token-Sicherheit gegen eine mögliche Kompromittierung. Angreifer können z. B. Brut-Force-Attacken gegen ein gewähltes Token fahren. Beim Brut-Force-Verfahren probiert ein Computer-Programm sämtliche Kombinationen eines möglichen Tokens durch. Die Effizienz dieses Verfahrens hängt mit der Länge des Tokens zusammen (Schröder, 2017). Die Kombinationsmöglichkeiten (KM) ergeben sich aus der Zeichenanzahl (za) und der Passwortlänge (pl): $KM = za^{pl}$. Ein aktueller Prozessor kann durchschnittlich 2.147.483.600 Milliarden Schlüssel pro Sekunde (SPS) durchprobieren (Ikeda, 2013). Die Zeit für einen vollständigen und erfolgreichen Brute-Force-Angriff ergibt sich aus: $BruteForceZeit = \frac{KM}{SPS}$. Ein Token besteht aus einer zufälligen alphanumerischen Zeichenfolge ([0-9, a-z, A-Z]). Daraus ergeben sich: $10 + 26 + 26 = 62\ Zeichen$. Der MobileCoach-Server benutzt bereits Tokens für die Authentifizierung von Interventionsteilnehmern der MobileCoach-App. Diese haben eine Länge von 128 Byte. Das Token eines externen Systems wird mit gleicher Länge generiert. Daraus ergeben sich 62^{128} mögliche Kombinationen. Ein aktueller Prozessor würde somit $\frac{1}{365}\left(\frac{1}{24}\left(\frac{1}{60} * \frac{\frac{62^{128}}{2147483600}}{60}\right)\right)$ Jahre benötigen, um einen erfolgreichen Angriff durchzuführen. Dies sind mehr als doppelt so viele Jahre wie Atome im Universum existieren. Die Chance eines erfolgreichen Angriffs ist damit minimal.

5.1.4 Authentifizierung eines externen Systems

Für die Authentifizierung und Kommunikation über den MobileCoach-Server (FA12) stehen die REST-Schnittstelle und der Deepstream-Server zur Verfügung. Für die Wahl der richtigen Schnittstelle werden die Vor- und Nachteile beider abgewogen:

REST:

- **Vorteile** *1)* Einfache Integration in bestehende Software über HTTP → keine extra Client-Programmbibliothek notwendig.

- **Nachteile** *1)* Generierung eines Authentifizierungstoken für die Schnittstelle nur über ein Deepstream Remote Procedure Call (RPC) möglich → REST-Schnittstelle erfordert die Nutzung von Deepstream*; 2)* Keine Rechtevergabe für implementierte Funktionalitäten möglich → Das Ausführen bestimmter REST-Funktionen durch einen Client kann nicht allgemein eingeschränkt werden.

Deepstream:

- **Vorteile** *1)* Wird bereits für die Nachrichtenkommunikation zwischen MobileCoach-Sever und App benutzt*; 2)* Einfache Implementierung neuer Funktionalität über RPC oder Records*; 3)* Erweiterbare Authentifizierungsschnittstelle in der `DeepstreamRESTServlet`-Klasse vorhanden*; 4)* Rechtevergabe für implementierte Funktionen möglich.

- **Nachteile** *1)* Integration in bestehende Software abhängig von der Existenz eines entsprechenden Deepstream-Clients → Eine Deepstream-Client-Programmbibliothek muss für eine spezifische Programmiersprache existieren.

Die Wahl fällt auf Deepstream. Dies hat folgende Gründe: Die `DeepstreamRESTServlet`-Klasse zur Authentifizierung besteht bereits und kann erweitert werden, eine Rechtevergabe für Funktionen ist möglich. Für die REST-Schnittstelle sind diese Funktionalitäten nicht verfügbar bzw. müssten entwickelt werden.

Bei der Authentifizierung über Deepstream werden Clients anhand spezifischer Rollen unterschieden. Folgende Rollen existieren: *1)* **server:** Für interne Verbindung vom MobileCoach- zum Deepstream-Server*; 2*) **participant:** Verbindungen eines Interventionsteilnehmers über die MobileCoach-App.

Der Deepstream-Server wird um die Rolle `external-system` erweitert.
Dafür wird in den `ImplementationConstants` der `conf`-Komponente die Konstante `external-system` definiert. Diese wird zur Überprüfung auf den Login eines externen Systems genutzt. Soll sich ein externes System mit dem MobileCoach-Server verbinden, müssen der `login` Methode des Deepstream-Clients folgende Daten übergeben werden: *1)* **client-version:** Versionsnummer des genutzten Deepstream-Client*; 2)* **role:** Nutzerrolle des zu authentifizierenden Systems (in diesem Fall `external-system`)*; 3)* **systemId:** Die ID des externen Systems*; 4)* **token:** Das Token des externen Systems.

Der Deepstream-Server kommuniziert bzw. akzeptiert Authentifizierungsanfragen über zwei verschiedene Protokollarten:

Websocket Authentifiziert sich ein Deepstream-Client über das Websocket-Protokoll, können der `login`-Methode alle Authentifizierungsdaten als Schlüssel-Wert-Paare in einem JSON-Dokument übergeben werden:

```
1 {
2   "client-version": 1,
3   "systemId": "jvc7coe3-mx26o9kns1s",
4   "role": "external-system",
5   "token": "L7kQHY2RWmvibLLVDYx6ovd2GLVp9T5KTrV5yMrda7xYww4xYXG9eLe0y9yz4wvTyEERiMfDLue-
GtSRXT1CuZ7B9afxAmByVs3GBHzJ7gXFMrmXcy9a1SGlY94af7n01"
6 }
```

Listing 5.1 JSON-Authentifizierungsdaten für das Websocket-Protokoll

HTTP Authentifiziert sich der Deepstream-Client über das HTTP-Protokoll, akzeptiert die `login`-Methode nur ein Schlüssel-Wert-Paar mit dem Schlüsselnamen `token` (deepstreamHub GmbH, 2019c). Authentifizierungsdaten wie z. B. die externe System-ID können nicht separat definiert werden. Um die Beschränkung zu umgehen, werden die Authentifizierungsdaten, separiert durch ein Semikolon, als `token`-Parameter übergeben:

```
1 {
2   "token": "1;external-system;jvc7coe3-mx26o9kns1s;L7kQHY2RWmvibLL-
VDYx6ovd2GLVp9T5KTrV5yMrda7xYww4xYXG9eLe0y9yz4wvTyEERiMfDLueGtSRXT1CuZ7B9afxA-
mByVs3GBHzJ7gXFMrmXcy9a1SGlY94af7n01"
3 }
```

Listing 5.2 JSON-Authentifizierungsdaten für das HTTP-Protokoll

Auf MobileCoach-Serverseite wird die Zeichenkette aus dem `token`-Parameter in ein JSON-Dokument zurückgewandelt. Dadurch kann die Validierung der Authentifizierungsdaten wie bei einer Anfrage über das Websocket-Protokoll erfolgen. Dies hat den Vorteil, dass keine extra Implementierung für eine Authentifizierung über das HTTP-Protokoll geschrieben werden muss. Der Authentifizierungsprozess durchläuft folgende Schritte (ein detailliertes Sequenzdiagramm befindet sich im Anhang 1.9):

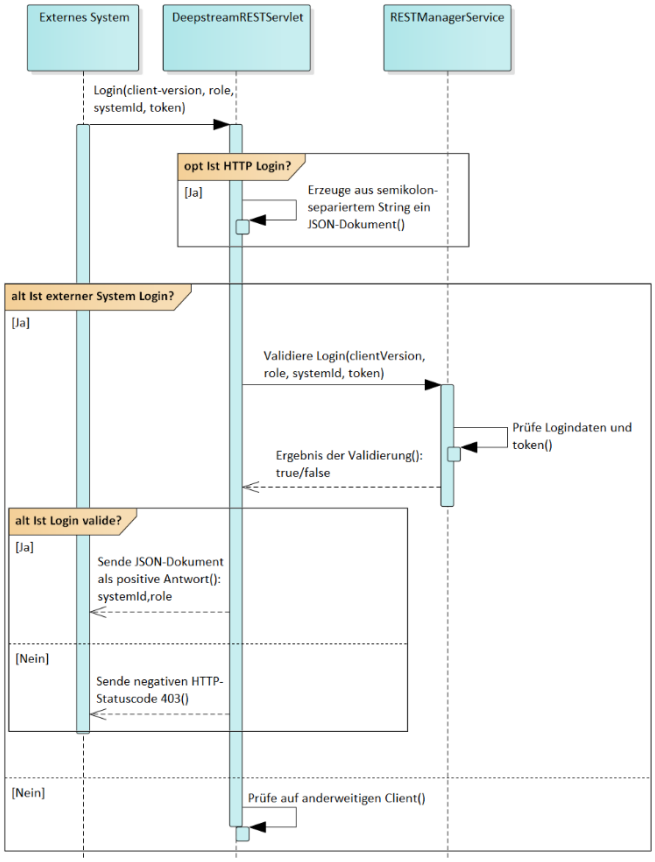


Abbildung 5.2 Sequenzdiagramm (vereinfacht) – Authentifizierung eines externen Systems

Schritt	Beschreibung
1	Ein externes System ruft die `login`-Methode der Deepstream-Client-Programmbibliothek auf und übergibt die Logindaten.
2	Auf MobileCoach-Serverseite wird überprüft, ob es sich um einen HTTP-basierten Login handelt. Liegt ein HTTP-Login vor, werden die im `token`-Parameter enthaltenen Logindaten in ein JSON-Dokument gewandelt.
3	Liegen die Logindaten eines externen Systems vor, wird eine entsprechende Validierung aufgerufen. Liegen Logindaten eines anderen Clients vor, werden anderweitige Validierungsfunktionen ausgeführt.
4	Das JSON-Dokument wird geparst und die enthaltenen Logindaten validiert.
5	Ist die Validierung erfolgreich, wird eine Response-Nachricht mit der Service-ID und Rolle des externen Systems zum Client zurückgeschickt. Schlägt die Validierung fehl, wird der negativer HTTP-Statuscode 403 (129c, Fielding, Roy, Reschke & Julian, 2019) zurückgeschickt.

Tabelle 5.1 Ablaufschritte für die Authentifizierung eines externen Systems

5.1.5 Übertragung einer externen Nachricht

Ist die Authentifizierung eines externen Systems erfolgreich, kann eine Kommunikation über Deepstream mit dem MobileCoach-Server stattfinden. Für die Übertragung von Nachrichten wird eine RPC-Methode mit folgender Spezifikation am Deepstream-Server registriert (FA13):

Methodenname	Parameter	Beschreibung	Rückgabewert
external-message	systemId	Die ID des externen Systems.	Ist die Nachrichtenübertragung erfolgreich → `true` sonst `false`.
	Participants	Ein Array, das Interventionsteilnehmer-IDs beinhaltet, an die eine Nachricht geschickt werden soll. Ist kein Array angegeben, wird die Nachricht an alle Interventionsteilnehmer der laufenden Intervention geschickt.	
	Variables	Ein verschachteltes JSON-Dokument mit Schlüssel-Wert-Paaren, die auf Interventionsvariablen abgebildet werden.	

Tabelle 5.2 Spezifikation der Nachrichtenübertragungs-RPC-Methode

Die `external-message` Methode wird über eine Deepstream-Client-Programmbibliothek aufgerufen und die Parameter als JSON-Dokument übergeben:

```
1 {
2   "systemId": "jvc7coe3-mx26o9kns1s",
3   "participants": [],
4   "variables": {
5     "jsonFeldA": "Test Nachricht"
6   }
7 }
```

Listing 5.3 Parameter der external-message-Methode

Auf Serverseite wird folgender Ablauf ausgeführt:

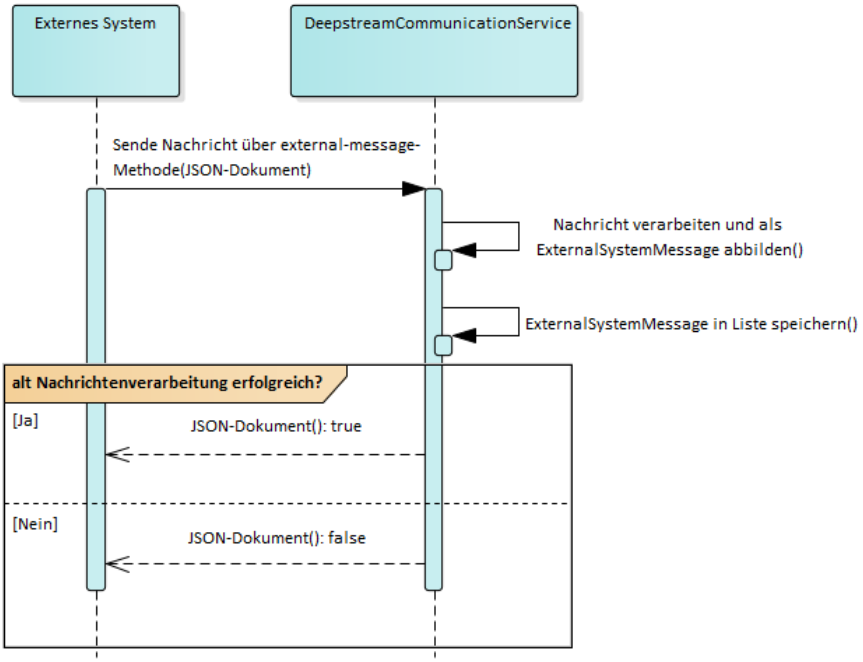


Abbildung 5.3 Sequenzdiagramm – Übertragung einer externen Nachricht

Schritt	Beschreibung
1	Die Nachricht wird von einem externen System über die `external-message-` RPC-Methode gesendet.
2	Diese wird in der `DeepstreamCommunicationService-`Klasse empfangen und verarbeitet. Die einzelnen Parameter bzw. Schlüssel-Wert-Paare des JSON-Dokuments werden einem transienten `ExternalSystemMessage-`Objekt zugewiesen. Die Klasse `ExternalSystemMessage` befindet sich in der Subkomponente `model/memory`.

3	Das `ExternalSystemMessage`-Objekt wird einer `receivedExternalSystemMessages`-Liste hinzugefügt. Diese wird zu einem späteren Zeitpunkt durch das Threading (Kapitel 3.5.3) zur weiteren Verarbeitung abgerufen.

Tabelle 5.3 Ablaufschritte für die Übertragung einer externen Nachricht

5.1.6 Erstellen der Interventionsregel „External Message“

Bevor eine externe Nachricht verarbeitet werden kann, muss die Möglichkeit bestehen, auf diese zu reagieren bzw. eine Aktion auszuführen (z. B. das Versenden einer Interventionsnachricht, FA14). Dafür wird eine neue Interventionsregel (Kapitel 3.5.1) „External Message“ angelegt. Diese ist über die Administrationsoberfläche im Entscheidungsbaum einer Intervention sichtbar.
Dafür müssen folgende Schritte umgesetzt werden:

Schritt	Beschreibung
1	In der `RecursiveAbstractMonitoringRulesResolver`-Klasse, die in der Subkomponente `services/internal` zu finden ist, wird eine neue Konstante `MONITORING_RULES_EXTERNAL_MESSAGE` der `EXECUTION_CASE`-Enumeration hinzugefügt. Mit dieser wird die Regelverarbeitung für die Interventionsregel „External Message“ ausgeführt.
2	Für die Anzeige in der Administrationsoberfläche wird ein neuer Regeltyp `EXTERNAL_MESSAGE` in der `MonitoringRuleTypes`-Enumeration definiert. Die Enumeration befindet sich in der Subkomponente `persistent/types` und wird durch die Entität `MonitoringRule` persistiert.
3	Die `EXTERNAL_MESSAGE`-Interventionsregel wird initial beim Erstellen einer Intervention über die `InterventionAdministrationManagerService`-Klasse in der `services`-Komponente angelegt und in der Administrationsoberfläche angezeigt.

Tabelle 5.4 Ablaufschritte für die Anzeige einer Interventionsregel in der Administrationsoberfläche

Für die Überprüfung auf ein bestimmtes externes System in der „External Message“ Interventionsregel werden die Systemvariablen `$externalSystemId` und `$externalSystemName` angelegt. Wird eine externe Nachricht verarbeitet, werden der `$externalSystemId`-Variable die externe System-ID und der `$externalSystemName`-Variable der externe Systemname des externen Systems, von dem die Nachricht gesendet wurde, zugewiesen. Dafür werden folgende Schritte umgesetzt:

Schritt	Beschreibung
1	In der `SystemVariables`-Klasse der `services`-Komponente wird die neue Enumeration `READ_ONLY_EXTERNAL_SYSTEM_VARIABLES` angelegt. In dieser werden die Enumerationswerte `externalSystemId` und `externalSystemName` definiert.
2	Die Enumeration wird in der `VariablesManagerService`-Klasse, die in der Subkomponente `services/internal` zu finden ist, referenziert. In der Methode `getAllVariablesWithValuesOfParticipantAndSystemAndExternalSystem` wird die Enumeration iteriert. Den Variablen bzw. Enumerationswerten `externalSystemId` und `externalSystemName` werden entsprechende Werte eines der Methode übergebenen `InterventionExternalSystem`-Objekts zugewiesen. Die Variablen werden in einem `Hashtable` von der Methode zurückgegeben.

Tabelle 5.5 Ablaufschritte für die Wertzuweisung von Systemvariablen

5.1.7 Verarbeitung einer externen Nachricht im Entscheidungsbaum

Der MobileCoach-Server speichert alle empfangenen Nachrichten (z. B. von der MobileCoach-App) als `ReceivedMessage`-Objekt. Dieses wird innerhalb der `InterventionExecutionManagerService`-Klasse der `services`-Komponente verarbeitet.
Um die Verarbeitung einer externen Systemnachricht zu ermöglichen, wird eine neue Service-Klasse `ExternalSystemsManagerService` in der `services`-Komponente erstellt. Diese muss die empfangenen externen Nachrichten, die als `ExternalSystemMessage`-Objekte vorliegen, auf `ReceivedMessage`-Objekte abbilden. Dafür wird die bestehende `ReceivedMessage`-Klasse, befindlich in der `model/memory`-Komponente, um Attribute erweitert. Diese sollen spezifische Werte eines `ExternalSystemMessage`-Objekts aufnehmen.
Folgende Attribute werden definiert: *1)* **externalSystemId:** Die externe System-ID*; 2)* **externalSystem:** Ein boolescher Wert, der angibt, ob das `ReceivedMessage`-Objekt eine externe Nachricht darstellt*; 3)* **externalSystemsVariables:** Eine Liste, die die abgebildeten Schlüssel-Wert-Paare eines externen Systems auf Interventionsvariablen enthält.
Die Abbildung eines `ExternalSystemMessage`-Objekts auf ein `ReceivedMessage`-Objekt durchläuft folgende Schritte (ein detailliertes Sequenzdiagramm befindet sich im Anhang 1.10):

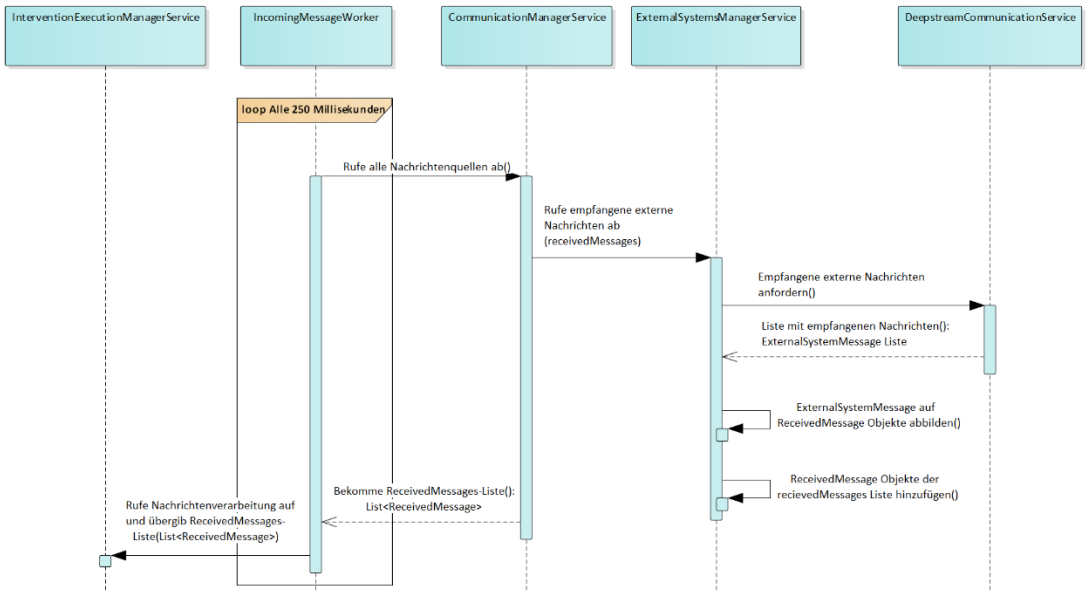


Abbildung 5.4 Sequenzdiagramm (vereinfacht) – Abbildung einer ExternalSystemMessage auf ein ReceivedMessage-Objekt

Schritt	Beschreibung
1	Das Threading (Kapitel 3.5.3) löst den Abruf empfangener Nachrichten aus und ruft alle Nachrichtenquellen, z. B. über den `CommunicationManagerService`, ab.
2	Der `CommunicationManagerService` ruft empfangene externe Nachrichten vom `ExternalSystemsManagerService` ab. Dafür wird diesem eine Liste (`receivedMessages`) übergeben, die dieser befüllt.
3	Der `ExternalSystemsManagerService` fordert die empfangenen externen Nachrichten vom `DeepstreamCommunicationService` an. Dieser gibt eine Liste mit externen Nachrichten zurück.
4	Die zurückgegeben externen Nachrichten liegen als `ExternalServiceMessage`-Objekte vor. Diese müssen für die weitere Verarbeitung in `ReceivedMessage`-Objekte gewandelt bzw. auf diese abgebildet werden.
5	Die umgewandelten `ReceivedMessage`-Objekte werden der Liste `receivedMessages` hinzugefügt.
6	Der `CommunicationManagerService` gibt die `receivedMessages`-Liste an den `IncomingMessageWorker` zurück.
7	Der `IncomingMessageWorker` übergibt die Liste zur weiteren Verarbeitung an den `InterventionExecutionManagerService`.

