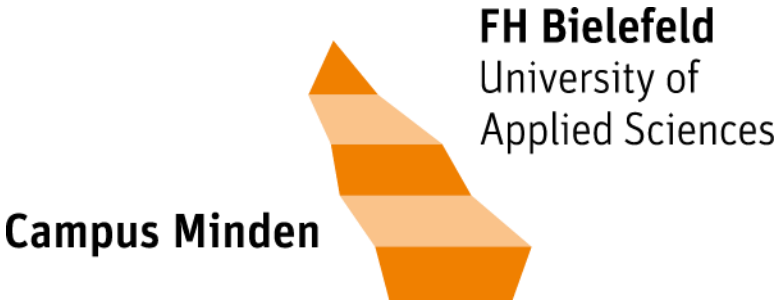


Webapplikations-Anbindung

Projektbericht

im Fach Web Engineering im Wintersemester 2019/20

bei Grit Behrens und Florian Fehring

erstellt von

Benedikt Wiest, Steffen Dorsch, Luca Berneking und Julien Riechmann

Januar 25, 2020

Inhalt

Einleitung

Motivation

Im Rahmen unseres Wahlfachs Webengeneering haben wir uns mit dem Themenbereich Web- und Businessapplikationsanbindung beschäftigt. Die Bedeutung von Schnittstellen innerhalb der Informatik steigt enorm, parallel steigt jedoch auch die Komplexität dieser. Besonders die Ereignisgesteuerte Entwicklung von Software, mit Hilfe herkömmlicher Schnittstellen gestaltet sich häufig schwierig.

Durch Entwicklung einer neuartigen Schnittstelle wollen wir es einfacher machen innerhalb des SmartMonitoring Umfelds auf Ereignisse zu reagieren und damit die Abläufe innerhalb anderer Projektkomponenten verbessern. Die neu zu entwickelnde Schnittstelle soll es Entwicklern aller Bereiche einfach machen relevante Ereignisse zu definieren und zu registrieren.

Themenbeschreibung

Das SmartMonitoring bietet im Rahmen einer REST-API eine Vielzahl von Schnittstellen für verschiedenste Objekte. Diese REST-API soll um ein Hook-System erweitert werden. Dieses Hook-System soll Ereignisse registrieren und Dritte automatisch über das Ereignis benachrichtigen. Prinzipiell soll es möglich sein jede bestehende REST-Schnittstelle an das Hook-System anzugliedern, so soll beispielsweise sowohl auf die Registrierung neuer Benutzer, als auch die Änderung von Messdaten reagiert werden können.

Ausgehend soll das Hook-System den Versand von Daten via HTTP als auch den Aufruf von Projektinternen Java-Funktionen ermöglichen. Im Fall des Versands der Daten via HTTP müssen die zu übertragenden Daten konfigurierbar sein. So soll es beispielsweise möglich sein den Nutzernamen eines neu angelegten Benutzer zu versenden. Auch die Konfiguration des HTTP-Headers und der URL-Parameter soll ermöglicht werden. Beim Aufruf von Java-Funktionen sollen passende Parameter übergeben werden können.

Die Ergebnisse der Übertragung sollen dokumentiert werden um eine Möglichkeit zu schaffen auf unterschiedliche Übertragungsergebnisse zu reagieren. Sollte der Empfänger der Daten beispielsweise kurzfristig nicht erreichbar sein, könnte man die Daten nach dem Ablauf einer definierten Zeit erneut senden.

Die Definition neuer Ereignisse soll Annotationsbasiert stattfinden. Eine REST-Schnittstelle soll also mit einer einfachen Annotation versehen werden können um zukünftig über eintretende Ereignisse informiert zu werden.

Organisation

Vorstellung des Projektteams

Benedikt Wiest

Benedikt ist 23 Jahre jung und studiert Informatik im 5. Fachsemester an der Fachhochschule Bielefeld am Campus Minden. Neben seinem Studium arbeitet Benedikt als Softwareentwickler bei einem mittelständischen Maschinenbauunternehmen. Seine Erfahrungen stammen sowohl aus den zurückliegenden Modulen im Studium, als auch aus seiner beruflichen Erfahrung.

Benedikt fungiert als Projektleiter, des Weiteren wird er sich um die Programmierung des Frontends kümmern. Als Aushilfe kann er auch kurzfristig im Backend eingesetzt werden.

Steffen Dorsch

Genauso wie Benedikt studiert Steffen am Campus Minden Informatik im 5. Fachsemester. Auch er kann auf berufliche Erfahrung im Bereich Softwareentwicklung zurückgreifen, da er neben dem Studium bei einem Werkzeugmaschinen Hersteller arbeitet. Auch er kann auf die Erfahrung aus den Grundlagen Modulen zurückgreifen.

Innerhalb des Projekts wird Steffen als Frontend-Entwickler eingesetzt.