Tabelle 5.6 Ablaufschritte für die Abbildung einer ExternalSystemMessage auf ein ReceivedMessage-Objekt

Die weitere Verarbeitung der zu `ReceivedMessage`-Objekten umgewandelten externen Nachricht wird im `InterventionExecutionManagerService` fortgesetzt und hat nachfolgenden Ablauf:

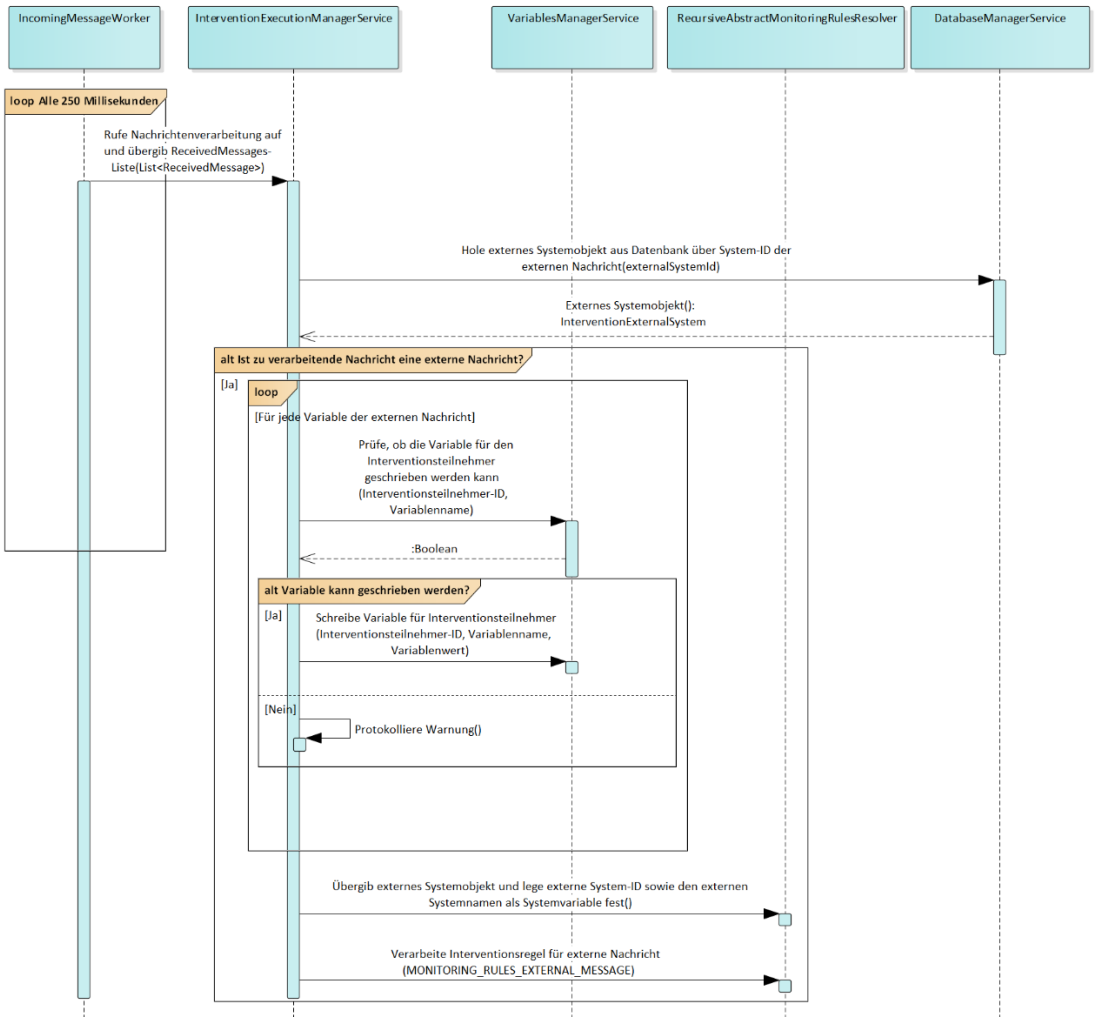


Abbildung 5.5 Sequenzdiagramm – Verarbeitung einer externen Nachricht

Schritt	Beschreibung
1	Die Nachrichtenliste mit den `ReceivedMessage`-Objekten wird dem `InterventionExecutionManagerService` übergeben.
2	Die Nachrichtenliste wird in einer Schleife durchlaufen, sodass jede Nachricht pro Interventionsteilnehmer einzeln verarbeitet wird.
3	Während der Verarbeitung wird versucht ein externes Systemobjekt (`InterventionExternalSystem`) aus der Datenbank zu beziehen. Ist die zu verarbeitende Nachricht keine externe Nachricht, wird das externe Systemobjekt ignoriert.

4	Handelt es sich um eine externe Nachricht, werden die in der externen Nachricht enthaltenden Variablen für den Interventionsteilnehmer geschrieben. Damit wird der Zugriff auf diese in Interventionsdialogen oder Interventionsnachrichten ermöglicht.
5	Die Regelverarbeitung für die Interventionsregel „External Message“ wird angestoßen. Das zuvor aus der Datenbank bezogene externe Systemobjekt wird der Regelverarbeitung übergeben. Aus diesem werden die Informationen für die Systemvariablen `$externalSystemId` und `$externalSystemName` bezogen.

Tabelle 5.7 Ablaufschritte für die Verarbeitung einer externen Nachricht

5.2 GET.ON-Plattform

Um eine Verknüpfung über die MobileCoach-App mit der GET.ON-Plattform zu realisieren, wird eine Weboberfläche erstellt. Die App kann diese im Interventionschat über eine Webansicht anzeigen. Dies hat folgende Vor- und Nachteile:

Vorteile: *1)* Die MobileCoach-App muss für neue Funktionalitäten nicht erweitert werden*; 2)* Eine flexible Anpassung der Weboberfläche für sämtliche Endgeräte (u. a. Smartphone oder PC) ist möglich.

Nachteile: *1)* Ein Zugriff auf native Funktionen des Endgeräts kann nicht oder nur eingeschränkt erfolgen.

5.2.1 Weboberfläche für die Verknüpfung mit der GET.ON-Plattform

Die Weboberfläche ermöglicht die Eingabe der Logindaten eines GET.ON-Profils. Dafür wird ein Textfeld zur Eingabe des Benutzernamen oder der E-Mail-Adresse angezeigt. In einem darunterliegenden Textfeld kann das Passwort angegeben werden.
Ein Mockup der Oberfläche befindet sich im Anhang 2.4. Folgende Funktionalitäten werden über die Oberfläche angeboten:

GET.ON-Profil verknüpfen (*GET.ON-Plattform verknüpfen*, FA10) Die eingegebenen Logindaten sowie die Interventionsteilnehmer-ID der MobileCoach-App werden an die GET.ON-Plattform gesendet. Sind die Logindaten nicht valide oder besteht bereits eine Verknüpfung zu einem GET.ON-Profil, wird eine Fehler- bzw. Informationsmeldung angezeigt.

5.2.2 Datenmodell für die Verknüpfung mit der GET.ON-Plattform

Die Daten für eine Verknüpfung werden in der MYSQL-Datenbank der GET.ON-Plattform persistiert.

Die Tabelle `wp_mc_participants`, die als Attribute die GET.ON-Profil-ID des Benutzers und die Interventionsteilnehmer-ID der MobileCoach-App enthält, wird erstellt. Die Profil-ID wird aus der Tabelle `wp_users` als Fremdschlüssel angelegt. Beide Attribute sind als zusammengesetzter Primärschlüssel konzipiert und eindeutig.
Eine Profil-ID kann zu n weiteren Interventionsteilnehmer-IDs zugeordnet sein (n: m).
Folgendes Entity-Relationship-Diagramm (ER-Diagramm) wird erstellt (grüne Entitäten sind bereits in der GET.ON-Plattform vorhanden):

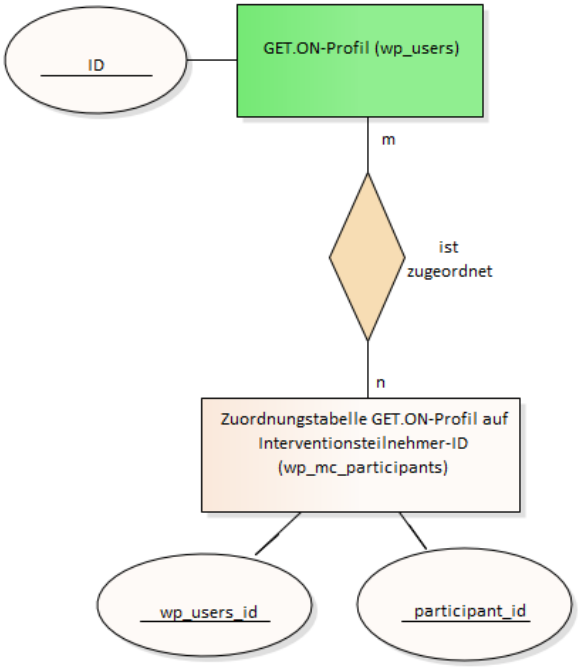


Abbildung 5.6 GET.ON-ER-Diagramm

5.2.3 Verknüpfung mit der GET.ON-Plattform

Um einem GET.ON-Studienteilnehmer Interventionsnachrichten über die MobileCoach-App zu schicken, muss eine eindeutige Zuordnung zwischen dem Studienteilnehmer und der App bestehen (FA10). Diese kommt zustande, indem die GET.ON-Profil-ID des Studienteilnehmers einer Interventionsteilnehmer-ID zugeordnet wird. Die Interventionsteilnehmer-ID wird vom MobileCoach-Server vergeben und kann jeden Interventionsteilnehmer eindeutig identifizieren (Kapitel 2.2.4.1). Die App bezieht die ID vom MobileCoach-Server.

5.2.3.1 Anzeige der Weboberfläche

Die Weboberfläche wird im Interventionschat der App angezeigt. Dafür wird über einen Interventionsdialog vom MobileCoach-Server eine Interventionsnachricht geschickt, die die

URL von der Weboberfläche auf GET.ON-Serverseite enthält. Mit einem in der Nachricht enthaltenen Befehl (*show-web*) kann diese URL im Interventionschat geöffnet werden:

Abbildung 5.7 „GET.ON-Plattform verknüpfen“ Button in der MobileCoach-App

5.2.3.2 Übertragung der Logindaten und Interventionsteilnehmer-ID

Die in der Weboberfläche eingegeben Logindaten werden als HTTP-POST-Argument an die GET.ON-Plattform weitergegeben. Die Interventionsteilnehmer-ID wird als HTTP-GET-Argument zusammen mit der URL in der Interventionsnachricht (siehe Abbildung 5.7) gesendet. Somit werden folgende Parameter an die GET.ON-Plattform übergeben:

Beispiel-URL
`https://coachstaging.geton-training.de/?page_id=45284&pid=$participantIdentifier`[8]

Parameter	Typ	Beschreibung
loginUser	POST	Benutzername/E-Mail-Adresse des zu verknüpfenden GET.ON-Profils.
loginPwd	POST	Passwort des zu verknüpfenden GET.ON-Profils.
pid	GET	Interventionsteilnehmer-ID

Tabelle 5.8 Verknüpfungs-URL-Parameter

[8] Das `page_id` Argument stellt die eindeutige Seiten-ID der Weboberfläche dar. Diese wird von WordPress vergeben (Kapitel 2.3.1).

5.2.3.3 Verarbeitung der übertragenen Daten auf GET.ON-Serverseite

Die vom HTTP-Protokoll übertragenen Daten werden auf GET.ON-Serverseite verarbeitet (FA11). Dazu werden u. a. Funktionalitäten der Wordpress-API genutzt. Die Daten durchlaufen folgenden Funktionen:

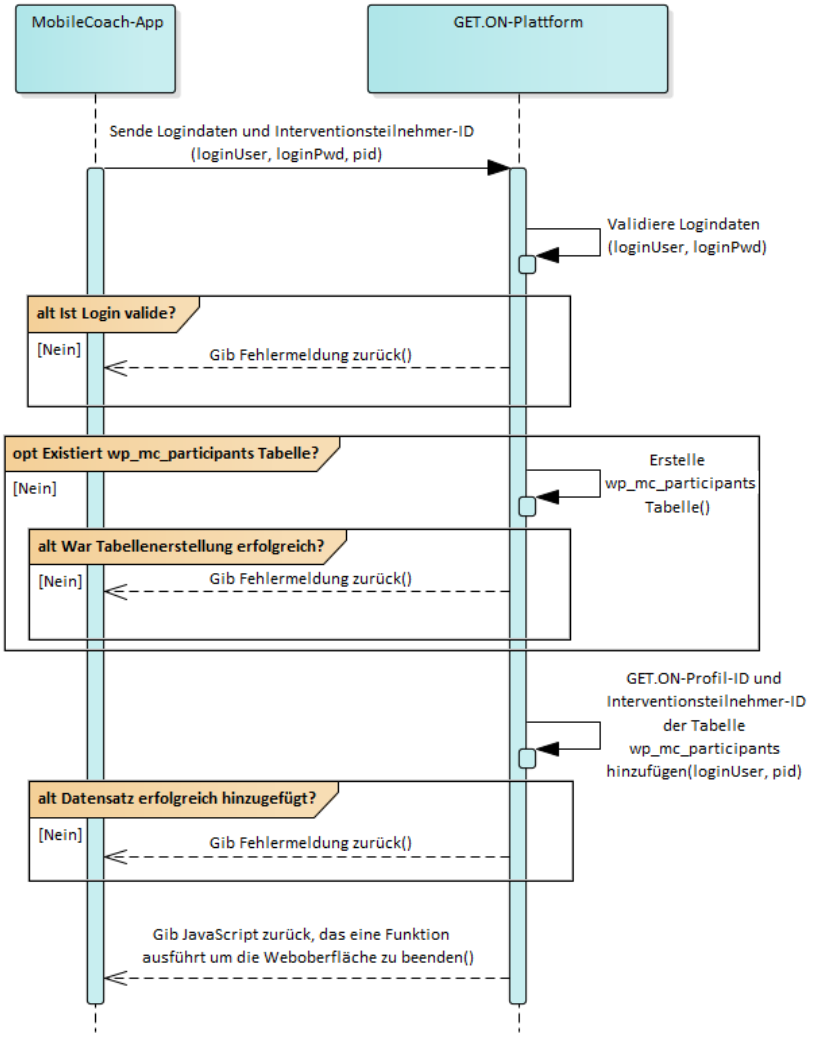


Abbildung 5.8 Sequenzdiagramm – Verarbeitung der übertragenen Daten auf GET.ON-Serverseite

Schritt	Beschreibung
1	Der Interventionsteilnehmer gibt seine Logindaten in der Weboberfläche ein und drückt den Button „Mit GET.ON verknüpfen“. Benutzername/E-Mail-Adresse, das Passwort und die Interventionsteilnehmer-ID (`loginUser`, `loginPwd`, `pid`) werden zur GET.ON-Plattform übertragen.

2	Über die WordPress-API werden die Logindaten (`loginUser`, `loginPwd`) validiert. Sind diese nicht valide, wird eine Fehlermeldung zurückgeschickt und in der Weboberfläche angezeigt.
3	Falls die Zuordnungstabelle `wp_mc_participants` noch nicht existiert, wird diese angelegt. Tritt bei der Erstellung ein Fehler auf, wird eine Fehlermeldung zurückgeschickt und in der Weboberfläche angezeigt.
4	Die Profil-ID des GET.ON-Studienteilnehmers wird anhand der Logindaten (`loginUser`) aus der Datenbank bezogen. Diese wird zusammen mit der übertragenen Interventionsteilnehmer-ID (`pid`) in die Tabelle `wp_mc_participants` geschrieben. Tritt ein Fehler auf oder existiert der Datensatz bereits in der Tabelle, wird eine Fehler-/Warnmeldung zurückgeschickt und in der Weboberfläche angezeigt.
5	Ist der Datensatz erfolgreich in die Tabelle `wp_mc_participants` geschrieben, wird eine JavaScript-Funktion ausgeführt, die die Weboberfläche in der Mobile-Coach-App beendet.

Tabelle 5.9 Ablaufschritte für die Verarbeitung der übertragenen Daten auf GET.ON-Serverseite

6 Realisierung

In diesem Kapitel wird der zuvor erstellte Entwurf realisiert. Dafür werden auf MobileCoach- sowie GET.ON-Serverseite neue Funktionalitäten erstellt und vorhandene erweitert.
Die spezifizierten Anforderungen (Kapitel 4.4) der Anwendungsfälle (Kapitel 4.3) werden implementiert.

6.1 MobileCoach-Server

Die Implementierung auf MobileCoach-Serverseite ist in der Programmiersprache Java umgesetzt.
Es existiert keine Programmierkonvention bis auf die Vorlage einer allgemeinen Quelltextformatierung. Daher orientiert sich die Implementation an der Struktur des vorhandenen Quelltextes und nutzt die Vorlage „MobileCoach_Formatter_v02“ zur Quelltextformatierung.

6.1.1 Implementierung AWF01

Wie im fachlichen Datenmodell (Abbildung 5.1) beschrieben, wird in der Subkomponente `model/persistent` die Entität `InterventionExternalSystem` angelegt (Kapitel 3.5.5).
Die Oberflächensubkomponente `external_systems` wird in die `ui`-Komponente `components/main_view/interventions` integriert (Kapitel 3.5.7). Für die Darstellung des Tabs „External Systems“ in der Administrationsoberfläche ist die Klasse `ExternalSystemsTabComponent` verantwortlich (Listing 2). In der `buildMainLayout`-Methode ist das Hauptlayout für die tabellarische Übersicht zur Anzeige externer Systeme implementiert. Die Klasse `ExternalSystemsEditComponent` definiert die eigentliche Oberfläche der tabellarischen Übersicht (Listing 1). Dafür ist neben einem Haupt- und Buttonlayout, ein TextArea-Layout angelegt, das u. a. die generierten Authentifizierungsdaten eines in der Tabelle selektierten externen Systems als JSON-`String` ausgibt. Dieser kann kopiert und der `login`-Methode eines Deepstream-Clients übergeben werden. Die `ExternalSystemsTabComponentWithController`-Klasse implementiert die Geschäftslogik der Oberfläche (Anhang 4.3). Über die `createExternalSystem`-Methode wird ein neues externes System erzeugt und in der tabellarischen Übersicht angezeigt. Dafür wird die Erzeugung über die Methode `interventionExternalSystemCreate` zum `InterventionAdministrationManagerService` delegiert. Dieser ruft u. a. die Methode `registerExternalSystem` in der Klasse `DeepstreamCommunicationService` (Listing 8) auf die die Authentifizierungsdaten wie in Kapitel 5.1.3 beschrieben anlegt.
Für die weiteren Oberflächenfunktionalitäten sind ähnliche Methoden (z. B. `renameExternalSystem` etc.) implementiert.

6.1.2 Implementierung AWF02

Das fachlichen Datenmodell (Abbildung 5.1) beschreibt die Entität `InterventionExternalSystemFieldVariableMapping`. Diese wird in der Subkomponente `model/persistent` angelegt.
Die bestehende `external_systems`-Komponente wird um die Klassen `ExternalSystemsFieldVariableMappingComponent` (Listing 6) und `ExternalSystemsFieldVariableMappingComponentWithController` (Listing 7) erweitert.
Erstere übernimmt die Definition der Oberfläche zur tabellarischen Anzeige der Abbildungen von Schlüssel-Wert-Paaren auf Interventionsvariablen. Da die Oberfläche in einem modalen Fenster angezeigt wird, ist neben dem Hauptlayout ein Buttonlayout implementiert das einen „Close“ Button anzeigt. Auf diesem kann über die Methode `registerOkButtonListener` ein `Listener` definiert werden, der weitere Funktionen beim Schließen des Fensters ausführt.
Letztere implementiert die Geschäftslogik u. a. zur Erstellung von Zuordnungen der Schlüssel- bzw. Feldnamen auf Interventionsvariablen. Dafür wird eine Erstellung über die Methode `createExternalSystemFieldVariableMapping` zum `InterventionAdministrationManagerService` delegiert.
Damit eine Zuordnung von Schlüssel- bzw. Feldnamen auf Interventionsvariablen in der Oberfläche angelegt werden kann, muss eine dafür vorgesehene Standard-Oberflächenkomponente existieren. Da dies nicht der Fall ist, wird die Oberflächenkomponente `ShortStringEditWithComboBoxComponent` in der Subkomponente `ui/components/basics` implementiert (Listing 5). Diese definiert ein Textfeld und eine Combobox. Über die Methoden `getStringValue` und `getSelectedVariable` kann ein gewählter Schlüssel- bzw. Feldname sowie eine gewählte Interventionsvariable bezogen werden. Für die initiale Befüllung der Oberflächenelemente, sind die Methoden `setStringValue(final String value)` und `addVariables(List<String> variables)` mit entsprechenden Parametern aufzurufen.

6.1.3 Implementierung AWF04

Die Klasse `DeepstreamRESTServlet` der `servlet`-Komponente wird um die Funktionalität zur Authentifizierung eines externen Systems erweitert (Listing 13). Diese übernimmt Deepstream-Anfragen über das HTTP- oder Websocket-Protokoll. Bei der Authentifizierung über HTTP führt die Methode `httpAuthTokenToJson` die Umwandlung der Authentifizierungsdaten in ein JSON-Dokument aus. Über die Methode `checkExternalSystemAccess` wird die Authentifikation an die `RESTManagerService`-Klasse delegiert. Diese validiert die Logindaten einer Anfrage (Listing 14).
In der `DeepstreamCommunicationService`-Klasse ist die RPC-Methode `external-message` registriert (Listing 9). Beim Methodenaufruf durch ein externes System wird aus dem empfangenen JSON-Dokument ein `ExternalSystemMessage`-Objekt erzeugt. Dieses wird einer Liste hinzugefügt, auf die das Threading zu einem späteren Zeitpunkt zugreift. Beim späteren Abruf der Methode `getReceivedMessages` durch das Threading übernimmt die Klasse `ExternalSystemsManagerService` die weitere Verarbeitung und das Abbilden auf `ReceivedMessage`-Objekte (Kapitel 4.13). Anschließend werden die `ReceivedMessage`-Objekte in der Methode

`handleReceivedMessage` der `InterventionExecutionManagerService`-Klasse verarbeitet. Diese wurde um die Implementation erweitert, die das Durchlaufen einer „External Message“ Interventionsregel ermöglicht. Die Interventionsregel „External Message“, ist innerhalb der Klasse `RecursiveAbstractMonitoringRulesResolver` implementiert (Listing 12). Dafür wird die Methode `executeRules` um den Enumerationswert MONITORING_RULES_EXTERNAL_MESSAGE erweitert. Auf diesen wird während des rekursiven Regeldurchlaufs geprüft, worauf die in der Administrationsoberfläche angelegten „External Message“ Regeln evaluiert werden.
Die Klasse `SystemVariables` wird für die Implementierung der Systemvariablen `$externalSystemId` und `$externalSystemName` erweitert (Listing 10). Die Funktionalität für eine korrekte Wertzuweisung der Variablen, ist in der Methode `getAllVariablesWithValuesOfParticipantAndSystemAndExternalSystem` der Klasse `VariablesManagerService` umgesetzt (Listing 11).