Luca Berneking

Der 22-jährige Luca studiert Informatik an der Fachhochschule Bielefeld am Campus Minden. Neben seinem Studium arbeitet Luca als Softwareentwickler bei einem bekannten Anbieter für Hosting Lösungen. Auch er konnte Erfahrungen in den zurückliegenden Modulen im Bereich Webentwicklung sammeln.

Luca wird im Projektverlauf das Backend entwickeln und sich fortlaufend um die Pflege der Versionsverwaltung kümmern. Außerdem übernimmt Luca die Verantwortung für die Integration der Software in das bestehende System.

Julien Riechmann

Auch der 22-jährige Julien studiert im 5. Fachsemester Informatik an der Fachhochschule Bielefeld – Campus Minden. Erfahrungen konnte er bereits in den Grundlagen Modulen im Studienverlauf sammeln.

Innerhalb des Projektteams wird Julien sich um die Entwicklung der Datenbank und im späteren Verlauf auch um die Entwicklung des Backends kümmern.

Schnittstellen im Team und Kommunikation

Für direkte und schnelle Kommunikation setzen wir eine WhatsApp-Gruppe ein. Hier werden Lösungen gefunden für dringende Probleme. Des Weiteren werden hier organisatorische Angelegenheiten kommuniziert.

Als virtuelles Team halten wir Freitags und Montags unsere Meetings auf einem Discord-Server ab. Hier hat jedes Teammitglied die Möglichkeit Probleme anzusprechen und relevante Themen zu diskutieren. Darüber hinaus nutzen wir das Meeting am Montag stets zur Vorbereitung der Wochenbesprechung mit dem Auftraggeber und zur allgemeinen technischen Abstimmung. Freitags findet eine interne Wochenbesprechung statt bei welcher die Implementierungsfortschritte besprochen und neue Aufgaben verteilt werden.

Schnittstellen zum Auftraggeber

Die primäre Schnittstelle zum Auftraggeber ist der wöchentliche Besprechungstermin am Dienstag. Bei Bedarf greift das Team auf eine Kommunikation via E-Mail zurück. Außerdem besteht das Angebot des Auftraggebers bei Rückfragen persönlich in Büro vorstellig zu werden.

Vorgehensmodell

Als agiles Vorgehensmodell setzen wir Scrum ein. Die Sprintlänge setzen wir auf 7 Tage fest. Als Scrum-Master und Moderator fungiert der Projektleiter. Als zentrales Umsetzungselement setzen wir das in GitLab integrierte Taskboard ein. Das Product Backlog wird ebenfalls im GitLab gepflegt.

Arbeitsplan

Der Arbeitsplan wurde vor Projektbeginn durch alle Projektmitglieder erstellt und enthält eine detaillierte Aufstellung der vorzunehmenden Arbeiten auf Basis von Kalenderwochen. Jedes Projektmitglied kennt so zu jedem Zeitpunkt seine Aufgabe. Ein Ziel der detaillierten Arbeitsplanung ist die Steuerung der Arbeitsergebnisse zu definierten Zeitpunkten.

Arbeitsplan	Benedikt Wiest	Steffen Dorsch	Luca Berneking	Julien Riechmann
KW 43	Vorbereitung Feinkonzept, Erstellung Sequenzdiagramme, Erstellung Klassendiagramm, Organisation	Definition der Use-Cases, Erstellung Use-Case-Diagramme, Erstellung UI-Mockups	Planung Architektur, Erstellung Klassendiagram, Erstellung Architekturübersicht	Planung Datenmodell, Erstellung ER-Diagramm

Im abgebildeten Beispiel ist unsere Arbeitsplanung für die Kalenderwoche 43 in 2019 zu sehen. In den jeweiligen Spalten befinden sich Informationen zu den Aufgaben der einzelnen Projektmitgliedern. Als Basis der Planung wurde das festgesetzte Abgabedatum des Feinkonzepts in Kalenderwoche 44 in 2019 angenommen. Die hierfür benötigten Artefakte (in Form von konzeptionellen Inhalten) wurden gleichmäßig auf die Teammitglieder aufgeteilt. Durch die Planung konnte sichergestellt werden das alle Inhalte zu Kalenderwoche 44 fertiggestellt wurden.

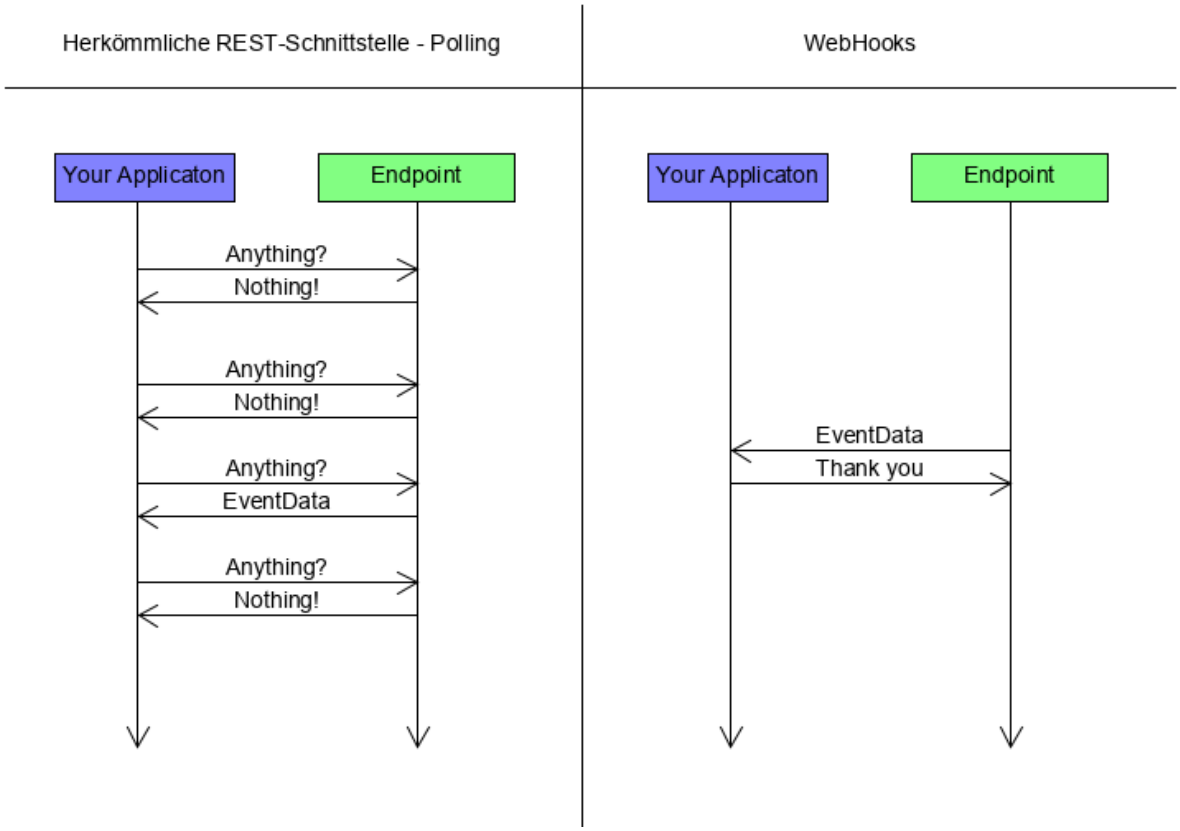
Im Projektverlauf haben wir festgestellt das die sehr detaillierte Arbeitsplanung auch Nachteile mit sich bringt. So konnten wir kurzfristig benötigte Ressourcen nicht schaffen und mussten mehrfach kurzfristig umplanen. Für zukünftige Projekte würden wir erneut eine detaillierte Arbeitsplanung erstellen, uns allerdings mehr Zeit für spontane Anforderungen einplanen.

Theoretische Grundlagen

WebHooks

WebHooks als solches sind ein nicht-standardisiertes Verfahren zur Kommunikation von Servern, Diensten und Nutzern dieser. Das Verfahren wird im Rahmen von verteiltem Rechnen und Nachrichtenorientierter Middleware genutzt. WebHooks ermöglichen es einem Teilnehmer mitzuteilen das ein definiertes Ereignis eingetreten ist damit diese eine damit verbundene Aktion auslösen kann. Wird eine Teilnehmer über ein Ereignis informiert, muss dieser kein klassisches Polling an der Schnittstelle betreiben, diese Tatsache reduziert Traffic zwischen Anwendungen und erleichtert die Ereignisgesteuerte Programmierung von angeschlossenen Anwendungen.

WebHooks werden außerdem als einfaches Callback-Verfahren im Bereich Daten-Synchronisierung, dezentralen Berechnungen und zur Validierung eingesetzt. Die wesentlichen Anwendungsbereiche sind jedoch Event-Notifications, Weiterleitung von Daten (Pipes) und Anbindung von Applikationen/Plugins.



In der Abbildung ist die Funktionsweise von WebHooks in stark vereinfachten Sequenzdiagrammen dargestellt. Im linken Bereich ist eine Anwendung mit Anbindung an eine herkömmliche REST-API des Endpoints skizziert. Die Anwendung muss Polling betreiben um Informationen über Ereignisse zu erhalten, daher schickt sie in regelmäßigen Abständen Anfragen an den Endpoint stellen. Um das Ereignis zu erhalten stellt Sie 4 Anfragen. Im rechten Bildabschnitt sehen wir dieselbe Anwendung, welche mit Hilfe einer WebHook angebunden ist. Der Endpoint benachrichtigt die Anwendung direkt

nach Eintritt des Ereignisses. Zu beachten ist nicht nur die eingesparte Nachrichtenlast im Netzwerk und die geringere Last des Endpoints, sondern auch der Zeitvorteil bei eintreten des Ereignisses.