6.2 GET.ON-Plattform

Die Implementation auf GET.ON-Serverseite ist mit Snippets (Kapitel 2.3.1) umgesetzt die eine in der Datenbank gespeicherte Quelltextdatei darstellen.
Als Programmiersprache kommt PHP und JavaScript zum Einsatz. Mit HTML und CSS ist die Weboberfläche beschrieben.

6.2.1 Implementierung AWF03

Das Snippet „MobileCoach get.on connection login form“ beinhaltet die Beschreibung der Weboberfläche. HTML definiert das Webformular das die Logindaten zur Verknüpfung eines GET.ON-Profils mit der MobileCoach-App übermittelt. Über JavaScript werden die Textfelder des Formulars clientseitig auf leere Einträge überprüft. Dies hat den Vorteil, dass bei einer unvollständigen Angabe der Logindaten keine Validierung auf Serverseite stattfinden muss, der Server entlastet und eine unnötige Datenübertragung vermieden wird.
Ist die Angabe der Logindaten vollständig und die Übertragung erfolgreich wird die in PHP implementierte, serverseitige Datenverarbeitung ausgeführt. Diese ist im Snippet „MobileCoach add participant id to get.on plattform“ implementiert und über einen Shortcode `[shortcode_connect_mobile_coach]` in der Weboberfläche referenziert (Listing 18).

6.2.2 Integration

Die realisierte Kommunikationsschnittstelle kann auf unterschiedlichen Arten in die GET.ON-Plattform integriert werden. Die im Anhang 6 befindliche Integrationsanleitung beschreibt das Szenario eine Erinnerungsnachricht an einen Studienteilnehmer zu versenden, sobald ein Zeitfenster abgelaufen ist.

6.3 Ergebnis

Die Implementation wurde für jeden Anwendungsfall erfolgreich umgesetzt. Im Anhang 3 befinden sich Screenshots der final implementierten Oberflächen.

7 Softwaretest

In diesem Kapitel wird der Entwurf und die daraus resultierende Implementierung gegen die Anforderungen der Software getestet.

7.1 Testumgebung

MobileCoach-Server Der Server wird auf einem DigitalOcean-Droplet (einer virtuellen Serverinstanz) ausgeführt. Diese besitzt folgende Spezifikationen:

- OS: Debian 9.9
- Kernel: 4.9.0-9-amd64
- CPU: Intel Xenon E5-2650 (2 Kerne) mit 2.20GHz
- RAM: 2GB
- Software: JDK 1.8.0_212, Tomcat 8.5, Deepstream 3.1.0-1, MongoDB 3.6.4

GET.ON-Plattform Die GET.ON-Plattform wird bereits auf einer Leuphana internen Testumgebung ausgeführt. Daher entfällt die Angabe der Umgebungsspezifikation.

MobileCoach-App Die App wird lediglich auf einem Android-Smartphone getestet, da aus organisatorischen Gründen kein iOS fähiges Gerät zur Verfügung stand:

- OS: Android 7.1.2
- Gerät: Nexus 5
- CPU: ARMv7 (4 Kerne) mit 2.20GHz
- RAM: 2GB

7.2 Testkonzept

Weder der MobileCoach-Server noch die GET.ON-Plattform setzten aktuell Unit-Tests um. Daher muss auf einen automatisierten Test-Driven-Ansatz verzichtet werden.
Ein manueller Systemtest wird durchgeführt der die spezifizierten Anforderungen (Kapitel 4.4) auf Fehler überprüft.
Um die Kommunikation zwischen GET.ON-Plattform und MobileCoach-Server zu testen, ist eine Testoberfläche auf GET.ON-Serverseite implementiert. Diese kann nur über ein administratives WordPress-Benutzerprofil abgerufen werden und ist nicht öffentlich einsehbar.

7.3 Testfälle

Anhand der spezifizierten Anwendungsfehlerfälle (Kapitel 4.3) wird die implementierte Oberfläche auf MobileCoach- und GET.ON-Serverseite getestet.
Die Nachrichtenkommunikation zwischen der GET.ON-Plattform und dem MobileCoch-Server wird mit der GET.ON-Testoberfläche geprüft.

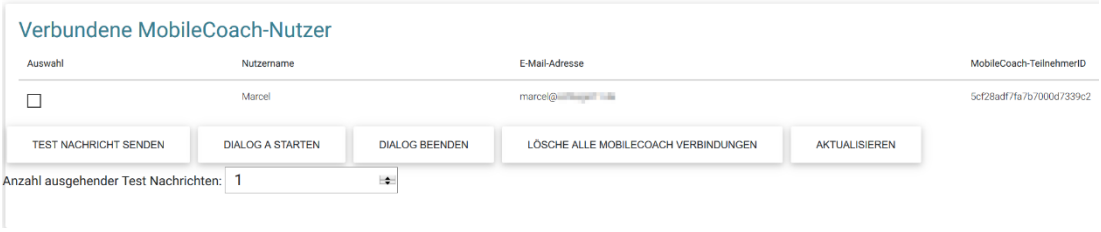


Abbildung 7.1 GET.ON-Testoberfläche

Für diese Tests finden sich entsprechende Testfälle im Anhang 5. Diese decken die Anforderungen FA12, FA13 und NFA01 ab. Eine laufende Studie der GET.ON-Plattform kann im Schnitt 150 bis 300 Studienteilnehmer besitzen. Davon sind ca. 50 bis 100 Teilnehmer über einen begrenzten Zeitraum (z. B. 24 Stunden) parallel aktiv. Als Anforderung sollen mindestens 20 Nachrichten parallel/sequentiell in einem kurzen Zeitraum an Studienteilnehmer versendet werden können. Daher stellt die nicht funktionale Anforderung NFA01 ein wichtiges Kriterium für den Einsatz innerhalb der GET.ON-Plattform dar.

7.4 Ergebnis

Alle spezifizierten Fehler- und Testfälle sind erfolgreich überprüft und lieferten die erwarteten positiven Ergebnisse.
Nachfolgend die erfüllten Anforderungen:

Anforderung	Ergebnis
FA01	Ein externes System kann über die Administrationsfläche des MobileCoach-Servers erstellt werden.
FA02	Die Umbenennung eines externen Systems ist möglich.
FA03	Die Erzeugung eines neuen Tokens für ein externes System ist möglich.
FA04	Das externe System kann aktiviert/deaktiviert werden. Für ein deaktiviertes externes System ist eine Authentifizierung nicht möglich.
FA05	Externe Systeme können gelöscht werden. Sind diesen Abbildungen von Schlüssel-Wert-Paaren auf Interventionsvariablen zugeordnet, werden diese ebenfalls gelöscht.
FA06	Die Abbildungen von Schlüssel-Wert-Paaren auf Interventionsvariablen für ein externes System können über die Administrationsoberfläche erstellt werden.
FA07	Schlüsselnamen der Abbildungen können im Nachhinein umbenannt werden.
FA08	Eine zu einem Schlüsselnamen zugeordnete Interventionsvariable lässt sich ändern.
FA09	Erstellte Abbildungen von Schlüssel-Wert-Paaren auf Interventionsvariablen lassen sich löschen.
FA10	Eine Weboberfläche für die Verknüpfung der MobileCoach-App mit einem GET.ON-Profil wurde erstellt.
FA11	Die GET.ON-Plattform wurde für die Authentifizierung aus der MobileCoach-App erweitert.
FA12	Externe Systeme können sich mit validen Logindaten am MobileCoach-Server anmelden.
FA13	Daten eines externen Systems können über eine Schlüssel-Wert-Paar-Datenstruktur an den MobileCoach-Server gesendet werden.
FA14	Über eine Interventionsregel „External Message“ kann auf übertragene Nachrichten eines externen Systems reagiert werden. Die enthalten Nachrichtendaten lassen sich z. B. als Interventionsnachricht an die MobileCoach-App eines Studienteilnehmers senden.
NFA01	Die Nachrichtenschnittstelle kann mindestens 20 Nachrichten parallel/sequentiell verarbeiten.

Tabelle 7.1 Erfüllte Anforderungen

8 Schluss

8.1 Fazit

Die Entwicklung der Nachrichtenschnittstelle ist erfolgreich abgeschlossen. Vorausgehend wurde eine Dokumentation erstellt, die die wichtigsten Bestandteile und Konzepte des MobileCoach-Servers beschreibt. Ohne diese wäre eine Entwicklung und Realisierung nur schwer möglich gewesen. Auch für zukünftige Erweiterungen der MobileCoach-Plattform bzw. des MobileCoach-Servers, kann die Dokumentation ein hilfreiches Werkzeug sein.
Die Planung und Entwurfserstellung der Nachrichtenschnittstelle war besonders umfangreich. Es sollte keine Schnittstelle entworfen werden, die ausschließlich mit der GET.ON-Plattform kommuniziert, sondern eine generische Lösung, die unabhängig von einem spezifischen System *sämtliche* externe Systeme ansprechen kann. Dies wurde mit der Implementierung einer tokenbasierten Kommunikation über Deepstream erreicht.
Für die Verknüpfung der MobileCoach-App mit der GET.ON-Plattform wurde eine Lösung gesucht, die einen möglichst geringen Implementationsaufwand besitzt. Der Entwurf einer webbasierten Verknüpfungsoberfläche und die Realisierung durch Einbettung dieser in die MobileCoach-App machen die Lösung, trotz geringen Aufwands, besonders flexibel. Eine Erweiterung der MobileCoach-App wurde dadurch vermieden.
Die Integrationsanleitung beschreibt, wie Erinnerungsnachrichten über die GET.ON-Plattform an die MobileCoach-App von Studienteilnehmern gesendet werden können.
Dabei nutzt das Szenario nur einen kleinen Teil der bestehenden Möglichkeiten.
Durch die Integration externer Systeme in den Entscheidungsbaum einer Intervention auf MobileCoach-Serverseite, können über externe Nachrichten komplette Interventionen dynamisch gesteuert und mit übertragenen Daten ergänzt werden.
Zukünftig können somit die unterschiedlichsten Szenarien von der GET.ON-Plattform umgesetzt werden, die einem Studienteilnehmer mit Hilfe der MobileCoach-App beim Durchlaufen seiner Trainingseinheiten helfen und unterstützen.

8.2 Ausblick

Basierend auf dieser Arbeit sind folgende zukünftige Entwicklungen und Erweiterungen möglich:

8.2.1 Erweitern der Dokumentation des MobileCoach-Servers

Mit Hilfe von arc42 (Kapitel 2.1.1) ist die Dokumentation des MobileCoach-Servers auch zukünftig leicht erweiterbar. Da sich die Dokumentation in dieser Arbeit weitestgehend auf den technischen Teil beschränkt, ist primär der fachliche Teil zu ergänzen.
Dennoch könnten auch auf technischer Ebene weitere Konzepte vertieft werden. Dazu gehören Themen wie Logging, Internationalisierung, REST-Schnittstelle oder Build-Management.

8.2.2 Test-driven development der MobileCoach-Plattform

In dieser Arbeit wurden Softwaretests manuell durchgeführt. Das lag zum einen daran, dass ein Konzept oder eine Konvention für einen testgetrieben Entwicklungsansatz (test-driven development → TDD) fehlte. Auf MobileCoach-Serverseite könnten z. B. Unit-Tests die Stabilität und Robustheit der Software erhöhen. Darauf aufbauend wäre der Einsatz von Continuous Integration (CI) denkbar. Dadurch würden Unit-Tests automatisiert, bei Änderung des Quelltextes der Software, ausgeführt. So könnten z. B. Fehler einer Implementation frühzeitig entdeckt werden.

8.2.3 Ereignis-basierte Erweiterung der Nachrichtenschnittstelle

Aktuell existiert mit der external-message-RPC-Methode ein synchroner Kommunikationsweg zwischen der GET.ON-Plattform und dem MobileCoach-Server. Der Deepstream-Server bietet neben RPC-Methoden auch die Möglichkeit Events zu verarbeiten (deepstreamHub GmbH, 2019a). Der MobileCoach-Server könnte um eine Event-Methode erweitert werden, wodurch externe Systeme die Möglichkeit hätten sich auf diese zu registrieren. Dadurch könnten Interventionsdaten (z. B. die Antwort auf eine Interventionsnachricht eines Interventionsteilnehmers) asynchron an die GET.ON-Plattform übertragen und verarbeitet werden.

Literaturverzeichnis

129c, r. v. 1., Fielding, Roy, Reschke & Julian. (2019, 26. Mai). *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content.* Zugriff am 26.05.2019. Verfügbar unter https://tools.ietf.org/html/rfc7231#section-6

Adler, A. J., Martin, N., Mariani, J., Tajer, C. D., Owolabi, O. O., Free, C. et al. (2017). *Mobile phone text messaging to improve medication adherence in secondary prevention of cardiovascular disease.* Zugriff am 13.02.2019. https://doi.org/10.1002/14651858.CD011851.pub2

Behrends, E. (2018). *React Native. Native Apps parallel für Android und iOS entwickeln* (Safari Tech Books Online, 1. Auflage). Heidelberg: O'Reilly; dpunkt.verlag.

Bhusal, S. (2015). *Difference between ORM and ODM.* Zugriff am 22.04.2019. Verfügbar unter https://cbabhusal.wordpress.com/2015/01/03/difference-between-orm-and-odm/

Bunge, S. (2019). *Code Snippets.* Zugriff am 24.05.2019. Verfügbar unter https://de.wordpress.org/plugins/code-snippets/#description

DeepstreamHub GmbH (Hrsg.). (2019a). *Events.* Zugriff am 01.06.2019. Verfügbar unter https://deepstreamhub.com/tutorials/guides/events/#how-to-use-events

DeepstreamHub GmbH (Hrsg.). (2019b). *Open source overview - deepstreamhub.* Zugriff am 10.05.2019. Verfügbar unter https://deepstreamhub.com/open-source/

DeepstreamHub GmbH (Hrsg.). (2019c). *PHP Client.* Zugriff am 31.05.2019. Verfügbar unter https://deepstreamhub.com/docs/client-php/DeepstreamClient/#deepstreamclient-url-authdata-

DeepstreamHub GmbH (Hrsg.). (2019d). *Records.* Zugriff am 09.05.2019. Verfügbar unter https://deepstreamhub.com/tutorials/guides/records/

DeepstreamHub GmbH (Hrsg.). (2019e). *Records.* Zugriff am 16.05.2019. Verfügbar unter https://deepstreamhub.com/tutorials/guides/records/#naming-records

DeepstreamHub GmbH (Hrsg.). (2019f). *Remote Procedure Calls.* Zugriff am 09.05.2019. Verfügbar unter https://deepstreamhub.com/tutorials/guides/remote-procedure-calls/

DeepstreamHub GmbH (Hrsg.). (2019g). *Valve Permissions.* Zugriff am 27.05.2019. Verfügbar unter https://deepstreamhub.com/docs/general/valve/

Eilebrecht, K. & Starke, G. (2019). *Patterns kompakt. Entwurfsmuster für effektive Softwareentwicklung* (IT kompakt, 5., aktualisierte und erweiterte Auflage). Berlin: Springer Berlin.

Filler, A., Kowatsch, T., Haug, S., Wahle, F., Staake, T. & Fleisch, E. (2015). *MobileCoach: A novel open source platform for the design of evidence-based, scalable and low-cost behavioral health interventions: Overview and preliminary evaluation in the public health context.* Zugriff am 14.02.2019. https://doi.org/10.1109/WTS.2015.7117255

Fowler, M. (2019a, 18. April). *P of EAA: Active Record.* Zugriff am 22.04.2019. Verfügbar unter https://www.martinfowler.com/eaaCatalog/activeRecord.html

Fowler, M. (2019b, 18. April). *P of EAA: Identity Field.* Zugriff am 22.04.2019. Verfügbar unter https://www.martinfowler.com/eaaCatalog/identityField.html

Fowler, M. (2019c, 18. April). *P of EAA: Layer Supertype.* Zugriff am 22.04.2019. Verfügbar unter https://martinfowler.com/eaaCatalog/layerSupertype.html

Fowler, M. & Rice, D. (2011). *Patterns of enterprise application architecture* (The Addison-Wesley signature series, 17. print). Boston, Mass.: Addison-Wesley.

Glass, R. (1989). Software maintenance documentation. In *Proceedings of the 7th annual international conference on Systems documentation - SIGDOC '89* (S. 99–101). New York, New York, USA: ACM Press. https://doi.org/10.1145/74311.74325

Grönroos, M. (2016). *Book of Vaadin: Vaadin 7 Edition* (7th Revision): Vaadin Ltd.

Guérout, B. (2016, 11. Mai). *Jongo {mongo-java-driver: 'with ease'}.* Zugriff am 12.04.2019. Verfügbar unter http://jongo.org/#mapping

Hochschule für angewandte Wissenschaften Augsburg (Hrsg.). (2019, 19. Mai). *SQL Injection – GlossarWiki.* Zugriff am 24.05.2019. Verfügbar unter https://glossar.hs-augsburg.de/SQL_Injection

Ikeda, M. (2013). *2Mega (or up) Rates Members - JLUG RC5-72 Cracking.* Zugriff am 16.05.2019. Verfügbar unter http://www.orange.co.jp/~masaki/rc572/fratee.php

Jaspers, T. (2012). *MongoDB: Second Round - codecentric AG Blog,* codecentric AG. Zugriff am 22.04.2019. Verfügbar unter https://blog.codecentric.de/en/2012/11/mongodb-second-round/

Kowatsch, T. (Center for Digital Health Interventions, Chair of Operations Management at the Institute of Technology Management, University of St. Gallen, Hrsg.). (2019). *MobileCoach.* Zugriff am 05.04.2019. Verfügbar unter https://www.mobile-coach.eu/

MongoDB Inc. (Hrsg.). (2019). *The MongoDB 4.0 Manual — MongoDB Manual.* Zugriff am 10.05.2019. Verfügbar unter https://docs.mongodb.com/manual/

Oracle Corporation (Hrsg.). (2010). *Message-Oriented Middleware (MOM) (Sun Java System Message Queue 4.3 Technical Overview).* Zugriff am 09.05.2019. Verfügbar unter https://docs.oracle.com/cd/E19340-01/820-6424/aeraq/index.html

Richards, M. (2015). *Software architecture patterns. Understanding common architecture patterns and when to use them* (First edition). Sebastopol, CA: O'Reilly Media.

Schröder, H. (2017). *Brute-Force-Attacke und Passwortlänge,* Heiko Schröder. Zugriff am 16.05.2019. Verfügbar unter https://www.1pw.de/brute-force.php

Seemann, M. (2012). *Dependency injection in .NET*. Shelter Island, NY: Mannin.

Sibley, B. (2018). *WordPress Page ID - Where to Find It.* Zugriff am 22.05.2019. Verfügbar unter https://www.competethemes.com/blog/find-page-id/

Software Engineering Institute (Hrsg.). (2010). *What Is Your Definition of Software Architecture?* Zugriff am 03.04.19. Verfügbar unter https://resources.sei.cmu.edu/asset_files/FactSheet/2010_010_001_513810.pdf

Spanke (Leibniz-Institut für Wissensmedien, Hrsg.). (2015, 21. Juli). *Client-Server — e-teaching.org.* Zugriff am 23.04.2019. Verfügbar unter https://www.e-teaching.org/technik/vernetzung/architektur/client-server

Starke, G. (2017). *Effektive Softwarearchitekturen*. München: Carl Hanser Verlag GmbH & Co. KG. https://doi.org/10.3139/9783446454200

Starke, G. & Hruschka, P. (2016). *arc42 in Aktion*. München: Carl Hanser Verlag GmbH & Co. KG. https://doi.org/10.3139/9783446449381

Starke, G. & Hruschka, P. (2019a). *arc42 FAQ, Question A-2.* Zugriff am 19.02.2019. Verfügbar unter https://faq.arc42.org/questions/A-2/

Starke, G. & Hruschka, P. (2019b). *Download arc42.* Zugriff am 04.04.2019. Verfügbar unter https://arc42.org/download

Trelle, T. (2014). *MongoDB. Der praktische Einstieg*. Heidelberg: dpunkt.verlag.