Definition von Bezeichnungen

WebHooks als solches sind kein standardisiertes Verfahren, sondern eher eine lose Sammlung von Pattern und Abläufen. Daher führen wir eigene Bezeichnungen ein, um Missverständnisse bei der Kommunikation zu vermeiden.

Eine **Hook** ist ein abstrakter Begriff welcher das Konstrukt aus auslösendem **Event** und **Subscriptions** (individuelle Konfigurationen pro Empfänger) vereint.

Ein **Event** meint ein Ereignis welches im SmartMonitoring Kontext stattfindet. Der Eingang neuer Sensordaten kann beispielsweise ein Event sein, sofern dieses entsprecht implementiert wurde.

Eine **Subscription** beinhaltet die individuelle Konfiguration für den Empfänger. In einer Subscription wird beispielsweise das Event, die Zieladresse für den Versand von Daten, als auch das Verhalten bei eintreten des Events gespeichert. Als Analogie kann das abonnieren (engl. „subscribe“) einer herkömmlichen Zeitung gesehen werden – der Herausgeber speichert zum die Empfängeradresse und Zahlungsinformationen.

Technologien

HTML

HTML steht für HyperText Markup Language. Es handelt sich dabei um eine textbasierte Auszeichnungssprache zur Strukturierung von digitalen Dokumenten. HTML-Dokumente bilden die Grundlage des World Wide Web.

CSS

CSS steht für Cascading-Style-Sheets und ist eine Möglichkeit für HTML-Dokumente, den Inhalt einer Seite von den Designanweisungen der einzelnen Elemente, wie zum Beispiel Überschriften, Zitaten etc, zu entkoppeln.

Man kann eine CSS-Datei für eine Domain erstellen und diese auf allen Unterseiten als externe Ressource einbinden. Dies kann viel Zeit sparen, wenn feste Vorgaben für meine Designelemente vorhanden sind, die sich zwischen Dokumenten nicht ändern.

Der kaskadierende Teil wird dann klar, wenn man sich mit verschiedenen Klassen beschäftigt. So kann man in der CSS-Datei z.B. festlegen, dass alle H2-Überschriften (Eltern-Element) in Schriftgröße 46 ausgespielt werden und dann eine Unterklasse der H2-Überschriften einfügen (Kind-Element) die den Text fett druckt, wenn eine bestimmte Klasse (zB. „fett“) definiert wird.

SWAC

SWAC steht für SmartWebApplicationComponents und versteht sich als Framework für UI-Komponenten. Die bereitgestellten Komponenten werden im SmartMonitoring genutzt. Im Vergleich zu

anderen UI-Frameworks kann SWAC vor allem durch einfache Bedienung, Skalierbarkeit und Datadriven-Development überzeugen.

REST-API

REST steht für REpresentational State Transfer, API für Application Programming Interface. Gemeint ist damit ein Programmierschnittstelle, die sich an den Paradigmen und Verhalten des World Wide Web (WWW) orientiert und einen Ansatz für die Kommunikation zwischen Client und Server in Netzwerken beschreibt.

Payara-Server

Der Payara Server ist eine offene Plattform, die sichere und zuverlässige Entwicklungen von Java unterstützt. Es handelt sich dabei um einen Webserver.

JavaScript

JavaScript ist eine interpretiert Sprache welche ursprünglich für das dynamische Laden von Webseiteninhalten entwickelt wurde, heute jedoch auch in vielen Browserfernen Bereichen Anwendung findet.

Java

Java ist eine objektorientierte, betriebssystemunabhängige Programmiersprache welche 1995 durch Sun Microsystems eingeführt wurde. Java gehört heute zu den weitverbreitetsten Programmiersprachen weltweit und wird in nahezu allen Bereichen genutzt.

PostgreSQL

PostgreSQL ist ein freies objektrelationales Datenbankmanagementsystem. PostgreSQL ist weitestgehend konform mit dem SQL-Standard SQL:2011. Es ist in den meisten Linux-Distributionen enthalten und wird auch auf Apple im Betriebssystem Mac OS genutzt.