Trelle, T. (Alkmene Verlags- und Mediengesellschaft mbH, Hrsg.). (2015). *Informatik Aktuell - MongoDB für Software-Entwickler.* Zugriff am 22.04.2019. Verfügbar unter https://www.informatik-aktuell.de/betrieb/datenbanken/mongodb-fuer-software-entwickler.html

Twilio Inc. (Hrsg.). (2019). *Twilio Products.* Zugriff am 10.05.2019. Verfügbar unter https://www.twilio.com/products

(VADIAN.NET AG, Hrsg.). (2019a). *aspsms.ch - Global Two-Way SMS,* VADIAN.NET AG, St. Gallen, Switzerland. Zugriff am 10.05.2019. Verfügbar unter https://www.aspsms.ch/two-way/

(VADIAN.NET AG, Hrsg.). (2019b). *aspsms.de - HTTP-Post - SMS,* VADIAN.NET AG, St. Gallen, Switzerland. Zugriff am 10.05.2019. Verfügbar unter https://www.aspsms.de/de/http/

WordPress Foundation (Hrsg.). (2019a, 18. Februar). *Pages « WordPress Codex.* Zugriff am 24.05.2019. Verfügbar unter https://codex.wordpress.org/Pages

WordPress Foundation (Hrsg.). (2019b, 13. März). *Class Reference/wpdb « WordPress Codex.* Zugriff am 24.05.2019. Verfügbar unter https://codex.wordpress.org/Class_Reference/wpdb

WordPress Foundation (Hrsg.). (2019c, 28. April). *Database Description « WordPress Codex.* Zugriff am 24.05.2019. Verfügbar unter https://codex.wordpress.org/Database_Description

Anhang

1 Laufzeitdiagramme

1.1 Externes System erzeugen

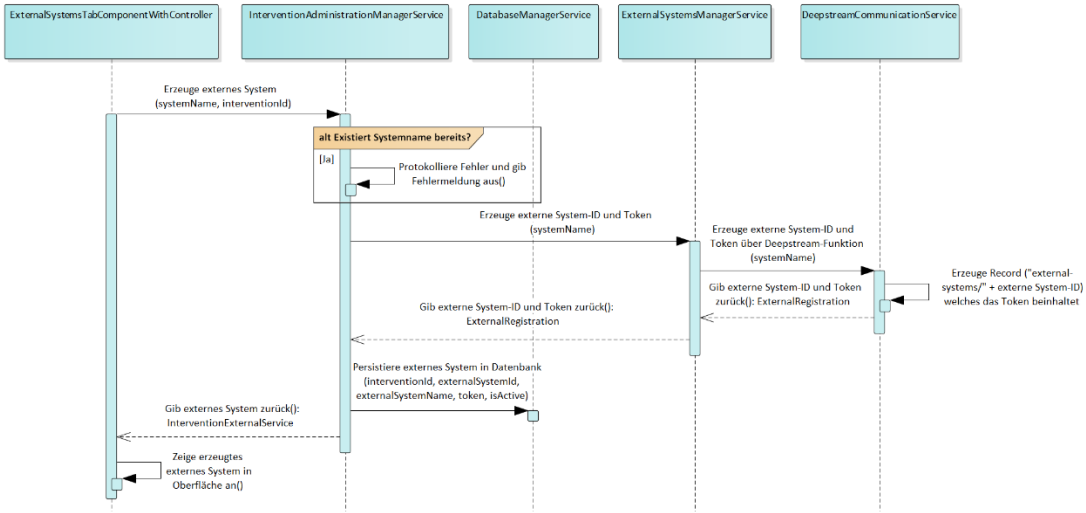


Abbildung 1 Sequenzdiagramm – Externes System erzeugen

1.2 Externes System umbenennen

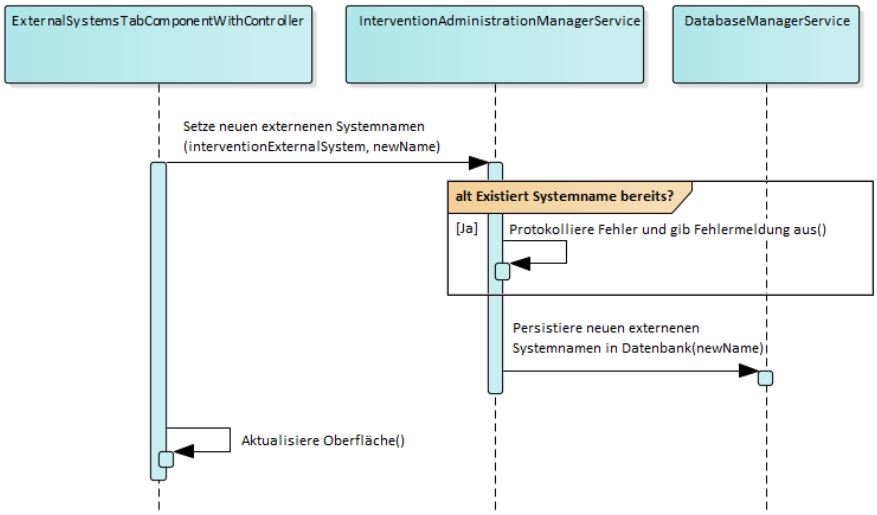


Abbildung 2 Sequenzdiagramm – Externes System umbenennen

1.3 Externes System löschen

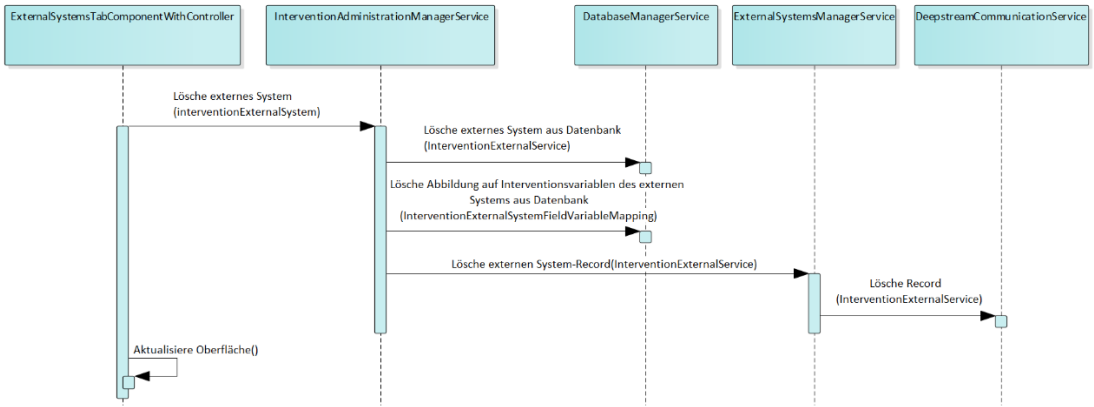


Abbildung 3 Sequenzdiagramm – Externes System löschen

1.4 Token erneuern

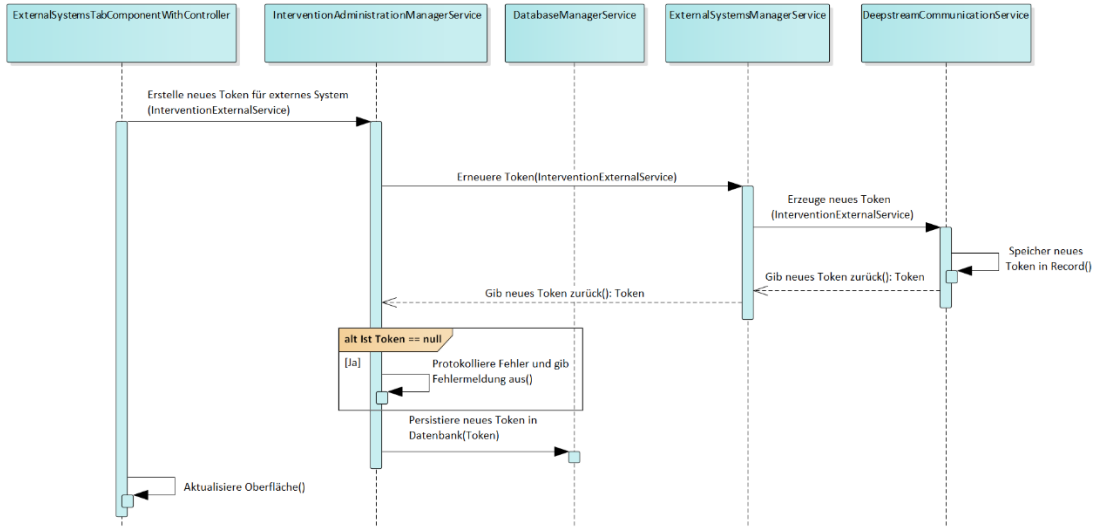


Abbildung 4 Sequenzdiagramm – Token erneuern

1.5 Externes System aktivieren/deaktivieren

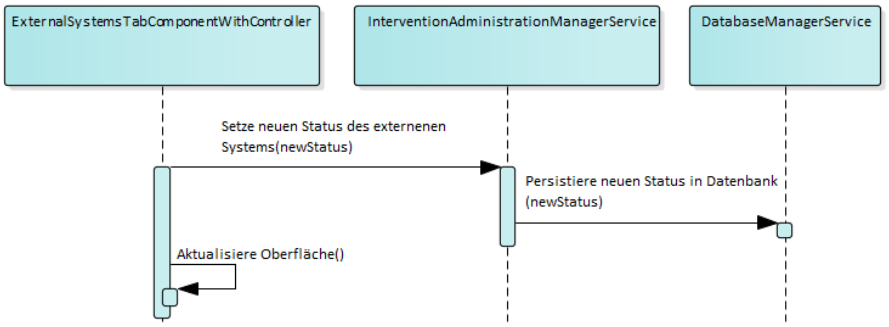


Abbildung 5 Sequenzdiagramm – Externes System aktivieren/deaktivieren

1.6 Abbildung erzeugen

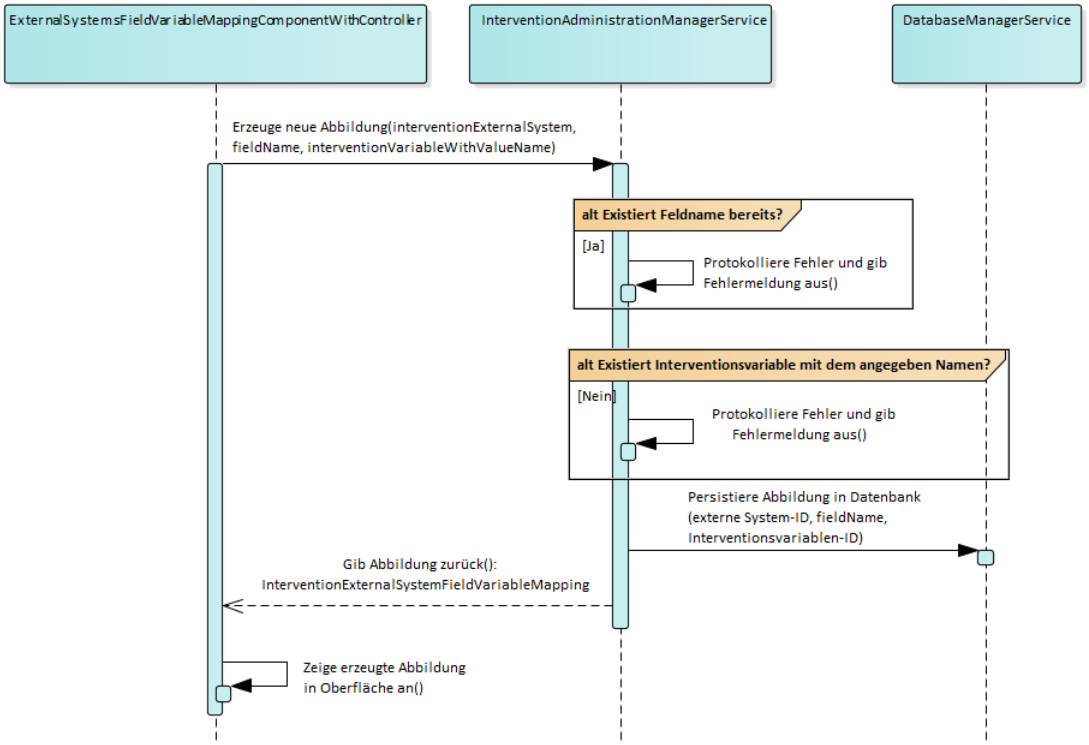


Abbildung 6 Sequenzdiagramm – Abbildung erzeugen

1.7 Abbildung bearbeiten

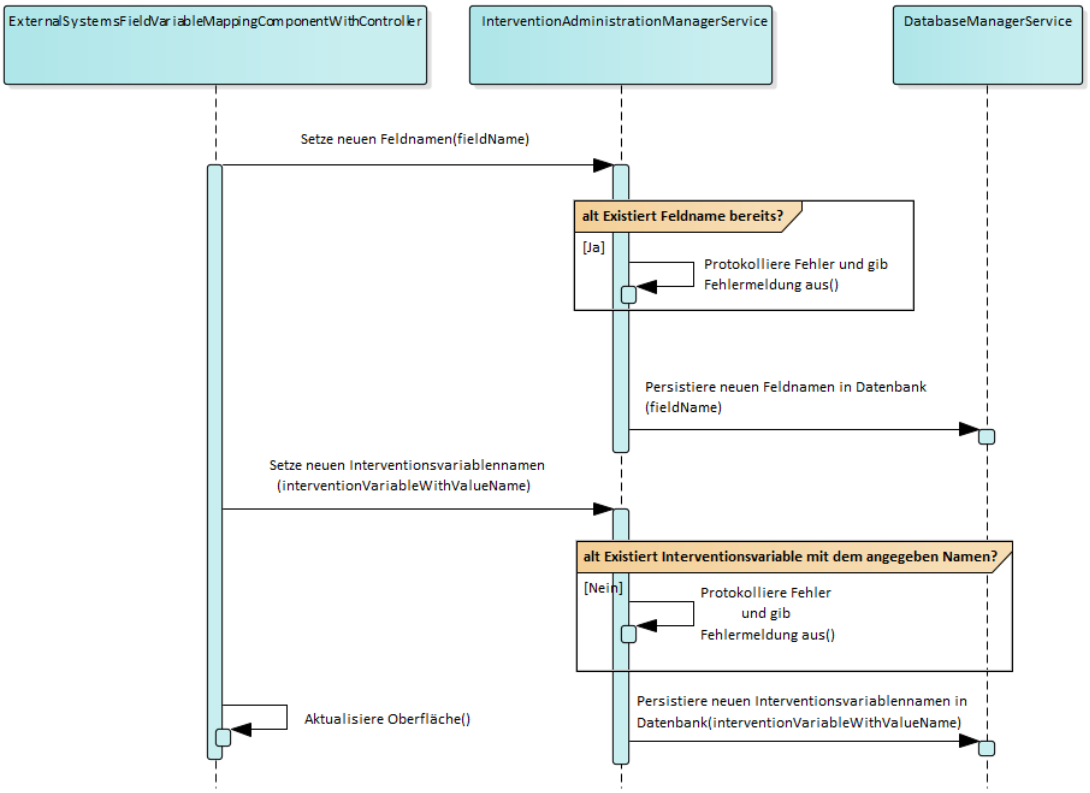


Abbildung 7 Sequenzdiagramm – Abbildung bearbeiten

1.8 Abbildung löschen

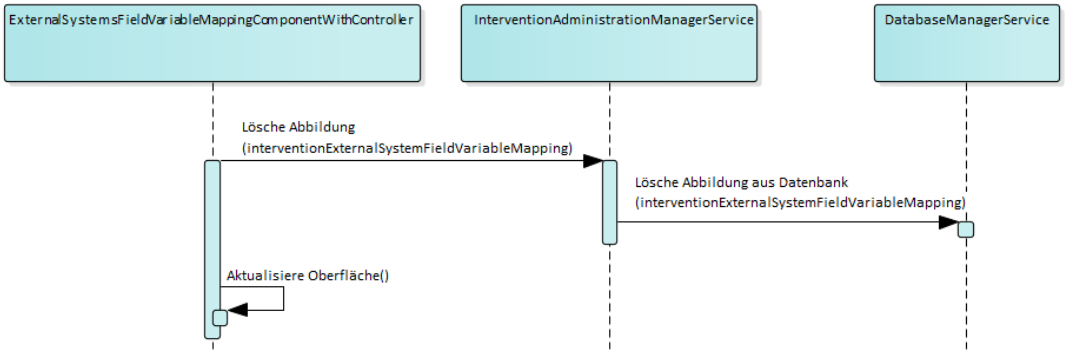


Abbildung 8 Sequenzdiagramm – Abbildung löschen

1.9 Authentifizierung eines externen Systems

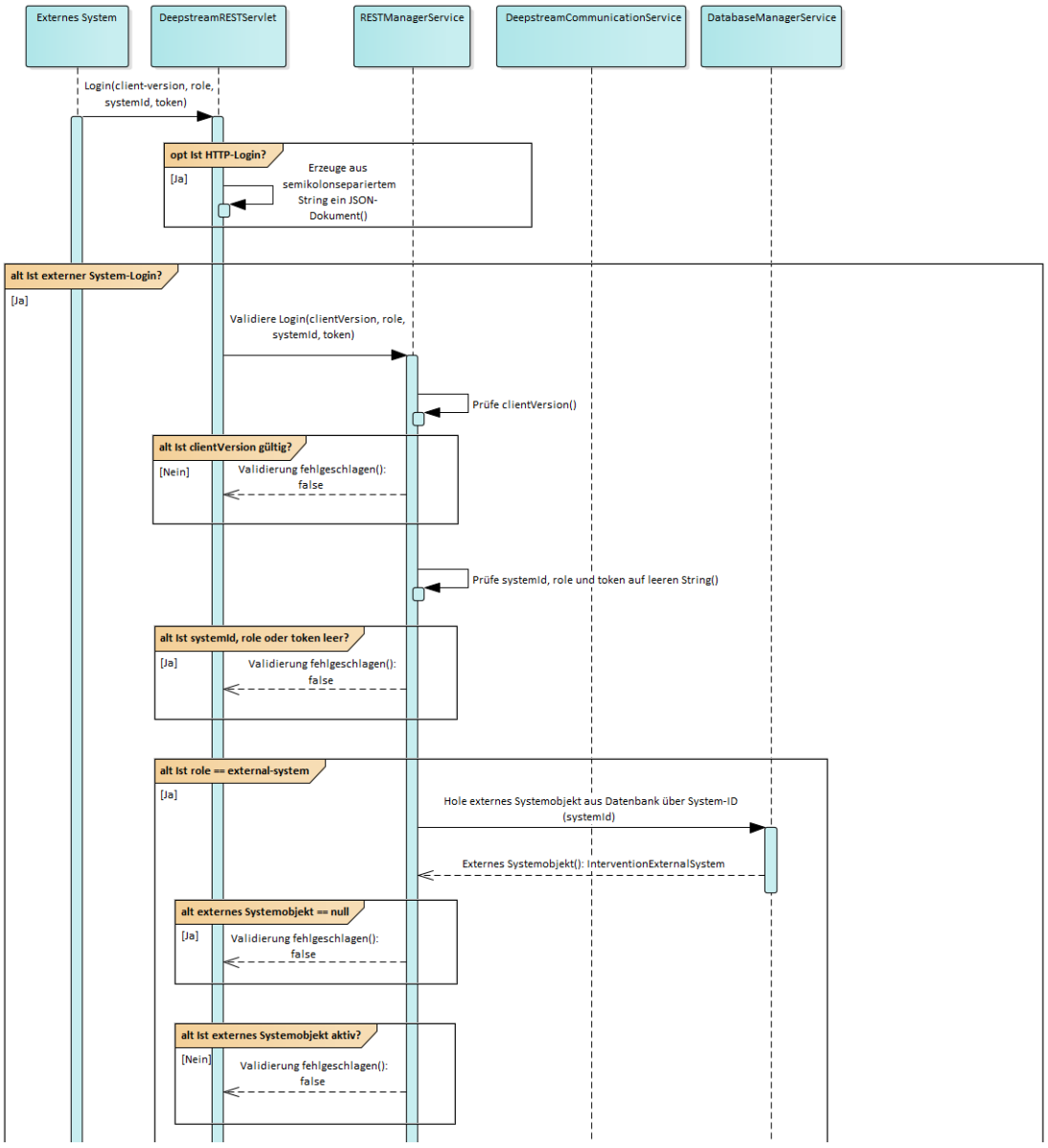


Abbildung 9 Sequenzdiagramm – Authentifizierung eines externen Systems – Teil 1

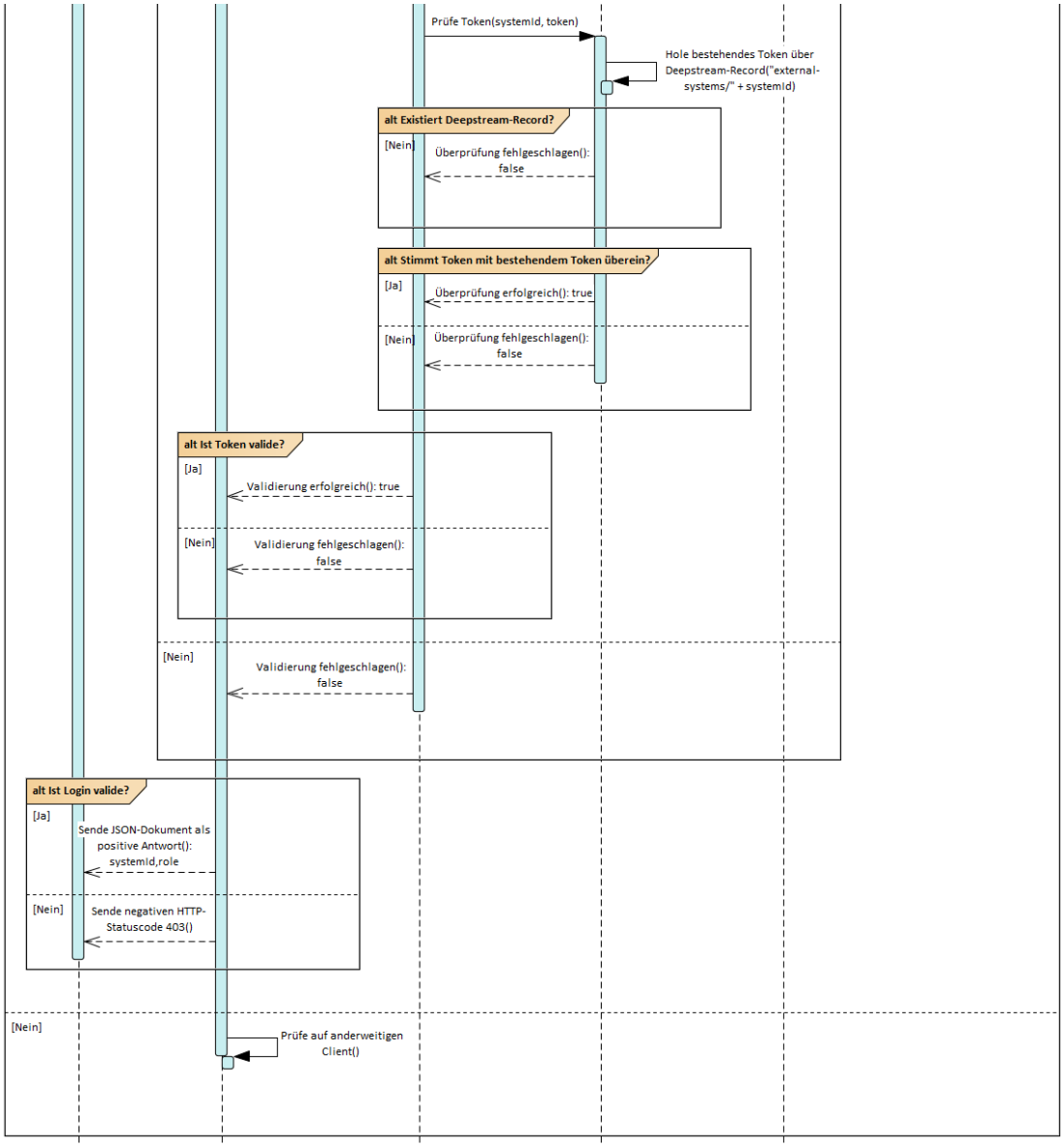


Abbildung 10 Sequenzdiagramm – Authentifizierung eines externen Systems – Teil 2

1.10 Abbildung einer ExternalSystemMessage auf ein ReceivedMessage-Objekt

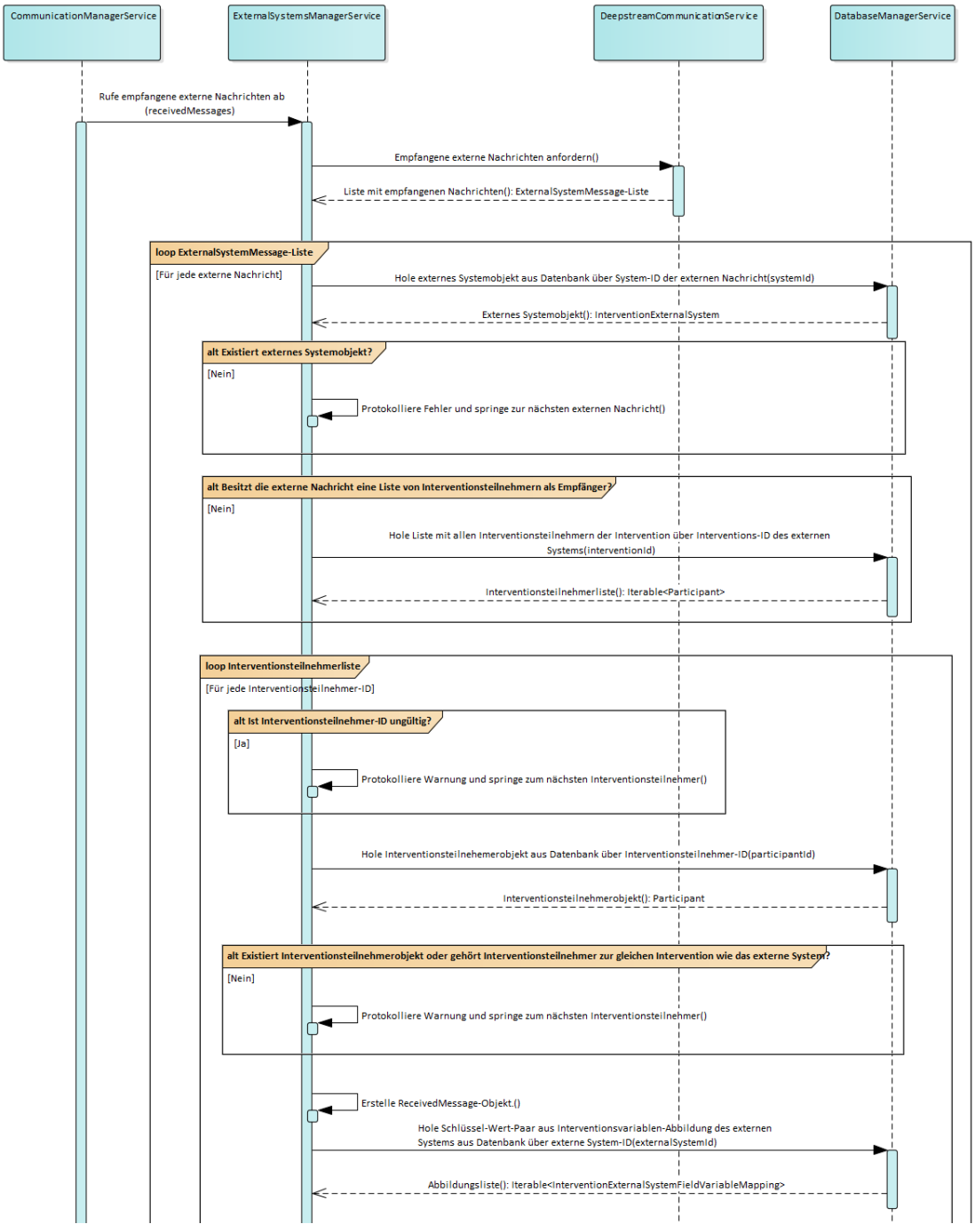


Abbildung 11 Sequenzdiagramm – Abbildung einer ExternalSystemMessage auf ein ReceivedMessage-Objekt – Teil 1

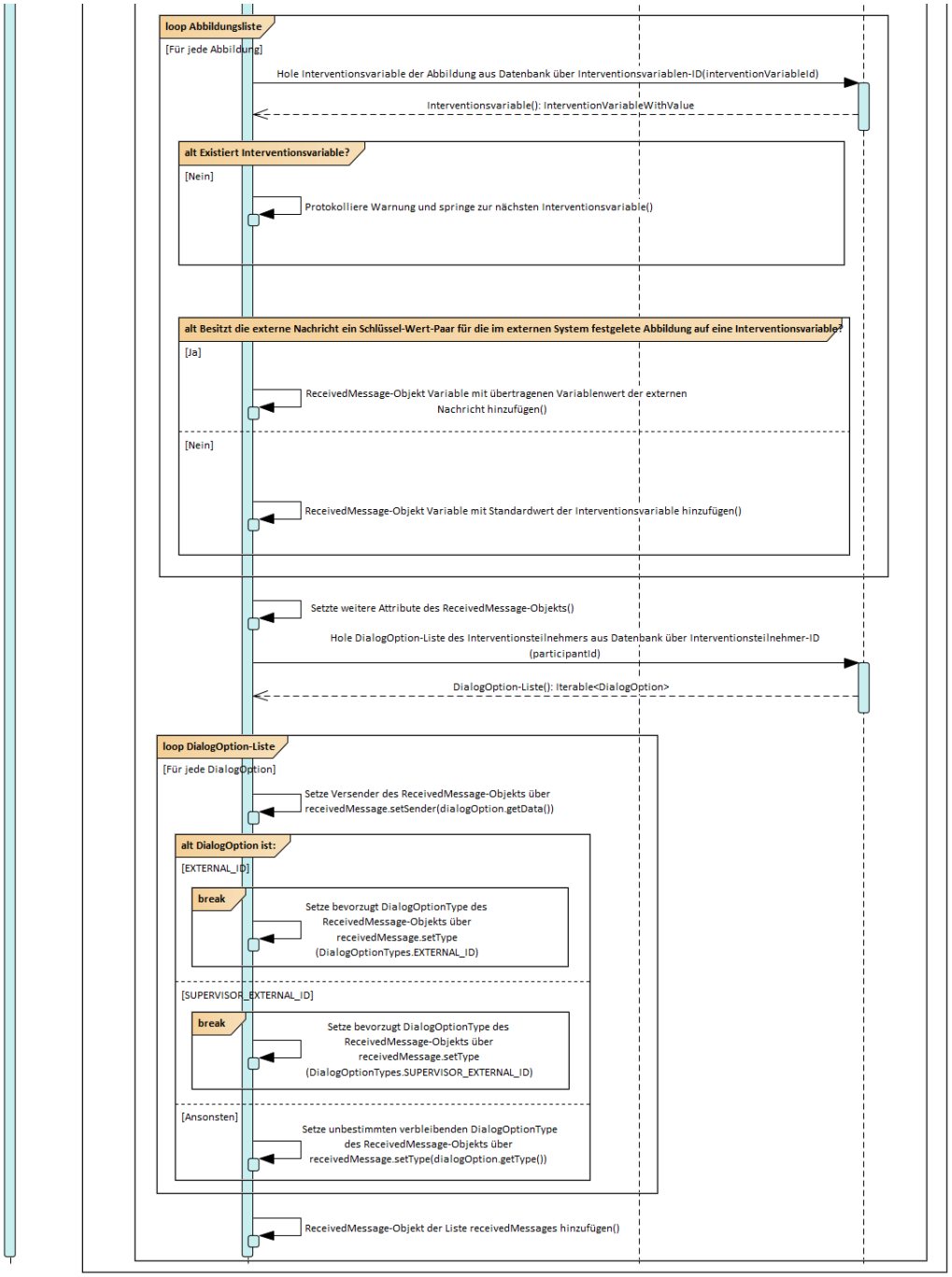


Abbildung 12 Sequenzdiagramm – Abbildung einer ExternalSystemMessage auf ein ReceivedMessage-Objekt – Teil 2

2 Mockups

2.1 Externe Systeme

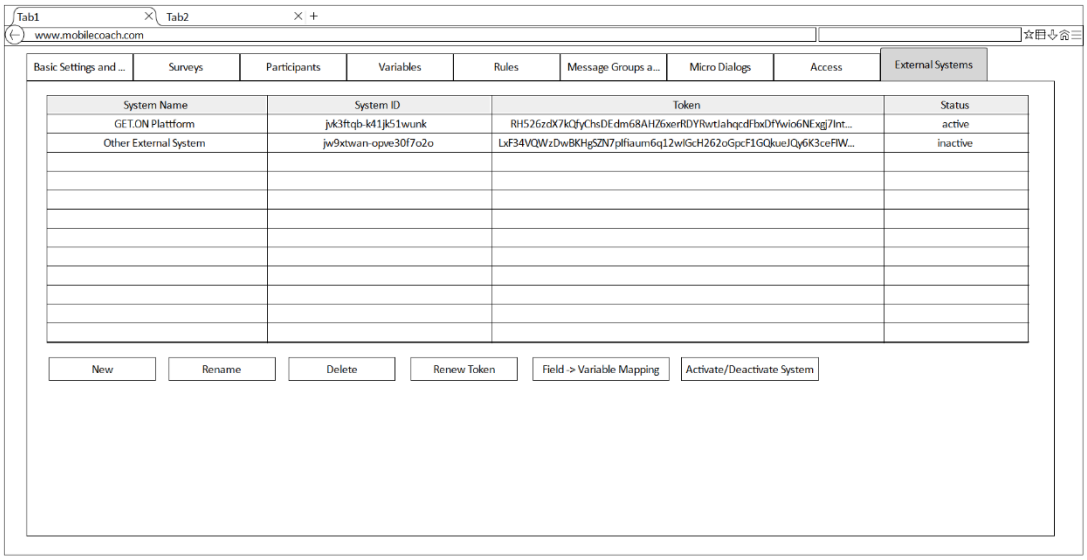


Abbildung 13 Mockup – Externe Systeme

2.2 Abbildung der Schlüssel-Wert-Paare auf Interventionsvariablen

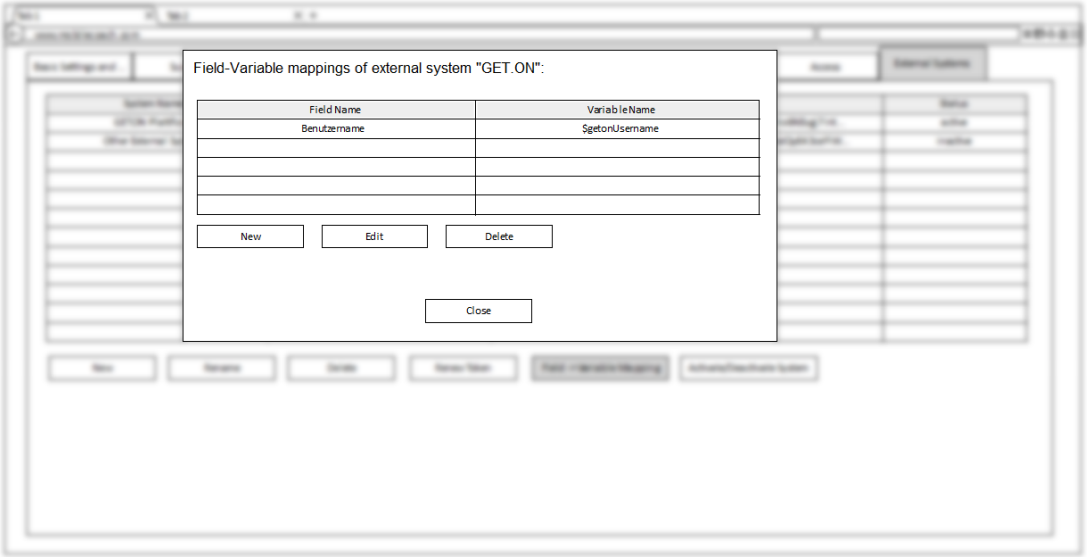


Abbildung 14 Mockup – Abbildung der Schlüssel-Wert-Paare auf Interventionsvariablen

2.3 Schlüsselname auf Interventionsvariable Zuordnungsdialog

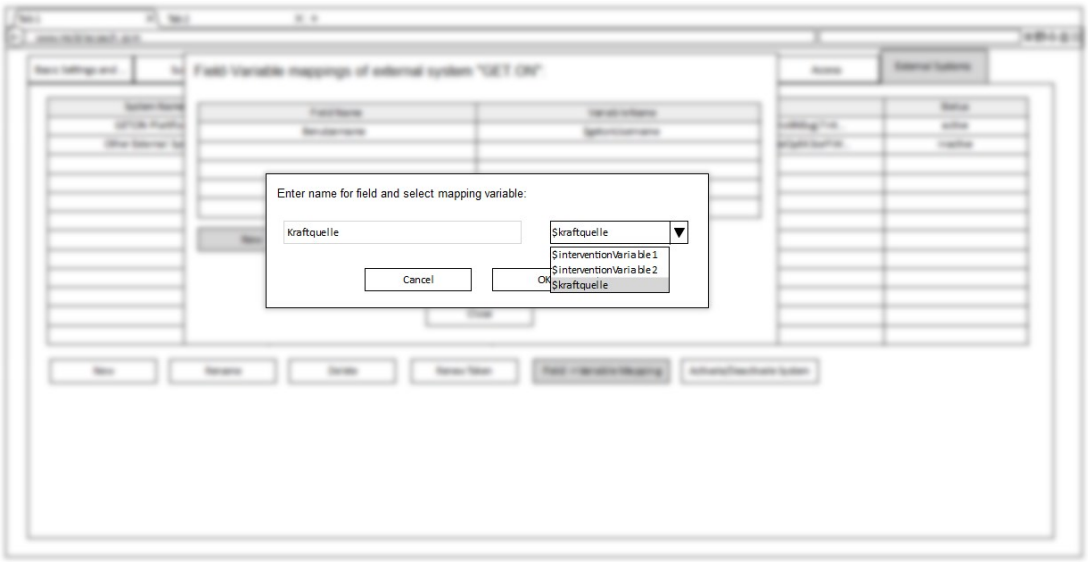


Abbildung 15 Mockup – Schlüsselname auf Interventionsvariable Zuordnungsdialog

2.4 Weboberfläche für die Verknüpfung mit der GET.ON-Plattform

Abbildung 16 Mockup – Weboberfläche für die Verknüpfung mit der GET.ON-Plattform

3 Screenshots

3.1 Externe Systeme

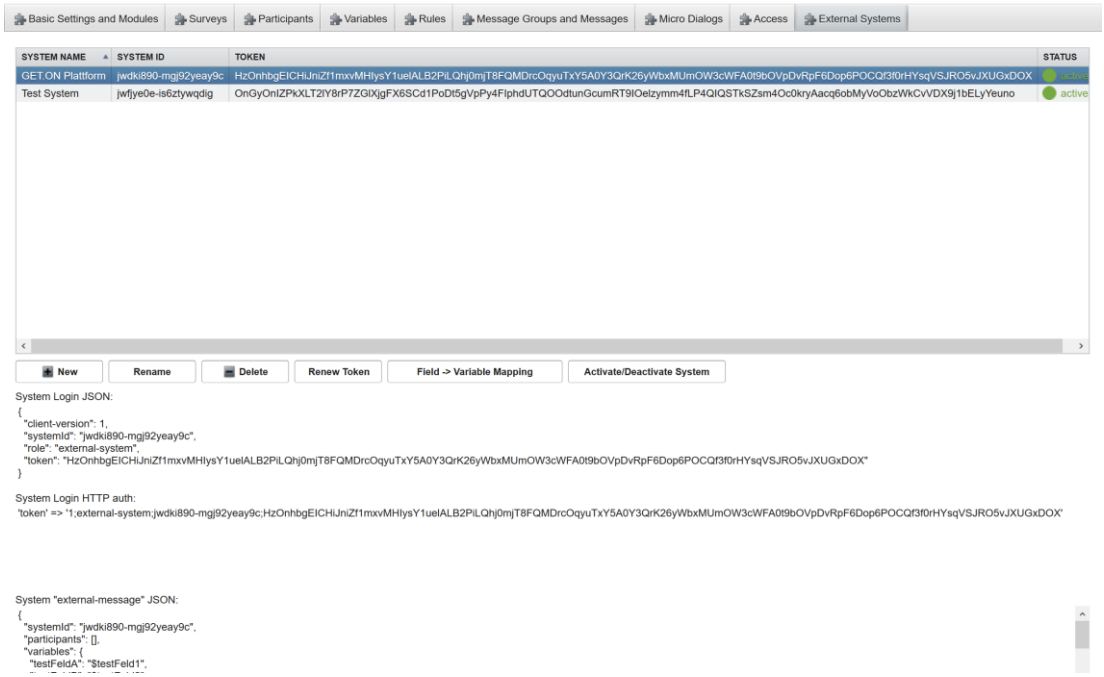


Abbildung 17 Screenshot – Externe Systeme

3.2 Abbildung der Schlüssel-Wert-Paare auf Interventionsvariablen

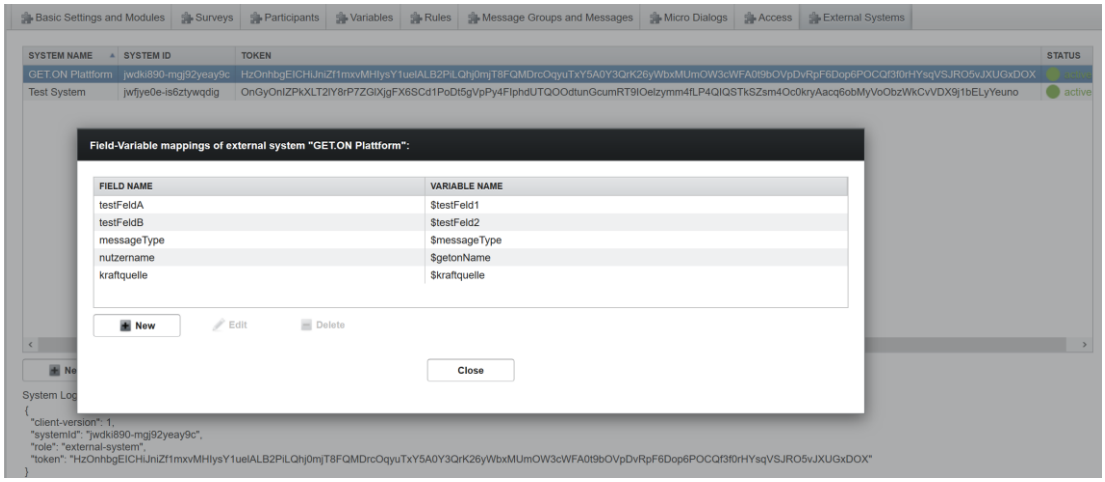


Abbildung 18 Screenshot – Abbildung der Schlüssel-Wert-Paare auf Interventionsvariablen

3.3 Schlüsselname auf Interventionsvariable Zuordnungsdialog

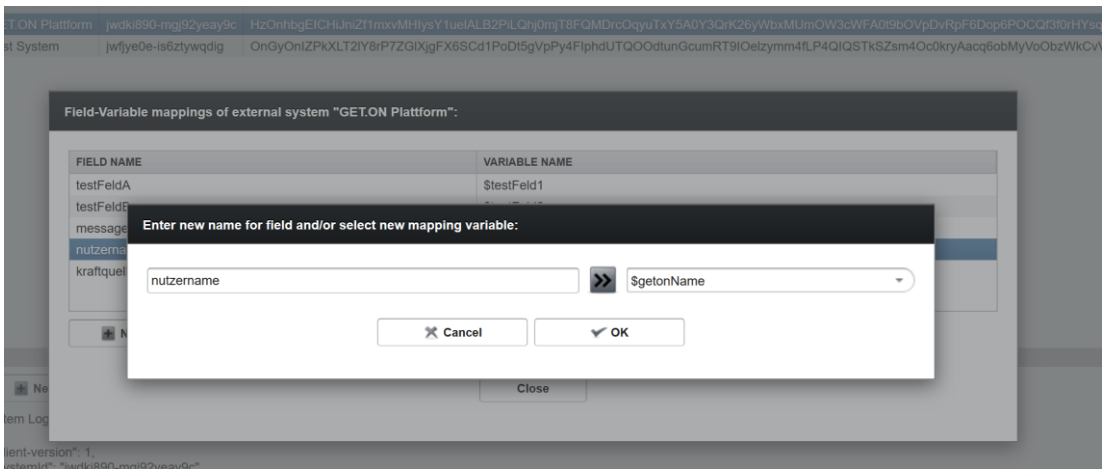


Abbildung 19 Screenshot – Schlüsselname auf Interventionsvariable Zuordnungsdialog

3.4 Weboberfläche für die Verknüpfung mit der GET.ON-Plattform

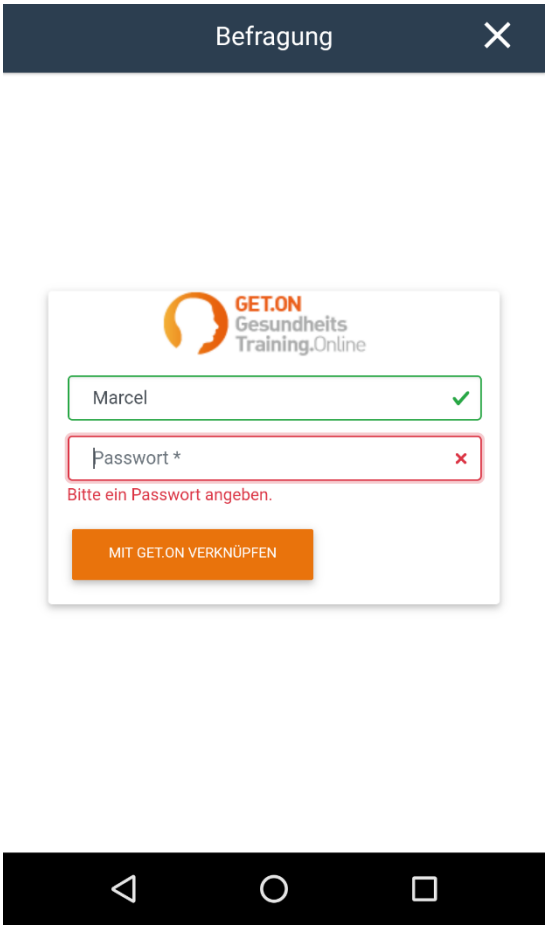


Abbildung 20 Screenshot – Weboberfläche für die Verknüpfung mit der GET.ON-Plattform

4 Quelltexte

Die nachfolgenden Quelltexte sind zur besseren Übersicht gekürzt.