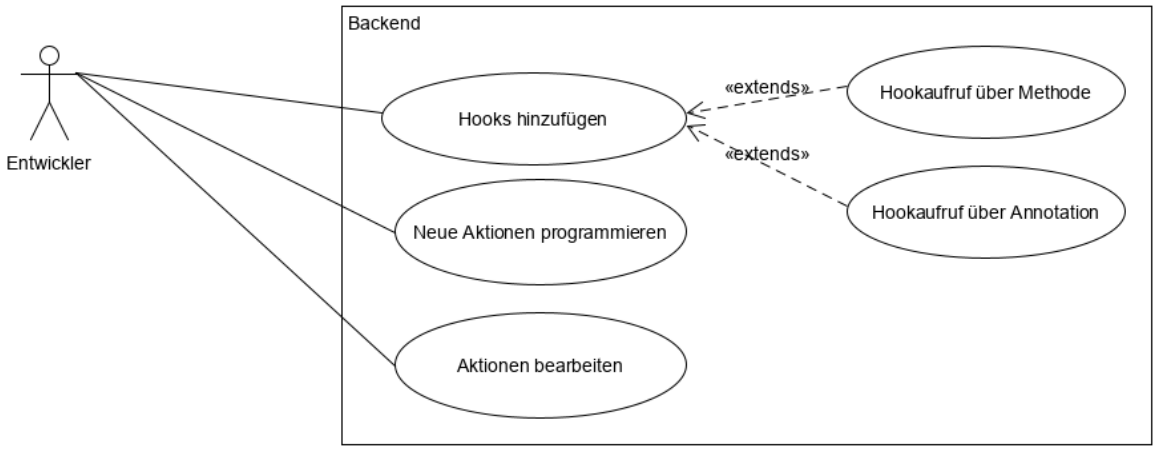
JPA

Die Java Persistence API (JPA) ist eine Schnittstelle für Java-Anwendungen, die die Zuordnung und die Übertragung von Objekten zu Datenbankeinträgen vereinfacht. Sie vereinfacht die Lösung des Problems der objektrelationalen Abbildung, das darin besteht, Laufzeit-Objekte einer Java-Anwendung über eine einzelne Sitzung hinaus zu speichern (Persistenz).