4.1 Klasse – ExternalSystemsEditComponent

```
 1 /**
 2  * Provides the external systems edit component.
 3  *
 4  * @author Marcel Schlegel
 5  */
 6 @SuppressWarnings("serial")
 7 @Data
 8 @EqualsAndHashCode(callSuper = false)
 9 public class ExternalSystemsEditComponent extends AbstractCustomComponent {
10
11     /*- VaadinEditorProperties={"grid":"RegularGrid,20","showGrid":true,"snapToGrid":true,"snap-
ToObject":true,"movingGuides":false,"snappingDistance":10} */
12
13     @AutoGenerated
14     private VerticalLayout          mainLayout;
15     @AutoGenerated
16     private HorizontalLayout        buttonLayout;
17     @AutoGenerated
18     private VerticalLayout          textAreaLayout;
19     @AutoGenerated
20     private Button                  deleteButton;
21     @AutoGenerated
22     private Button                  renameButton;
23     @AutoGenerated
24     private Button                  newButton;
25     @AutoGenerated
26     private Button                  renewTokenButton;
27     @AutoGenerated
28     private Button                  activeInactiveButton;
29     @AutoGenerated
30     private Button                  fieldVariableMappingButton;
31     @AutoGenerated
32     private Table                   externalSystemsTable;
33     @AutoGenerated
34     private TextArea                systemLoginJsonTextArea;
35     @AutoGenerated
36     private TextArea                systemLoginHttpAuthTextArea;
37     @AutoGenerated
38     private TextArea                systemExternalMessageJsonTextArea;
…
39     // Create the main layout which shows the external system table.
40     @AutoGenerated
41     private VerticalLayout buildMainLayout() {
       …
42             return mainLayout;
43     }
44
45     // Create the button layout for create, update, delete, … an external system.
46     @AutoGenerated
47     private HorizontalLayout buildButtonLayout() {
       …
48             return buttonLayout;
49     }
50
51     // Build text areas for show the login parameters for a deepstream websocket or http login.
52     // Show external-message RPC parameters.
53     @AutoGenerated
54     private VerticalLayout buildTextAreaLayout() {
       …
55             return textAreaLayout;
56     }
57 }
```

Listing 1 Klasse – ExternalSystemsEditComponent

4.2 Klasse – ExternalSystemsTabComponent

```
/**
 * Provides the external systems tab component.
 *
 * @author Marcel Schlegel
 */
@SuppressWarnings("serial")
@Data
@EqualsAndHashCode(callSuper = false)
public abstract class ExternalSystemsTabComponent extends AbstractCustomComponent {

	/*- VaadinEditorProperties={"grid":"RegularGrid,20","showGrid":true,"snapToGrid":true,"snapToObject":true,"movingGuides":false,"snappingDistance":10} */

	@AutoGenerated
	private VerticalLayout 					mainLayout;
	@AutoGenerated
	private ExternalSystemsEditComponent 	externalSystemsEditComponent;

	/**
	 * The constructor should first build the main layout, set the composition
	 * root and then do any custom initialization.
	 *
	 * The constructor will not be automatically regenerated by the visual
	 * editor.
	 */
	protected ExternalSystemsTabComponent() {
		buildMainLayout();
		setCompositionRoot(mainLayout);

		// manually added
		// table options
		val externalSystemsTable = externalSystemsEditComponent.getExternalSystemsTable();
		externalSystemsTable.setSelectable(true);
		externalSystemsTable.setImmediate(true);
	}

	@AutoGenerated
	private VerticalLayout buildMainLayout() {
		// common part: create layout
		mainLayout = new VerticalLayout();
		mainLayout.setImmediate(false);
		mainLayout.setWidth("100%");
		mainLayout.setHeight("-1px");
		mainLayout.setMargin(false);

		// top-level component properties
		setWidth("100.0%");
		setHeight("-1px");

		// externalSystemsEditComponent
		// Create the external systems edit UI which shows all external systems
		// in a table view.
		externalSystemsEditComponent = new ExternalSystemsEditComponent();
		externalSystemsEditComponent.setImmediate(false);
		externalSystemsEditComponent.setWidth("100.0%");
		externalSystemsEditComponent.setHeight("-1px");
		mainLayout.addComponent(externalSystemsEditComponent);

		return mainLayout;
	}
}
```

Listing 2 Klasse – ExternalSystemsTabComponent

4.3 Klasse – ExternalSystemsTabComponentWithController

```
1  /**
2   * Extends the external systems tab component with a controller.
3   *
4   * @author Marcel Schlegel
5   */
6  @SuppressWarnings("serial")
7  @Log4j2
8  public class ExternalSystemsTabComponentWithController extends ExternalSystemsTabComponent {
9
10     private final Intervention                      intervention;
11     private UIInterventionExternalSystem            selectedUIExternalSystem               = null;
12     private BeanItem<UIInterventionExternalSystem>  selectedUIExternalSystemBeanItem       = null;
13     private final BeanContainer<ObjectId, UIInterventionExternalSystem>     beanContainer;
14
…
15
16     private void fillLoginJsonTextArea(InterventionExternalSystem externalSystem) {
17             val jsonObject = new JsonObject();
               …
18             // Build JSON websocket login parameters.
               …
19             getExternalSystemsEditComponent().setLoginJsonTextAreaContent(jsonObject);
20     }
21
22     // Build JSON http login parameters.
23     private void fillLoginHttpAuthTextArea(InterventionExternalSystem externalSystem) {
24             getExternalSystemsEditComponent().setLoginHttpAuthTextAreaContent(
25                             Constants.getDeepstreamMaxClientVersion(),
26                             ImplementationConstants.DEEPSTREAM_EXTERNAL_SYSTEM_ROLE,
27                             externalSystem.getSystemId(), externalSystem.getToken());
28     }
29
30     private void fillExternalMessageJsonTextArea(InterventionExternalSystem externalSystem) {
31             val jsonObject = new JsonObject();
               …
32             // Build JSON external-message parameters.
               …
33             getExternalSystemsEditComponent().setExternalMessageJsonTextAreaContent(jsonObject);
34     }
35
36     private class ButtonClickListener implements Button.ClickListener {
37             @Override
38             public void buttonClick(final ClickEvent event) {
39                     val accessControlEditComponent = getExternalSystemsEditComponent();
40
41                     // If create external system button is clicked call the creation function.
42                     if (event.getButton() == accessControlEditComponent.getNewButton()) {
43                             createExternalSystem();
44                     }
                       …
45             }
46     }
47
48     // Refresh text areas if external system tab is entered.
49     public void adjust() {
50             if(selectedUIExternalSystem == null) {
51                     return;
52             }
53
54             Log.debug("Refresh JSON text areas for external system");
55
56             val selectedExternalSystem = selectedUIExternalSystem
57                             .getRelatedModelObject(InterventionExternalSystem.class);
58
59             fillLoginJsonTextArea(selectedExternalSystem);
60             fillLoginHttpAuthTextArea(selectedExternalSystem);
61             fillExternalMessageJsonTextArea(selectedExternalSystem);
62     }
```

Listing 3 Klasse – ExternalSystemsTabComponentWithController – Teil 1

```
 63     public void createExternalSystem() {
 64             log.debug("Create external system");
 65             // Show modal window for setting the external system name.
 66             showModalStringValueEditWindow(
 67                     AdminMessageStrings.ABSTRACT_STRING_EDITOR_WINDOW__ENTER_NAME_FOR_EXTERNAL_SYSTEM,
null, null, new ShortStringEditComponent(), new ExtendableButtonClickListener() {
 68     @Override
 69     public void buttonClick(final ClickEvent event) {
 70     InterventionExternalSystem newExternalSystem;
 71     try {
 72     // Create new external system. Call the façade method in InterventionAdministrationManagerService.
 73     newExternalSystem = getInterventionAdministrationManagerService()
 74                     .interventionExternalSystemCreate(getStringValue(),intervention.getId());
 75     } catch (final Exception e) {
 76             handleException(e);
 77             return;
 78     }
 79
 80     // Adapt UI
 81     // Add new external system entity to table. This is done over the UIModelObject.
 82     beanContainer.addItem(newExternalSystem.getId(),UIInterventionExternalSystem.class
 83                                                     .cast(newExternalSystem.toUIModelObject()));
 84     getExternalSystemsEditComponent().getExternalSystemsTable().select(newExternalSystem.getId());
 85     val extrenalSystemsTable = getExternalSystemsEditComponent().getExternalSystemsTable();
 86     extrenalSystemsTable.sort();
 87     extrenalSystemsTable.select(null);
 88     extrenalSystemsTable.select(newExternalSystem.getId());
 89
 90     // Create successful creation message and close the modal window.
 91     getAdminUI().showInformationNotification(AdminMessageStrings.NOTIFICATION__EXTERNAL_SYSTEM_CRE-
ATED);
 92     closeWindow();
 93     }
 94     }, null);
 95 }
 96
 97     public void renameExternalSystem() {
 98             // Rename external system and call façade method like in the creation method.
 99             log.debug("Rename external system");
       …
100     }
101
102     public void deleteExternalSystem() {
103             // Delete external system and call façade method like in the creation method.
104             log.debug("Delete external system");
       …
105     }
106
107     public void activeInactiveExternalSystem() {
108             // Activate/Deactivate external system and call façade method like in the creation method.
109             log.debug("Active/Inactive system");
       …
110     }
111
112     public void openFieldVariableMappings() {
113             // Open the mapping window for mapping key-value pairs on intervention variables.
                …
114             log.debug("Open field variable mappings of external system {}", selectedExternalSys-
tem.getSystemId());
                …
115     }
116
117 }
```

Listing 4 Klasse – ExternalSystemsTabComponentWithController – Teil 2

4.4 Klasse – ShortStringEditWithComboBoxComponent

```
/**
 * Provides a short string edit {@link TextField} with a {@link ComboBox} for variables window.
 *
 * @author Marcel Schlegel
 */
@SuppressWarnings("serial")
@Data
@EqualsAndHashCode(callSuper = false)
public class ShortStringEditWithComboBoxComponent extends AbstractStringValueEditComponent {

    /*- VaadinEditorProperties={"grid":"RegularGrid,20","showGrid":true,"snapToGrid":true,"snapToObject":true,"movingGuides":false,"snappingDistance":10} */

    @AutoGenerated
    private VerticalLayout          mainLayout;
    @AutoGenerated
    private GridLayout              buttonLayout;
    @AutoGenerated
    private Button                  okButton;
    @AutoGenerated
    private Button                  cancelButton;
    @AutoGenerated
    private TextField               stringTextField;
    @AutoGenerated
    private Embedded                arrowRightIcon;
    @AutoGenerated
    private HorizontalLayout        editAreaLayout;
    @AutoGenerated
    private ComboBox                variableComboBox;
…
    @Override
    public void setStringValue(final String value) {
            // Set the key (field) name of the key-value pair.
            stringTextField.setValue(value);
    }

    @Override
    public String getStringValue() {
            return stringTextField.getValue();
    }

    @Override
    public void addVariables(List<String> variables) {
            // Add the possible selection of intervention variable names for the ComboBox.
            variableComboBox.addItems(variables);
    }

    @Override
    public void registerOkButtonListener(final ClickListener clickListener) {
            okButton.addClickListener(clickListener);
    }

    @Override
    public void registerCancelButtonListener(
                    final ClickListener clickListener) {
            cancelButton.addClickListener(clickListener);
    }

    public String getSelectedVariable() {
            return Objects.toString(variableComboBox.getValue(), null);
    }

    public void setSelectedVariable(String variableName) {
            // Set a variable name which should be selected from the ComboBox values.
            variableComboBox.setValue(variableName);
    }
// Build layouts.
…

}
```

Listing 5 Klasse – ShortStringEditWithComboBoxComponent

4.5 Klasse – ExternalSystemsFieldVariableMappingComponent

```
/**
 * Provides the external system variable mapping component.
 *
 * @author Marcel Schlegel
 */
@SuppressWarnings("serial")
@Data
@EqualsAndHashCode(callSuper = false)
public class ExternalSystemsFieldVariableMappingComponent extends AbstractClosableEditComponent{

    /*- VaadinEditorProperties={"grid":"RegularGrid,20","showGrid":true,"snapToGrid":true,"snapToObject":true,"movingGuides":false,"snappingDistance":10} */

    @AutoGenerated
    private VerticalLayout          mainLayout;
    @AutoGenerated
    private VerticalLayout          closeButtonLayout;
    @AutoGenerated
    private Button                  closeButton;
    @AutoGenerated
    private HorizontalLayout        mappingButtonLayout;
    @AutoGenerated
    private Button                  newButton;
    @AutoGenerated
    private Button                  editButton;
    @AutoGenerated
    private Button                  deleteButton;
    @AutoGenerated
    private Table                   mappingTable;
…
    @Override
    public void registerOkButtonListener(final ClickListener clickListener) {
            closeButton.addClickListener(clickListener);
    }
    @Override
    public void registerCancelButtonListener(final ClickListener clickListener) {
            // Nothing required
    }
    @AutoGenerated
    private VerticalLayout buildMainLayout() {
            // common part: create layout
            mainLayout = new VerticalLayout();
            mainLayout.setImmediate(false);
            mainLayout.setWidth("900px");
            mainLayout.setHeight("-1px");
            mainLayout.setMargin(true);
            mainLayout.setSpacing(true);

            // top-level component properties
            setWidth("900px");
            setHeight("-1px");

            // mappingTable
            mappingTable = new Table();
            mappingTable.setImmediate(false);
            mappingTable.setWidth("100.0%");
            mappingTable.setHeight("150px");
            mainLayout.addComponent(mappingTable);

            // mappingButtonLayout
            mappingButtonLayout = buildButtonLayout();
            mainLayout.addComponent(mappingButtonLayout);

            // closeButtonLayout
            closeButtonLayout = buildCloseButtonLayout();
            mainLayout.addComponent(closeButtonLayout);
            mainLayout.setComponentAlignment(closeButtonLayout, new Alignment(48));

            return mainLayout;
    }
```

Listing 6 Klasse – ExternalSystemsFieldVariableMappingComponent

4.6 Klasse – ExternalSystemsFieldVariableMappingComponentWithController

```
1 /**
2  * Extends the external system variable mapping component with a controller
3  *
4  * @author Marcel Schlegel
5  */
6 @SuppressWarnings("serial")
7 @Log4j2
8 public class ExternalSystemsFieldVariableMappingComponentWithController extends
9                                                         ExternalSystemsFieldVariableMappingComponent {
10
11      private final InterventionExternalSystem        interventionExternalSystem;
12      private UIInterventionExternalSystemFieldVariableMapping
13                                                        selectedUIExternalSystemMapping = null;
14      private BeanItem<UIInterventionExternalSystemFieldVariableMapping>
15                                                        selectedUIExternalSystemMappingBeanItem = null;
16      private final BeanContainer<ObjectId, UIInterventionExternalSystemFieldVariableMapping>
17                                                        beanContainer;
…
18      private class ButtonClickListener implements Button.ClickListener {
19              @Override
20              public void buttonClick(final ClickEvent event) {
21
22                      if (event.getButton() == getNewButton()) {
23                              createExternalSystemFieldVariableMapping();
24                      }
                        …
25              }
26      }
27
28      public void createExternalSystemFieldVariableMapping() {
29              log.debug("Create external system field variable mapping");
30      // Show modal window for setting the mapping. Using the ShortStringEditWithComboBoxComponent.
31              val shortStringEditWithComboBoxComponent = new ShortStringEditWithComboBoxComponent();
32              showModalStringValueEditWindow(AdminMessageStrings.ABSTRACT_STRING_EDITOR_WINDOW__EN-
TER_NAME_AND_SELECT_VARIABLE_FOR_SYSTEM_MAPPING, null,
33                              getInterventionAdministrationManagerService()
34                                      .getAllInterventionVariablesManageableByServiceOrLessRestric-
tive(interventionExternalSystem.getIntervention()), shortStringEditWithComboBoxComponent,
35                              new ExtendableButtonClickListener() {
36      @Override
37      public void buttonClick(final ClickEvent event) {
38      InterventionExternalSystemFieldVariableMapping newExternalSystemFieldVariableMapping;
39      try {
40      // Create new mapping. Call the façade method in InterventionAdministrationManagerService.
41      val selectedVariable = shortStringEditWithComboBoxComponent.getSelectedVariable();
42
43      newExternalSystemFieldVariableMapping = getInterventionAdministrationManagerService()
44      .interventionExternalSystemFieldVariableMappingCreate(interventionExternalSystem,getStringValue(),
45      selectedVariable);
46      } catch (final Exception e) {
47              handleException(e);
48              return;
49      }
50
51      // Adapt UI
52      // Add new external system variable mapping entity to table. This is done over the UIModelObject.
53      beanContainer.addItem(newExternalSystemFieldVariableMapping.getId(),
54      UIInterventionExternalSystemFieldVariableMapping.class
55              .cast(newExternalSystemFieldVariableMapping.toUIModelObject()));
56      getMappingTable().select(newExternalSystemFieldVariableMapping.getId());
57      // Create successful creation message and close the modal window.
58      getAdminUI().showInformationNotification(AdminMessageStrings.NOTIFICATION__EXTERNAL_SYSTEM_MAP-
PING_CREATED);
59      closeWindow();
60      }
61      }, null);
62      }
63 // Edit and delete methods
…
64 }
```

Listing 7 Klasse – ExternalSystemsFieldVariableMappingComponentWithController

4.7 Klasse – DeepstreamCommunicationService

```
@Log4j2
public class DeepstreamCommunicationService extends Thread
             implements PresenceEventListener, DeepstreamRuntimeErrorHandler,
             ConnectionStateListener {
…
public ExternalRegistration registerExternalSystem(final String externalSystemName) {

     log.debug("Trying to register external system for {}", externalSystemName);

     Record record = null;
     String externalSystemId;
     String token;
synchronized (client) {
     try {
     // Generate token and UID.
     token = RandomStringUtils.randomAlphanumeric(128);
     externalSystemId = client.getUid();
     // Write token and role to record with path: "external-systems/[systemId]"
     record = client.record.getRecord(DeepstreamConstants.PATH_EXTERNAL_SYSTEMS + externalSystemId);
     record.set(DeepstreamConstants.TOKEN, token);
     record.set(DeepstreamConstants.ROLE, externalSystemRole);
     } finally {
             // If something goes wrong discard record.
             if (record != null) {
                     try {
                             record.discard();
                     } catch (final Exception e) {
                             log.warn("Could not discard record on external system registration");
                     }
             }
             }
     }
     log.debug("External system registered for {}", externalSystemName);

     return new ExternalRegistration(externalSystemId, token);
     }
…
}
```

Listing 8 Erstelle externe System-UID und Token im DeepstreamCommunicationService

```
@Log4j2
public class DeepstreamCommunicationService extends Thread
        implements PresenceEventListener, DeepstreamRuntimeErrorHandler,
        ConnectionStateListener {
…

    /**
     * Provide RPC methods
     */
    private void provideMethods() {
    …
    // Provide external-message RPC method.
    client.rpc.provide(DeepstreamConstants.RPC_EXTERNAL_MESSAGE, (rpcName, data, rpcResponse) -> {
                final JsonObject jsonData = (JsonObject) gson.toJsonTree(data);

    try {
    // Collect all method parameters.
    val systemId = jsonData.get(DeepstreamConstants.REST_FIELD_SYSTEM_ID).getAsString();
    val jsonVariables = jsonData.get(DeepstreamConstants.REST_FIELD_VARIABLES).getAsJsonObject();
    List<String> participants = new ArrayList<>();
    if (jsonData.has(DeepstreamConstants.REST_FIELD_PARTICIPANTS)) {
            val participantJsonArray = jsonData.get(DeepstreamConstants.REST_FIELD_PARTICIPANTS)
                                                .getAsJsonArray();
            participantJsonArray.forEach(jsonElement -> participants.add(jsonElement.getAsString()));
    }
    // Collect all key-value variable pairs.
    Map<String, Variable> variables = new HashMap<>();
    for (Map.Entry<String, JsonElement> entry : jsonVariables.entrySet()) {
            String name = entry.getKey();
            JsonElement element = entry.getValue();

            variables.put(name, new Variable(name, element.getAsString()));
    }
    // Create a ExternalSystemMessage and add it to the received messages.
    final boolean receivedSuccessful = receiveExternalSystemMessage(systemId, participants,
                                                                    variables);

    if (receivedSuccessful) {
            rpcResponse.send(new JsonPrimitive(true));
    } else {
            rpcResponse.send(new JsonPrimitive(false));
    }
    } catch (final Exception e) {
            log.warn("Error when receiving external system message: {}", e.getMessage());
            rpcResponse.send(new JsonPrimitive(false));
    }
    });
    …
    }
…
}
```

Listing 9 External-message-RPC-Methode im DeepstreamCommunicationService

4.8 Klasse – SystemVariables

```
 1 public class SystemVariables {
…
 2      public enum READ_ONLY_EXTERNAL_SYSTEM_VARIABLES {
 3              externalSystemName,
 4              externalSystemId;
 5
 6              public String toVariableName() {
 7                      return ImplementationConstants.VARIABLE_PREFIX + toString();
 8              }
 9      }
…
10 }
```

Listing 10 Erstelle Systemvariablen für externe Systeme

4.9 Klasse – VariablesManagerService

```
 1 @Log4j2
 2 public class VariablesManagerService {
…
 3 public Hashtable<String, AbstractVariableWithValue> getAllVariablesWithValuesOfParticipantAndSys-
temAndExternalSystem(final Participant participant,
 4      final MonitoringMessage relatedMonitoringMessage,
 5      final MicroDialogMessage relatedMicroDialogMessage,
 6      final InterventionExternalSystem externalSystem) {
 7
 8              val variablesWithValues = new Hashtable<String, AbstractVariableWithValue>();
…
 9              // Add all read only external system variables.
10              for (val variable : SystemVariables.READ_ONLY_EXTERNAL_SYSTEM_VARIABLES
11                              .values()) {
12                      val readOnlyExternalSystemVariableValue = getReadOnlyExternalSystemVariableValue(
13                                      externalSystem, variable);
14
15                      if (readOnlyExternalSystemVariableValue != null) {
16                              addToHashtable(variablesWithValues, variable.toVariableName(),
17                                              readOnlyExternalSystemVariableValue);
18                      }
19              }
20
21              return variablesWithValues;
22      }
…
23
24 // Get the variable value by the given externalSystem.
25 private String getReadOnlyExternalSystemVariableValue(final InterventionExternalSystem externalSystem,
26      final READ_ONLY_EXTERNAL_SYSTEM_VARIABLES variable) {
27
28              if (externalSystem != null) {
29                      switch (variable) {
30                      case externalSystemId:
31                              return externalSystem.getSystemId();
32                      case externalSystemName:
33                              return externalSystem.getName();
34                      }
35              }
36              return null;
37      }
…
38 }
```

Listing 11 Erzeuge Werte für Systemvariablen eines externen Systems

4.10 Klasse – RecursiveAbstractMonitoringRulesResolver

```
@Log4j2
public class RecursiveAbstractMonitoringRulesResolver {
    public static enum EXECUTION_CASE {
            MONITORING_RULES_DAILY,
            MONITORING_RULES_PERIODIC,
            MONITORING_RULES_UNEXPECTED_MESSAGE,
            MONITORING_RULES_EXTERNAL_MESSAGE,
            MONITORING_RULES_USER_INTENTION,
            MONITORING_REPLY_RULES,
            MICRO_DIALOG_DECISION_POINT
    }
…
/**
 * Recursively walks through rules until one rule matches
 *
 * @param parent
 * @throws Exception
 */
private void executeRules(final AbstractMonitoringRule parent) {
    // Start with the whole process
    Iterable<? extends AbstractMonitoringRule> rulesOnCurrentLevel = null;
    if (parent == null) {
            // Root of rules tree
            if (ONE_OF_MONITORING_RULES_CASES) {
                    // Handle monitoring rules separately due to complexity
                    MonitoringRule masterParent = null;

                    switch (executionCase) {
                            …
                            case MONITORING_RULES_EXTERNAL_MESSAGE:
                                    masterParent = databaseManagerService
                                            .findOneModelObject(MonitoringRule.class,
                                                    Queries.MONITORING_RULE__BY_INTERVENTION_AND_TYPE,
                                                    intervention.getId(),
                                                    MonitoringRuleTypes.EXTERNAL_MESSAGE);
                                            break;
                            …
                            }
    …
    }
    …
}
}
```

Listing 12 Verarbeite Interventionsregel „External Message“

4.11 Klasse – DeepstreamRESTServlet

```
@Log4j2
public class DeepstreamRESTServlet extends HttpServlet {
…
protected void doPost(final HttpServletRequest request,
            final HttpServletResponse response)
            throws ServletException, IOException {
…
    val stringPayload = IOUtils.toString(request.getReader());
…
    final JsonElement jsonElement = gson.fromJson(stringPayload, JsonElement.class);
    final JsonObject jsonObjectPayload = jsonElement.getAsJsonObject();

    // Get auth data from request.
    val authData = (JsonObject) jsonObjectPayload.get(DeepstreamConstants.DS_FIELD_AUTH_DATA);

    // Check for HTTP login.
    if (!isWebSocketConnection(jsonObjectPayload)) {
            // Create from HTTP token a JSON document.
            authData = httpAuthTokenToJson(authData.get(DeepstreamConstants.REST_FIELD_TOKEN)
                                            .getAsString());
    }

    // Get deepstream client version.
    val clientVersion = authData.get(DeepstreamConstants.REST_FIELD_CLIENT_VERSION).getAsInt();

    // Get deepstream role.
    val role = authData.get(DeepstreamConstants.REST_FIELD_ROLE).getAsString();

    // Get system ID.
    String systemId = null;
    if (authData.has(DeepstreamConstants.REST_FIELD_SYSTEM_ID)) {
            systemId = authData.get(DeepstreamConstants.REST_FIELD_SYSTEM_ID).getAsString();
    }

    // Get auth token.
    String token = null;
    if (authData.has(DeepstreamConstants.REST_FIELD_TOKEN)) {
            token = authData.get(DeepstreamConstants.REST_FIELD_TOKEN).getAsString();
    }


    if (systemId != null) {
            // Check access.
            log.debug("Checking deepstream access for external system {}", systemId);
            val accessGranted = restManagerService.checkExternalSystemAccess(clientVersion, role,
                                                                              systemId, token);

            if (!accessGranted) {
                    response.sendError(HttpServletResponse.SC_FORBIDDEN);
                    return;
            }
            // Send response.
            val responseServerData = new JsonObject();
            responseServerData.addProperty(DeepstreamConstants.REST_FIELD_SYSTEM_ID, systemId);
            responseServerData.addProperty(DeepstreamConstants.REST_FIELD_ROLE, role);

            val responseClientData = new JsonObject();
            responseClientData.addProperty(DeepstreamConstants.REST_FIELD_SYSTEM_ID, systemId);

            val responseData = new JsonObject();
            responseData.addProperty(DeepstreamConstants.DS_FIELD_USERNAME, systemId + " " + role);
            responseData.add(DeepstreamConstants.DS_FIELD_CLIENT_DATA, responseClientData);
            responseData.add(DeepstreamConstants.DS_FIELD_SERVER_DATA, responseServerData);

            val responseAsBytes = gson.toJson(responseData).getBytes(Charsets.UTF_8);

            response.setContentType(MediaType.APPLICATION_JSON);
            response.setContentLength(responseAsBytes.length);

            response.getOutputStream().write(responseAsBytes);
    }
…
}
}
```

Listing 13 Verarbeitung der Logindaten eines externen Systems

4.12 Klasse – RESTManagerService

```
@Log4j2
public class RESTManagerService extends Thread {
…
    public boolean checkExternalSystemAccess(final int clientVersion,
                                             final String role,
                                             final String systemId,
                                             final String token) {

    // Prevent access for too old or new clients.
    if (clientVersion < deepstreamMinClientVersion || clientVersion > deepstreamMaxClientVersion) {
            return false;
    }

    // Prevent unauthorized access with empty values.
    if (StringUtils.isBlank(systemId) || StringUtils.isBlank(role) || StringUtils.isBlank(token)) {
            return false;
    }

    // Check access based on role.
    if (role.equals(deepstreamExternalServiceRole)) {

            // Check if systemId exists.
    val externalSystem = databaseManagerService.findOneModelObject(InterventionExternalSystem.class,
                                    Queries.INTERVENTION_EXTERNAL_SYSTEM__BY_SYSTEM_ID, systemId);

            if (externalSystem == null) {

    log.debug("System id {} not authorized for deepstream access: System id not found", systemId);
                    return false;
            }

            // Check if external system is active.
            if(!externalSystem.isActive()) {

                    log.debug("System with id {} is inactive", systemId);
                    return false;
            }

            // Validate token saved in record with path: "external-systems/[systemId]".
            if (deepstreamCommunicationService != null
                    && deepstreamCommunicationService.checkExternalSystemToken(systemId, token)) {

    log.debug("System {} with id {} authorized for deepstream access", externalSystem.getName(),
                                                                        systemId);
                    return true;
            } else {

    log.debug("System {} with id {} not authorized for deepstream access: Wrong token",
                                            externalSystem.getName(), systemId);
                    return false;
            }

    } else {

    log.debug("Unauthorized access with wrong role {}", role);
            return false;
    }
    }
…
}
```

Listing 14 Validierung der Logindaten eines externen Systems

4.13 Klasse – ExternalSystemsManagerService

```
 1 /**
 2  * Manages all defined external systems for a specific intervention.
 3  *
 4  * @author Marcel Schlegel
 5  */
 6 @Log4j2
 7 public class ExternalSystemsManagerService {
…
 8     // Convert ExternalSystemMessages to ReceivedMessages
 9     public boolean getReceivedMessages(
10                     final ArrayList<ReceivedMessage> receivedMessages) {
11
12             List<ExternalSystemMessage> externalSystemMessages = new ArrayList<>();
13             // Get all received external system messages from deepstream service.
14             deepstreamCommunicationService
15                             .getReceivedExternalSystemMessages(externalSystemMessages);
16             // Iterate external system messages.
17             for (ExternalSystemMessage externalSystemMessage : externalSystemMessages) {
18                     // Find the related external system from the received external
19                     // message.
20                     val externalSystem = databaseManagerService.findOneModelObject(
21                                     InterventionExternalSystem.class,
22                                     Queries.INTERVENTION_EXTERNAL_SYSTEM__BY_SYSTEM_ID,
23                                     externalSystemMessage.getSystemId());
24                     // Check if the external system exists.
25                     if (externalSystem == null) {
26                     log.error("There exists no external system with system id {}.
27                      Message can not be processed.", externalSystemMessage.getSystemId());
28                             continue;
29                     }
30                     // Check if the received external message provides participants
31                     if (externalSystemMessage.getParticipants().isEmpty()) {
32                             // No participants provided so use all participants from current
33                             // intervention.
34                             val participants = databaseManagerService.findModelObjects(
35                                             Participant.class, Queries.PARTICIPANT__BY_INTERVENTION,
36                                             externalSystem.getIntervention());
37                             participants.forEach(participant -> externalSystemMessage
38                                             .addParticipant(participant.getId().toString()));
39                     }
40                     // Iterate the participants and build for everyone a ReceivedMessage
41                     // object.
42                     for (String participantId : externalSystemMessage
43                                     .getParticipants()) {
44                             // Check if the participant id is valid.
45                             if (!ObjectId.isValid(participantId)) {
46                     log.warn("Participant id {} is not valid. Message with system id {}
47                      can not be processed for participant id {}.", participantId,
48                      externalSystem.getSystemId(), participantId);
49                                     continue;
50                             }
51                             val participant = databaseManagerService.getModelObjectById(
52                                             Participant.class, new ObjectId(participantId));
53                             // Check if a persisted participant with the given id exists.
54                             if (participant == null) {
55                     log.warn("Participant with id {} not found.
56                     Message with system id {} can not be processed for participant id {}.",
57                      participantId, externalSystem.getSystemId(), participantId);
58                                     continue;
59                             }
60                             // Check if a found participant belongs to the current
61                             // intervention.
62                             if (!participant.getIntervention()
63                                             .equals(externalSystem.getIntervention())) {
64                             log.warn("Participant with id {} is not in the same intervention as the
65                             external system. Message with system id {} can not be processed for
66                              participant id {}.",
67                              participantId, externalSystem.getSystemId(), participantId);
68                                     continue;
69                             }
```

Listing 15 Klasse – ExternalSystemsManagerService – Teil 1

```
70                          // Build the ReceivedMessage.
71                          val receivedMessage = mapExternalSystemMessageOnReceivedMessage(
72                                  participant.getId(), externalSystem,
73                                  externalSystemMessage);
74
75                          receivedMessages.add(receivedMessage);
76                  }
77          }
78          return !externalSystemMessages.isEmpty();
79  }
80
81  // Map ExternalSystemMessage on ReceivedMessage.
82  private ReceivedMessage mapExternalSystemMessageOnReceivedMessage(
83                  ObjectId participantId, InterventionExternalSystem externalSystem,
84                  ExternalSystemMessage externalSystemMessage) {
85          // Set all necessary values for the ReceivedMessage.
86          val receivedMessage = new ReceivedMessage();
87          receivedMessage.addAllExternalSystemVariables(
88                          mapVariables(externalSystem, externalSystemMessage));
89
90          receivedMessage.setTypeIntention(false);
91          receivedMessage.setRelatedMessageIdBasedOnOrder(-1);
92          receivedMessage.setReceivedTimestamp(InternalDateTime.currentTimeMillis());
93          receivedMessage.setExternalSystemId(externalSystemMessage.getSystemId());
94          receivedMessage.setExternalSystem(true);
95          receivedMessage.setMessage("");
96
97          val dialogOptions = databaseManagerService.findModelObjects(
98                          DialogOption.class, Queries.DIALOG_OPTION__BY_PARTICIPANT,
99                          participantId);
100         for (DialogOption dialogOption : dialogOptions) {
101                 receivedMessage.setSender(dialogOption.getData());
102
103                 if (dialogOption.getType() == DialogOptionTypes.EXTERNAL_ID) {
104                         receivedMessage.setType(DialogOptionTypes.EXTERNAL_ID);
105                         break;
106                 } else if (dialogOption
107                                 .getType() == DialogOptionTypes.SUPERVISOR_EXTERNAL_ID) {
108                         receivedMessage.setType(DialogOptionTypes.SUPERVISOR_EXTERNAL_ID);
109                         break;
110                 } else {
111                         receivedMessage.setType(dialogOption.getType());
112                 }
113         }
114
115         return receivedMessage;
116 }
```

Listing 16 Klasse – ExternalSystemsManagerService – Teil 2

```
117     // Map the received external message key-value pairs to intervention
118     // variables.
119     private Collection<Variable> mapVariables(
120                     InterventionExternalSystem externalSystem,
121                     ExternalSystemMessage externalSystemMessage) {
122
123             // Get all defined variable mappings from the external system.
124             val variableMappings = databaseManagerService.findModelObjects(
125                             InterventionExternalSystemFieldVariableMapping.class,
126     Queries.INTERVENTION_EXTERNAL_SYSTEM_FIELD_VARIABLE_MAPPING__BY_INTERVENTION_EXTERNAL_SYSTEM,
127                             externalSystem.getId());
128
129             Collection<Variable> variables = new ArrayList<>();
130             // Iterate the variable mappings.
131             for (InterventionExternalSystemFieldVariableMapping variableMapping : variableMappings) {
132
133                     // Get the related InterventionVariableWithValue.
134                     val interventionVariable = databaseManagerService
135                                     .getModelObjectById(InterventionVariableWithValue.class,
136                                             variableMapping.getInterventionVariableWithValue());
137                     // Check if InterventionVariableWithValue exists.
138                     if (interventionVariable == null) {
139                     log.warn("Mapped intervention variable for field {} not found.
140                      Mapping is ignored for message with system id {}.",
141                      variableMapping.getFieldName(), externalSystem.getSystemId());
142                             continue;
143                     }
144                     // Check if the received external message contains a key (field)
145                     // name which equals
146                     // to a defined variable mapping from the external system.
147                     val externalSystemVariable = externalSystemMessage.getVariables()
148                                     .get(variableMapping.getFieldName());
149
150                     // If the received message contains a key (field) name and a value
151                     // for a mapping use it.
152                     // If not use the default value from the mapped
153                     // InterventionVariableWithValue.
154                     Variable externalSystemVariableWithInterventionVariableName = new Variable(
155                                     interventionVariable.getName(),
156                                     externalSystemVariable == null
157                                                     // Intervention variable value as
158                                                     // default
159                                                     ? interventionVariable.getValue()
160                                                     : externalSystemVariable.getValue());
161
162                     variables.add(externalSystemVariableWithInterventionVariableName);
163             }
164
165             return variables;
166     }
167 }
```

Listing 17 Klasse – ExternalSystemsManagerService – Teil 3

4.14 Snippet – Verknüpfung mit der GET.ON-Plattform

```
<?php
…
// Compose functions for calling it via shortcode.
add_shortcode('shortcode_connect_mobile_coach', function ($attributes) {
  $validAtts = shortcode_atts(array(
       'user' => '',
    'password' => '',
    'participant_id' => ''
), $attributes);
// Check logindata.
$result = auth_user($validAtts[user], $validAtts[password]);
if ($result->failure) {
     return $result->message;
}
// Create table if it doesn't exist.
$wp_user = $result->result_object;
$result = create_participant_table();
if ($result->failure) {
    return $result->message;
}
// Insert data record
$result = insert_wp_user_mc_participant($wp_user, $validAtts[participant_id]);
if ($result->failure) {
    return $result->message;
}
// If everything is successful send javascript back to the MobileCoach App for closing the webview.
return <<<EOF
            <script type="text/javascript">
                    function checkReady() {
                            if (window.postMessage != undefined && window.postMessage != null &&
typeof window.postMessage === 'function' && window.postMessage.length === 1) {
                                    sendResults();
                            } else {
                                    setTimeout(checkReady, 100);
                            }
                    }
                    function sendResults() {

                            window.postMessage('close');
                    }
                    $(document).ready(function() {
                            setTimeout(checkReady, 100);
                    });
            </script>
EOF;
});
```

Listing 18 Verknüpfung der MobileCoach-Interventions-ID mit der GET.ON-Plattform

5 Testfälle

5.1 Testfall FA12.1

Testfall	**FA12.1**
Anforderung	FA12
Beschreibung	Authentifizierung der GET.ON-Plattform am MobileCoach-Server.
Vorbedingung	- Das externe System „GET.ON Plattform“ existiert auf MobileCoach-Serverseite. - Der PHP-Deepstream-Client ist auf GET.ON-Serverseite implementiert.
Fehlerfall	-
Eingaben	1. Der Konstruktor der `DeepstreamClient`-Klasse wird mit den Parametern: `"https://ba.eatnbyte.com/ds-http/", array('token' => '1;external-system;jwdki890-mgj92yeay9c;HzOnhbgEICHiJniZf1m…')` aufgerufen. 2. Das `DeepstreamClient`-Objekt wird erzeugt. 3. Der Login wird ausgeführt.
Erwartetes Verhalten	Der Deepstream-Client ist am MobileCoach-Server authentifiziert.

Ausgaben	**MobileCoach-Serverkonsole:**
	`DEBUG h.ethz.mc.servlets.DeepstreamRESTServlet [http-nio-8080-exec-3] - Deepstream servlet call`
	`DEBUG h.ethz.mc.servlets.DeepstreamRESTServlet [http-nio-8080-exec-3] - Checking deepstream access for external system jwdki890-mgj92yeay9c`
	`DEBUG .internal.DeepstreamCommunicationService [http-nio-8080-exec-3] - Checking system token for jwdki890-mgj92yeay9c`
	`DEBUG .internal.DeepstreamCommunicationService [http-nio-8080-exec-3] - Token check for system jwdki890-mgj92yeay9c returns true`
	`DEBUG ch.ethz.mc.services.RESTManagerService [http-nio-8080-exec-3] - System GET.ON Plattform with id jwdki890-mgj92yeay9c authorized for deepstream access`

Erfolgreich	Ja

Tabelle 1 Testfall FA12.1

5.2 Testfall FA12.2

Testfall	**FA12.2**
Anforderung	FA12
Beschreibung	Authentifizierung der GET.ON-Plattform am MobileCoach-Server.
Vorbedingung	- Das externe System „GET.ON Plattform“ existiert auf Mobile-Coach-Serverseite. - Der PHP-Deepstream-Client ist auf GET.ON-Serverseite implementiert.
Fehlerfall	- Die angegebene System-ID und/oder das Token ist falsch.
Eingaben	1. Der Konstruktor der `DeepstreamClient`-Klasse wird mit einem falschen System-ID/Token-Parameter aufgerufen. 2. Das `DeepstreamClient`-Objekt wird erzeugt. 3. Der Login wird ausgeführt.
Erwartetes Verhalten	Die Authentifizierung des Deepstream-Clients schlägt fehl.

Ausgaben	**MobileCoach-Serverkonsole:**
	Falsches Token:
	`DEBUG h.ethz.mc.servlets.DeepstreamRESTServlet [http-nio-8080-exec-9] - Deepstream servlet call`
	`DEBUG h.ethz.mc.servlets.DeepstreamRESTServlet [http-nio-8080-exec-9] - Checking deepstream access for external system jwdki890-mgj92yeay9c`
	`DEBUG .internal.DeepstreamCommunicationService [http-nio-8080-exec-9] - Checking system token for jwdki890-mgj92yeay9c`
	`DEBUG .internal.DeepstreamCommunicationService [http-nio-8080-exec-9] - Token check for system jwdki890-mgj92yeay9c returns false`
	`DEBUG ch.ethz.mc.services.RESTManagerService [http-nio-8080-exec-9] - System GET.ON Plattform with id jwdki890-mgj92yeay9c not authorized for deepstream access: Wrong token`
	Falsche System-ID:
	`DEBUG h.ethz.mc.servlets.DeepstreamRESTServlet [http-nio-8080-exec-10] - Deepstream servlet call`
	`DEBUG h.ethz.mc.servlets.DeepstreamRESTServlet [http-nio-8080-exec-10] - Checking deepstream access for external system jjwdki890-mgj92yeay9c`
	`DEBUG ch.ethz.mc.services.RESTManagerService [http-nio-8080-exec-10] - System id jjwdki890-mgj92yeay9c not authorized for deepstream access: System id not found`

Erfolgreich	Ja

Tabelle 2 Testfall FA12.2

5.3 Testfall FA13.1

Testfall	FA13.1
Anforderung	FA13
Beschreibung	Nachricht von der GET.ON-Plattform, über den MobileCoach-Server, an MobileCoach-App senden.
Vorbedingung	- Das externe System „GET.ON Plattform“ existiert auf Mobile-Coach-Serverseite. - Der PHP-Deepstream-Client ist auf GET.ON-Serverseite implementiert. - Ein Studienteilnehmer hat die MobileCoach-App mit seinem GET.ON-Profil verknüpft. - Eine „External Message“ Interventionsregel verschickt beim Empfang einer externen Nachricht eine Interventionsnachricht.
Fehlerfall	-
Eingaben	1. Das `DeepstreamClient`-Objekt wird erzeugt. 2. Der Login wird erfolgreich ausgeführt. 3. Die Variable `$participants` enthält eine Liste mit Interventionsteilnehmern, an die eine Nachricht versendet wird. 4. Die Variable `$variables` enthält Schlüssel-Wert-Paare, die auf Interventionsvariablen abgebildet werden. 5. Die `external-message`-RPC-Methode wird mit den Parametern: `array('systemId' => 'jwdki890-mgj92yeay9c', 'participants' => $participants, 'variables' => $variables)` aufgerufen.
Erwartetes Verhalten	Der Studienteilnehmer empfängt eine Nachricht in seiner MobileCoach-App.

Ausgaben	**MobileCoach-Serverkonsole:**
	`DEBUG .internal.DeepstreamCommunicationService [pool-5-thread-1] - Received external message for system jwdki890-mgj92yeay9c`
	`DEBUG ces.internal.CommunicationManagerService [Incoming Message Worker] - Retrieving messages from external system...`
	`DEBUG ervices.internal.VariablesManagerService [Incoming Message Worker] - Storing variable $participantUnexpectedMessage with value  for participant 5cf28adf7fa7b7000d7339c2`
	`...`
	`DEBUG ervices.internal.VariablesManagerService [Incoming Message Worker] - Storing variable $testFeld1 with value GET.ON Nutzer: Marcel for participant 5cf28adf7fa7b7000d7339c2`
	`DEBUG ervices.internal.VariablesManagerService [Incoming Message Worker] - Storing variable $testFeld2 with value GET.ON Mail: marcel@... for participant 5cf28adf7fa7b7000d7339c2`
	`...`
	`DEBUG .internal.DeepstreamCommunicationService [Incoming Message Worker] - Acknowledging message 5cf2e9f75d7156000d98508b`
	`DEBUG .internal.DeepstreamCommunicationService [Incoming Message Worker] - Message 5cf2e9f75d7156000d98508b acknowledged`

Erfolgreich	Ja

Tabelle 3 Testfall FA13.1

5.4 Testfall FA13.2

Testfall	**FA13.2**
Anforderung	FA13
Beschreibung	Nachricht von der GET.ON-Plattform über den MobileCoach-Server an die MobileCoach-App senden.
Vorbedingung	- Das externe System „GET.ON Plattform“ existiert auf MobileCoach-Serverseite. - Der PHP-Deepstream-Client ist auf GET.ON-Serverseite implementiert. - Ein Studienteilnehmer hat die MobileCoach-App mit seinem GET.ON-Profil verknüpft. - Eine „External Message“ Interventionsregel verschickt beim Empfang einer externen Nachricht eine Interventionsnachricht.
Fehlerfall	- Die angegebene System-ID ist falsch.
Eingaben	1. Das `DeepstreamClient`-Objekt wird erzeugt. 2. Der Login wird erfolgreich ausgeführt. 3. Die `external-message`-RPC-Methode wird mit einem falschen System-ID-Parameter aufgerufen.
Erwartetes Verhalten	Der Studienteilnehmer empfängt keine Nachricht in seiner MobileCoach-App und eine Fehlermeldung wird erzeugt.
Ausgaben	**GET.ON-Plattform:** `Es ist ein Fehler aufgetreten. Nachricht konnte nicht verarbeitet werden. Siehe MobileCoach-Serverlogs.`
Erfolgreich	Ja

Tabelle 4 Testfall FA13.2

5.5 Testfall NFA01.1

Testfall	NFA01.1
Anforderung	NFA01
Beschreibung	20 Nachrichten können parallel/sequentiell von der GET.ON-Plattform über den MobileCoach-Server an die MobileCoach-App gesendet werden.
Vorbedingung	- Das externe System „GET.ON Plattform“ existiert auf MobileCoach-Serverseite. - Der PHP-Deepstream-Client ist auf GET.ON-Serverseite implementiert. - Ein Studienteilnehmer hat die MobileCoach-App mit seinem GET.ON-Profil verknüpft. - Eine „External Message“ Interventionsregel verschickt beim Empfang einer externen Nachricht eine Interventionsnachricht.
Fehlerfall	-
Eingaben	1. Das `DeepstreamClient`-Objekt wird erzeugt. 2. Der Login wird erfolgreich ausgeführt. 3. Die `external-message`-RPC-Methode wird 20-mal aufgerufen.
Erwartetes Verhalten	Der Studienteilnehmer empfängt 20 Nachrichten in seiner MobileCoach-App.