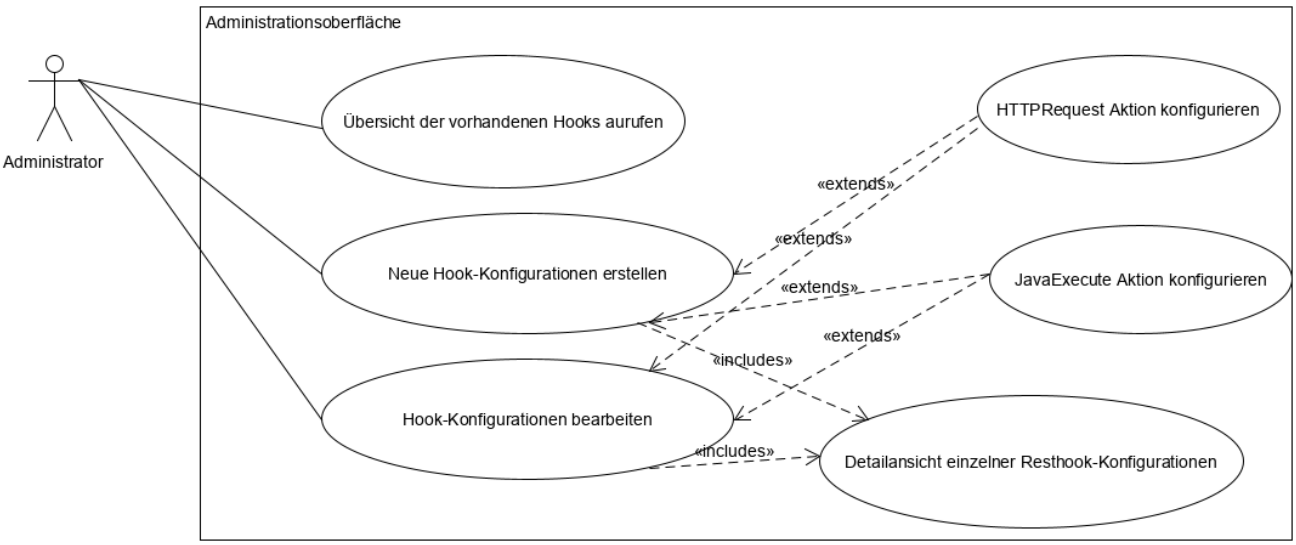
Konzeptionelle Arbeiten

Use-Case Diagramme

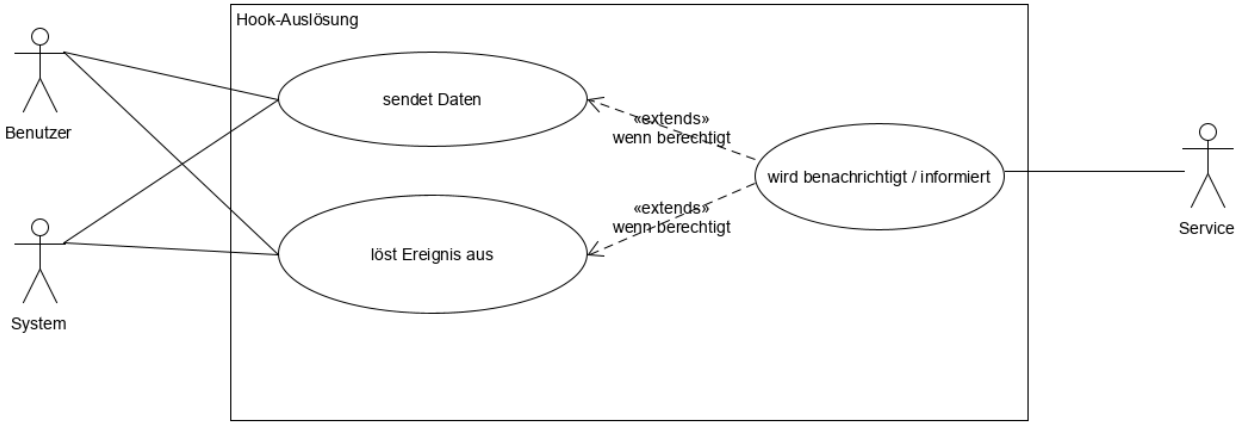
Im Backend soll der Entwickler Hooks hinzufügen können, entweder über den expliziten Aufruf der Methode hookManager.triggerHook() oder über die Annotation @HookEvent. Zudem kann der Entwickler neue Aktionen, die von Hooks ausgelöst werden können implementieren, oder vorhandene Aktionen bearbeiten.



In der Administrationsoberfläche soll der Administrator eine Übersicht aller bereits konfigurierten Hooks aufrufen können. Diese bereits existierenden Konfigurationen, soll er bearbeiten können. Auch neue Konfigurationen können vom Administrator erstellt werden. Beides öffnet eine Detailansicht der selektierten Konfiguration, bei der eingestellt werden kann, ob es sich um eine HTTPRequestAction oder um eine JavaExecuteAction handelt.

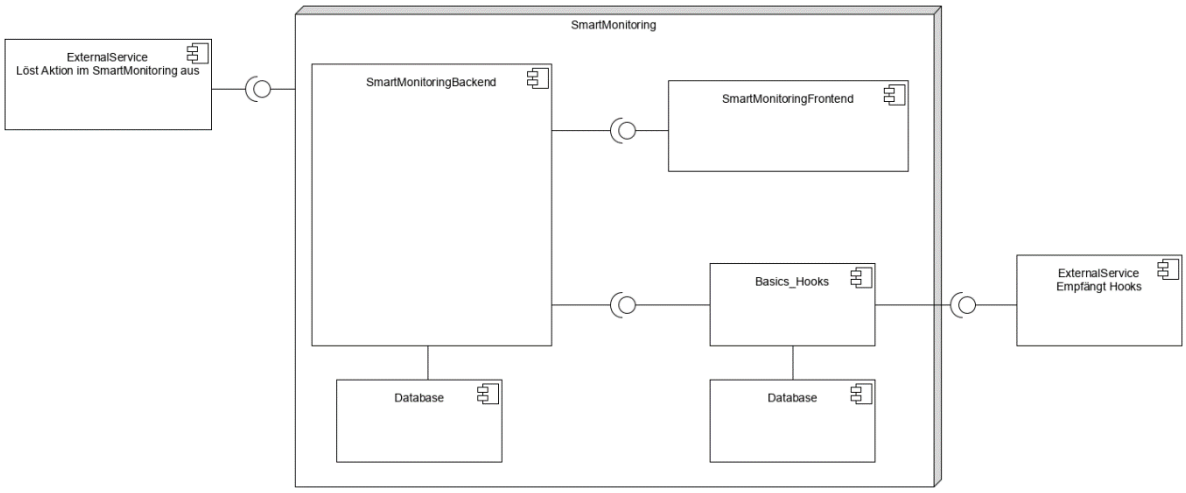


Um eine Hook auszulösen senden Benutzer, oder ein angeschlossenes System, Daten und löst damit ein Ereignis im Backend aus. Darüber wird ein konfigurierter dritter Service benachrichtigt, falls dies nicht durch fehlende Berechtigungen oder andere Filter, die der Service nicht erfüllt, verhindert wird.



Software-Architektur

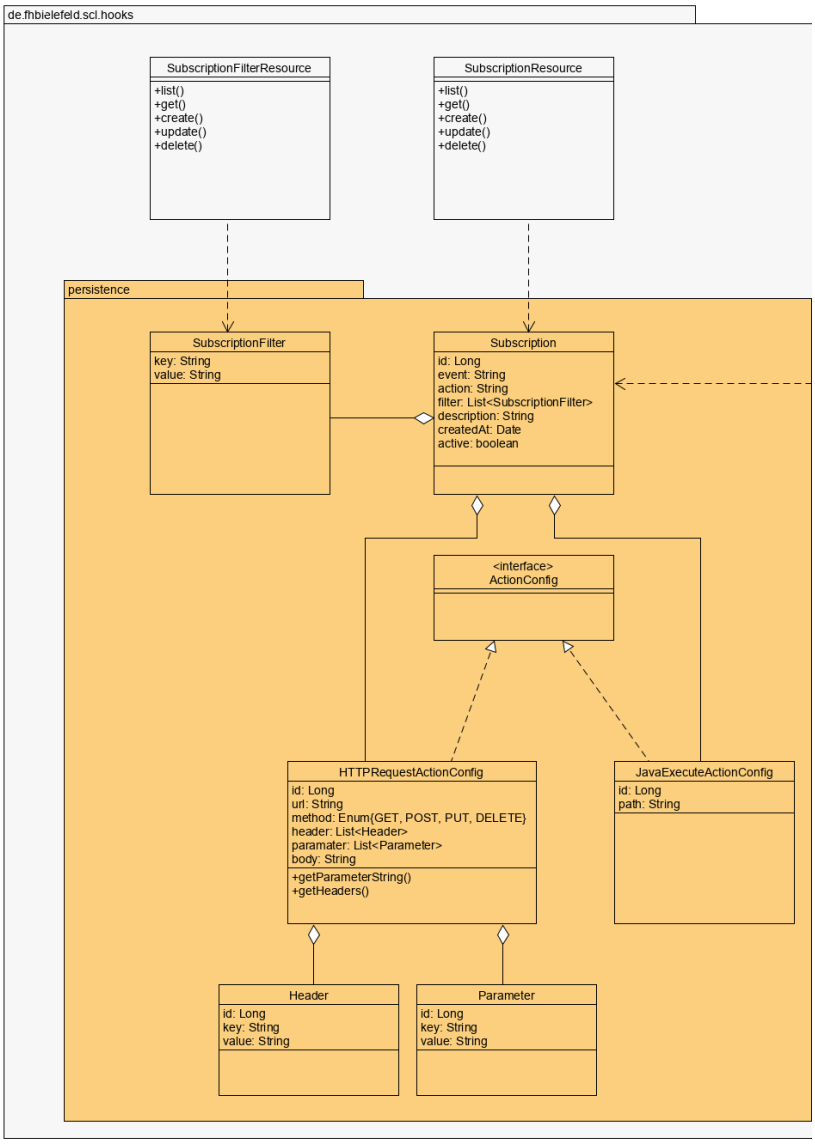
Das Smartmonitoring System wird durch die neue Library "Basic_Hooks" ergänzt. Diese kümmert sich um die Verwaltung der angelegten Subscriptions über eine REST-Schnittstelle. Die Library legt ihr eigenes Datenbank-Schema an. Die Library wird bei einer Aktion, welche beispielsweise durch einen externen Service ausgeführt wird, von dem bestehenden Backend des SmartMonitoring aufgerufen. Die neu entwickelte Libary sendet anschließend automatisiert die gewünschten Daten einen weiteren Service. Die Administrationsoberfläche gliedert sich in das bestehende SmartMonitoring Frontend ein.



Klassendiagramm

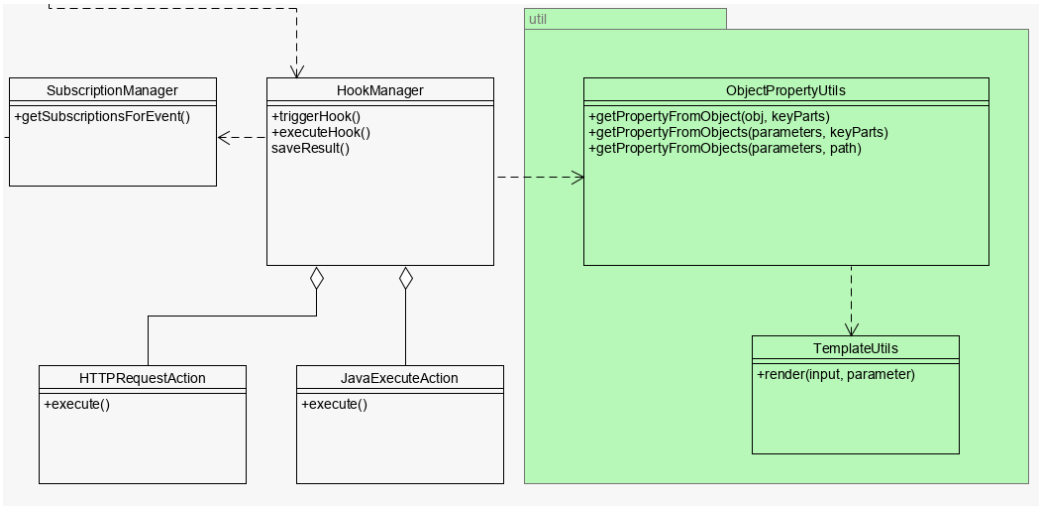
Alle in dem Projekt erstellen Klassen befinden sich innerhalb des „de.fhbielefeld.scl.hooks“ Paketes. Das Subpackage „persistence“ beinhaltet hier die Klassen die durch JPA in der Datenbank persistiert werden. Die wichtigste Klasse ist hier die Subscription, alle weiteren Klassen sind mit dieser aggregiert. So kann eine Subscription mehrere SubscriptionFilter haben. Die detailiertere Konfiguration der Aktion befindet sich in den „ActionConfig“ Klassen „HTTPRequestActionConfig“ und „JavaExecuteActionConfig“. Eine Subscription kann beide gleichzeitig beinhalten, so geht eine vorherige Konfiguration bei einem Wechsel des Typs nicht verlohren. Der aktivierte Aktionstyp wird in dem „action“ Attribut hinterlegt.