Ausgaben	**MobileCoach-Serverkonsole:**
	`DEBUG .internal.DeepstreamCommunicationService [pool-5-thread-1] - Received external message for system jwdki890-mgj92yeay9c`
	`...`
	`DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5d85d7156000d985095`
	`DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5d85d7156000d985097`
	`DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5d85d7156000d985099`
	`DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5d85d7156000d98509b`
	`DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5d95d7156000d98509d`
	`DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5d95d7156000d98509f`
	`DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5d95d7156000d9850a1`
	`DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5da5d7156000d9850a3`
	`DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5da5d7156000d9850a5`
	`DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5da5d7156000d9850a7`
	`DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5da5d7156000d9850a9`
	`DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5da5d7156000d9850ab`

	DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5db5d7156000d9850ad DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5db5d7156000d9850af DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5db5d7156000d9850b1 DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5db5d7156000d9850b3 DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5dc5d7156000d9850b5 DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5dc5d7156000d9850b7 DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5dc5d7156000d9850b9 DEBUG .internal.DeepstreamCommunicationService [Outgoing Message Worker] - Sending message 5cf2f5dc5d7156000d9850bb
Erfolgreich	Ja

Tabelle 5 Testfall NFA01.1

6 Integrationsanleitung

LEUPHANA
UNIVERSITÄT LÜNEBURG

Integrationsanleitung

Anleitung für die Integration der Nachrichtenschnittstelle zwischen der GET.ON GesundheitsTraining.Online-Plattform und dem MobileCoach

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

In dieser Integrationsanleitung wird schrittweise erklärt wie die GET.ON-Plattform im MobileCoach-Server als externes System integriert werden kann. Ein aufgesetzter MobileCoach-Server mit valider Deepstream-Konfiguration wird vorausgesetzt.
Als Beispiel und möglicher Anwendungsfall wird folgendes Szenario beschrieben:
Ein Studienteilnehmer der GET.ON-Plattform ist über einen längeren Zeitraum abwesend.
Die GET.ON-Plattform überprüft diesen Zeitraum (z. B. 7 Tage) und sendet über die MobileCoach-App eine Erinnerungsnachricht das studienrelevante Training fortzuführen.

2 MobileCoach-Server

2.1 Schritt 1 – GET.ON-Plattform als externes System erstellen

Über die Administrationsoberfläche des MobileCoach-Servers wird ein externes System erstellt. Folgende Schritte sind auszuführen:

1. Der Administrator/Autor einer Intervention ruft die webbasierte Administrationsoberfläche des MobileCoach-Servers auf und meldet sich mit Benutzernamen und Passwort an.

2. Zur Interventionsübersicht wechseln, eine Intervention auswählen und diese bearbeiteten (auf „Edit“ Button klicken).

3. Zum letzten Tab „External Systems“ wechseln.
   3.1. Das Monitoring der Intervention muss deaktiviert sein (siehe Tab „Basic Settings and Modules“), da ansonsten eine Bearbeitung der externen Systeme nicht möglich ist.

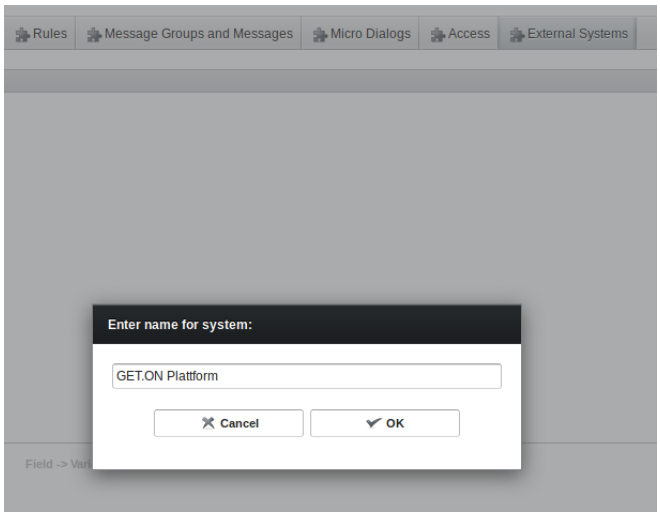


Abbildung 1 Externes System „GET.ON Plattform“ erstellen

4. Auf „New“ Button klicken, einen externen Systemnamen vergeben z. B. „GET.ON Plattform“ und mit „OK“ bestätigen.

5. Das externe System „GET.ON Plattform“ ist erfolgreich erstellt.

2.2 Schritt 2 – Abbildungen auf Interventionsvariablen erstellen

2.2.1 Interventionsvariable erstellen

Damit in einer Erinnerungsnachricht, die an die MobileCoach-App gesendet wird übertragene Daten eingebettet werden können, ist die Erstellung einer Interventionsvariable erforderlich. Diese kann z. B. das Datum des letzten Logins eines Studienteilnehmers beinhalten. Die Variable kann später in einer Interventionsnachricht referenziert werden.
Für die Erstellung sind folgende Schritte auszuführen:

1. Zum Tab „Variables“ wechseln.

2. Auf „New“ Button klicken, einen Variablennamen vergeben z. B. „`$lastGetOnLogin`“ und mit „OK“ bestätigen.
   2.1. Optional kann der Standardwert „0“ verändert werden. Dieser wird z. B. in der Interventionsnachricht angezeigt, wenn keine Daten, die auf die Interventionsvariable abgebildet werden übertragen wurden.
   2.2. Zum Ändern des Standardwertes auf „Edit“ Button klicken, einen Wert vergeben z. B. „kein Datum bekannt“ und mit „OK“ bestätigen.

3. Das Zugriffslevel der Variable darf nicht auf „internal“ gesetzt sein und muss verändert werden.

4. Dafür auf den Button „Switch Access“ klicken, bis das Zugriffslevel auf „manageable by service“ oder ein anderweitiges (außer „internal“) gesetzt ist.

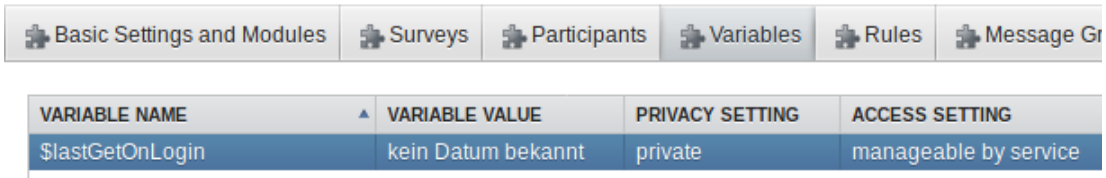


Abbildung 2 Interventionsvariable erstellen

2.2.2 Abbildung erstellen

Die erstellte Interventionsvariable soll einem externen System zugeordnet werden. Dafür wird eine Abbildung erstellt die die übertragenen Daten, anhand eines Schlüsselnamens, einer Interventionsvariable zuordnet.
Für die Erstellung der Abbildung sind folgende Schritte auszuführen:

1. Zum Tab „External Systems“ wechseln.

2. Externes System „GET.ON Plattform“ selektieren.

3. Auf Button „Field -> Variable Mapping“ klicken.

4. Auf „New“ Button klicken.

5. Als Schlüssel- bzw. Feldnamen z. B. „letzterLogin“ eingeben.

6. Im Dropdown-Menü die Interventionsvariable „`$lastGetOnLogin`“ auswählen.

7. Mit „OK“ bestätigen und das Fenster mit „Close“ schließen.

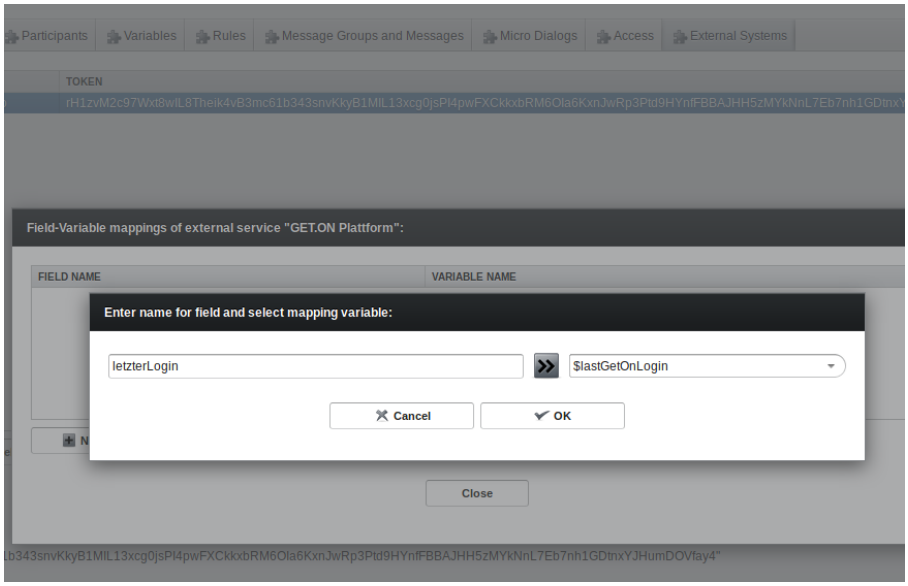


Abbildung 3 Abbildung erstellen

2.3 Schritt 3 – Interventionsnachricht erstellen

Eine Interventionsnachricht wird später an Studienteilnehmer über die MobileCoach-App geschickt. Zur Erstellung der Interventionsnachricht sind folgende Schritte notwendig:

1. Zum Tab „Message Groups and Messages“ wechseln.

2. Auf Button „New Group“ klicken, einen Namen für die Nachrichtengruppe vergeben und mit „OK“ bestätigen.

3. Auf Button „New“ klicken.

4. Im geöffneten Fenster ganz oben, neben „Text (with placeholders)“, auf „Edit“ klicken.

5. Jetzt kann eine Interventionsnachricht erstellt werden. Dabei wird die Interventionsvariable „`$lastGetOnLogin`“ in den Text eingebettet z. B.:
   ```
   „Hallo,
   du hast dich seit dem $lastGetOnLogin nicht mehr an der GET.ON Plattform angemeldet. Das ist schon 7 Tage her.
   Ich würde mich freuen, wenn du mal wieder vorbeischauen würdest.“
   ```

6. Mit „OK“ bestätigen und das Fenster mit „Close“ schließen.

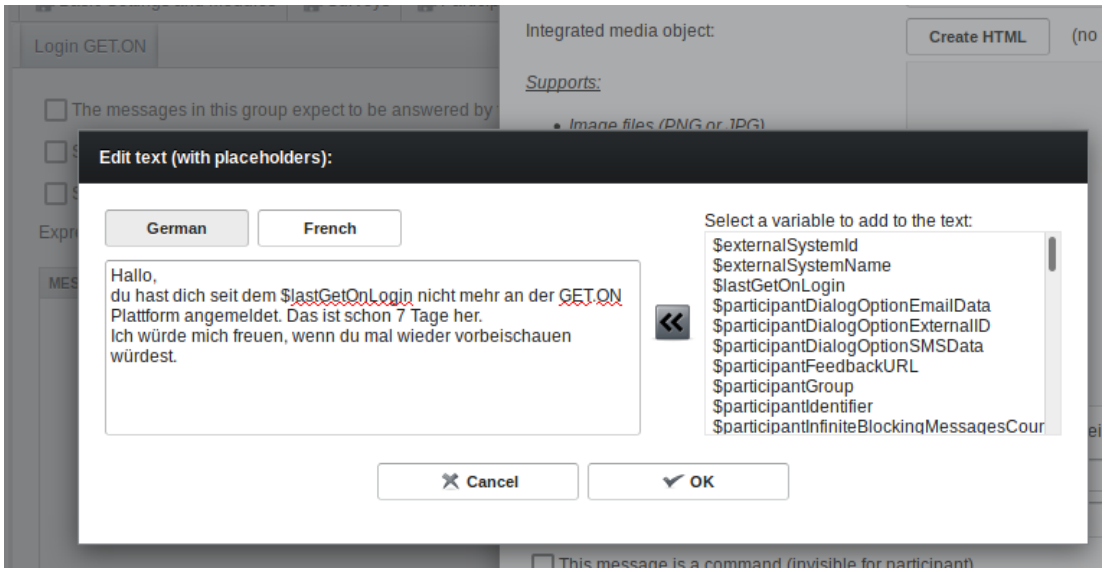


Abbildung 4 Interventionsnachricht erstellen

2.4 Schritt 4 – Interventionsregel erstellen

Um auf eine eingehende Nachricht vom externen System „GET.ON Plattform“ zu reagieren, muss eine entsprechende Interventionsregel erstellt werden. Diese initiiert das Versenden der Interventionsnachricht an Studienteilnehmer. Folgende Schritte sind notwendig:

1. Zum Tab „Rules“ wechseln.

2. Die Regel „Execution on EXTERNAL MESSAGE“ selektieren.

3. Auf „New“ Button klicken.
   Innerhalb der Regel wird auf das externe System „GET.ON Plattform“ abgefragt.

4. Auf „Edit“ Button „Rule [x] (with placeholders)“ klicken, die Systemvariable „`$externalSystemName`“ auswählen und mit „OK“ bestätigen.

5. Im mittigen Dropdown-Menü „text value equals“ auswählen.

6. Auf „Edit“ Button „Comparison term [y] (with placeholders)“ klicken, „GET.ON Plattform“ eingeben und mit „OK“ bestätigen.

7. Haken bei „Send message if rule result is TRUE“ setzen und aus dem „Message group to send messages from“ Dropdown-Menü die zuvor angelegte „Message Group“ auswählen.

8. Das Fenster mit „Close“ schließen.

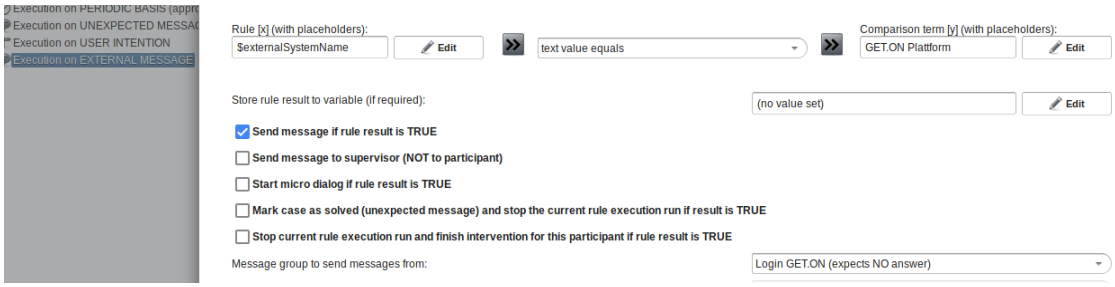


Abbildung 5 Interventionsregel erstellen

Es wäre auch möglich gewesen, auf weitere Regeln zu prüfen bzw. weitere Interventionsvariablen mit einzubeziehen.

3 Externes System – GET.ON-Plattform

Ein externes System wie die GET.ON-Plattform, kann über einen Deepstream-Client mit dem MobileCoach-Server kommunizieren. Das in der Einleitung beschriebene Szenario wird in der WordPress-basierten GET.ON-Plattform mit Hilfe eines Cron-Jobs umgesetzt.

3.1 Schritt 1 – Deepstream-Client implementieren

Für die GET.ON-Plattform wird der PHP-Deepstream-Client gewählt. Dieser kommuniziert ausschließlich über HTTP. Die Installation der Deepstream-Client-Programmbibliothek wird vorausgesetzt.
Beim erzeugen des Deepstream-Client-Objekts wird die URL zum Deepstream-Server des MobileCoach-Servers übergeben. Die Login-Parameter werden aus der MobileCoach-Server Administrationsoberfläche des externen Systems kopiert und übergeben. Folgende Schritte sind zu durchlaufen:

1. In der Administrationsoberfläche des MobileCoach-Servers zum Tab „External Systems“ wechseln.

2. Externes System „GET.ON Plattform“ selektieren.
   Unterhalb der Übersichtstabelle werden Textbereiche generiert die zum einen die Login-Parameter einer Kommunikation des Deepstream-Clients über das Websocket-Protokoll („System Login JSON“) und HTTP („System Login HTTP auth“) abbilden.

3. Das Token aus dem „System Login HTTP auth“ Textbereich wird kopiert.

Abbildung 6 HTTP-Auth-Parameter

4. Mit dem kopierten Token kann folgendes PHP-Skript mit der Initialisierung der `DeepstreamClient`-Klasse erstellt werden:

```php
<?php
// Change URL to correct MobileCoach Deepstream-Server route.
$client = new \Deepstreamhub\DeepstreamClient("https://ba.eatnbyte.com/ds-http/", array(
    // Token for authetification.
    'token' =>
    '1;external-system;jwfela2e-iz9nepfa4io;HzOnhbgEICHiJniZf1mxvM
HIysY1uelALB2PiLQhj0mjT8FQMDrcOqyuTxY5A0Y3QrK26yWbxMU…'
));
…
```

Listing 1 Initialisierung der DeepstreamClient-Klasse

3.2 Schritt 2 – Aufruf der RPC-Methode external-message

Um einen RPC-Aufruf der Methode `external-message` auszuführen wird die Methode `makeRpc` vom `DeepstreamClient` aufgerufen. Dieser wird als Parameter der Methodenname „external-message“ übergeben. Die Weiteren Paramater können aus der Administrationsoberfläche des externen Systems kopiert und übergeben werden. Folgende Schritte sind auszuführen:

1. In der Administrationsoberfläche des MobileCoach-Servers zum Tab „External Systems“ wechseln.

2. Externes System „GET.ON Plattform“ selektieren.
   Unterhalb der Übersichtstabelle wird der Textbereich („System „external-message“ JSON“) generiert der die Parameter der `external-message`-Methode enthält.

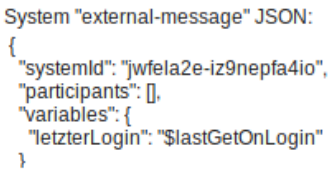


Abbildung 7 External-message-Methodenparameter

3. Die im JSON-Dokument enthaltenen Parameter werden aus dem Textbereich kopiert.

4. Mit den kopierten Parametern kann das erstellte PHP-Skript erweitert werden:

```php
<?php
// Change URL to correct MobileCoach Deepstream-Server route.
$client = new \Deepstreamhub\DeepstreamClient("https://ba.eatnbyte.com/ds-http/", array(
    // Token for authetification.
    'token' =>
    '1;external-system;jwfela2e-iz9nepfa4io;HzOnhbgEICHiJniZf1mxvM
HIysY1uelALB2PiLQhj0mjT8FQMDrcOqyuTxY5A0Y3QrK26yWbxMU…'
));
// Result is true for successful data transmission and false if data transmission fails.
$result = $client->makeRpc('external-message', array(
    'systemId' => 'jwfela2e-iz9nepfa4io',
    'participants' => [],
    'variables' => array(
        'letzterLogin' => ''
    )
));
…
```

Listing 2 Aufruf der RPC-Methode external-message

3.3 Schritt 3 – Erstellen des Cron-Jobs

Cron-Jobs können innerhalb von WordPress über das Plugin „WP Crontrol“ erstellt werden. Dieses wird als installiert vorausgesetzt.
Für die Erstellung des Cron-Jobs und Implementierung des PHP-Skripts sind folgende Schritte auszuführen:

1. In der WordPress-Administrationsoberfläche auf „Werkzeuge“ >> „Cron-Ereignisse“ klicken.

2. In der Ansicht unten den Tab „PHP-Cron-Ereignis hinzufügen“ auswählen.

3. Das zuvor erstellte und nochmals erweiterte Skript in das „PHP-Code“-Textfeld einfügen:

```php
// Change URL to correct MobileCoach Deepstream-Server route.
$client = new \Deepstreamhub\DeepstreamClient("https://ba.eatnbyte.com/ds-http/", array(
    // Token for authetification.
    'token' =>
    '1;external-system;jwfela2e-iz9nepfa4io;HzOnhbgEICHiJniZf1mxvM
HIysY1uelALB2PiLQhj0mjT8FQMDrcOqyuTxY5A0Y3QrK26yWbxMU…'
));

// DB query which selects the last login date from all GET.ON users.
// If the last login date differs more than 7 days from now, select these records.
// The last step is an inner join which tries to map the user_id to a connect MobileCoach-App user_id
in wp_mc_participants.
$queryResult = $wpdb->get_results('SELECT a.participant_id, b.login_date
FROM ' . $wpdb->prefix . 'mc_participants a
       INNER JOIN (SELECT user_id,
                          user_login,
                          login_date
                   FROM ' . $wpdb->prefix . 'aiowps_login_activity
                   WHERE  ( user_id, user_login, login_date ) IN (SELECT
                          user_id,
                          user_login,
                          Max(login_date)
                                                                  FROM
                          ' . $wpdb->prefix . 'aiowps_login_activity
                          GROUP  BY user_id)
                          AND Datediff(Now(), login_date) > 7) b
               ON a.wp_users_id = b.user_id');

// If there are selected users send a message to their MobileCoach-App.
foreach ($queryResult as $row) {
    // Result is true for successful data transmission and false if data transmission fails.
    // Result isn't currently handled.
    $result = $client->makeRpc('external-message', array(
        'systemId' => 'jwfela2e-iz9nepfa4io',
        'participants' => array(
            $row->participant_id
        ),
        'variables' => array(
            'letzterLogin' => $row->login_date
        )
    ));
}
```

Listing 3 Komplettes Listing zum Überprüfen der GET.ON-Logins

4. Ein Ereignisname kann vergeben werden z. B. „Prüfe letzten Login -> MobileCoach-App“.

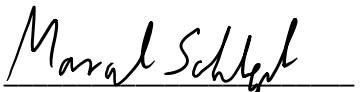
5. Im Dropdown-Menü „Wiederholung“ kann „Einmal täglich“ ausgewählt werden.

6. Mit einem Klick auf den Button „PHP-Cron-Ereignis hinzufügen“ wird der Cron-Job angelegt.

Eigenständigkeitserklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne unerlaubte fremde Hilfe angefertigt, andere als die angegeben Quellen und Hilfsmittel nicht benutzt und die benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Die Arbeit habe ich in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt.

Lüneburg, 04.06.2019
Ort, Datum



Unterschrift (Marcel Schlegel)