Zum Erstellen, Anzeigen und Bearbeiten der Subscriptions bieten die Klassen „SubscriptionResource“ und „SubscriptionFilterResource“ eine Rest-Schnittstelle an. Diese wird vom SmartMonitoringFrontend angesprochen.

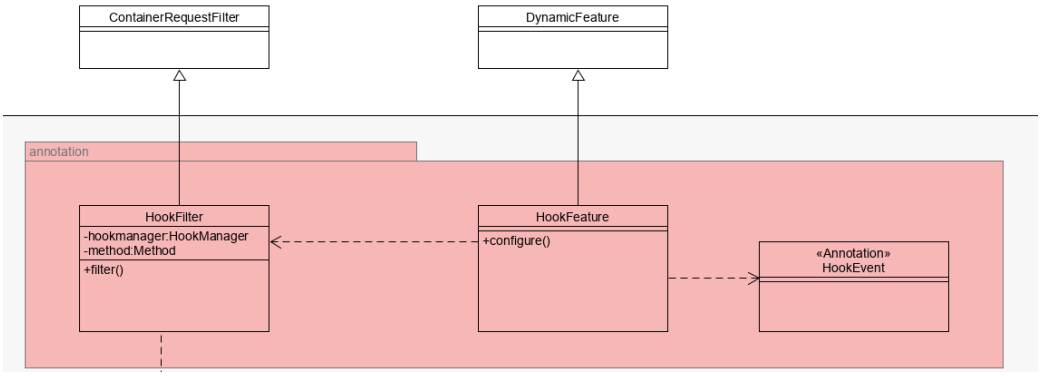


Der „HookManager“ kümmert sich um die Ausführung der Hooks. Zum Auftreten eines Events wird die triggerHook Methode ausgeführt. Diese lädt die entsprechenden Subscriptions mit Hilfe des „SubscriptionManager“ aus der Datenbank. Wenn alle weiteren Filter zutreffen wird durch die „executeHook“ Methode jeweils die Komfigurierte Aktion ausfegührt. Möglich sind hier aktuell das Ausführen einer statischen Methode über die „JavaExecuteAction“ oder das Absenden eines HTTP Requests über die „HTTPRequestAction“.

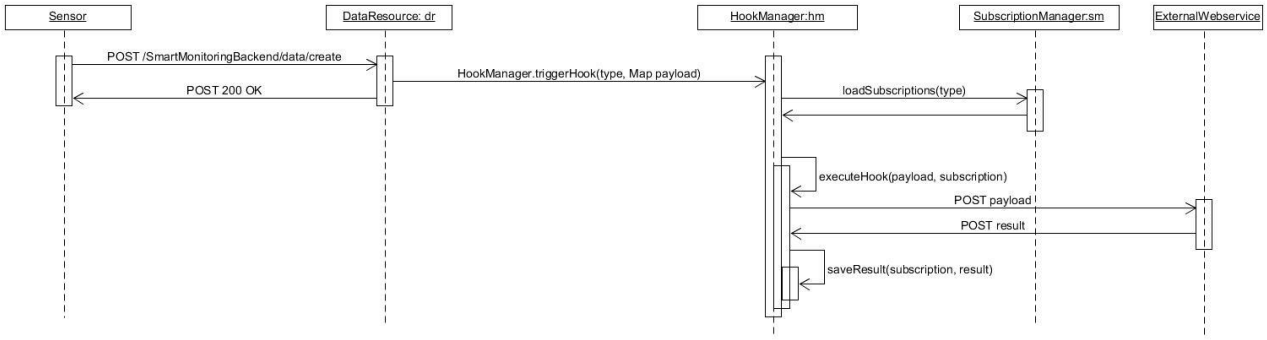
Durch das Subpackage „util“ werden hier mögliche Platzhalter ersetzt. Zunächst bietet die Klasse „ObjectPropertyUtils“ Methoden zum Zugriff auf Properties eines beliebigen Objektes an. Diese wird von der „TemplateUtils“ Klasse zum eigentlichen erstetzen der Platzhalter verwendet.



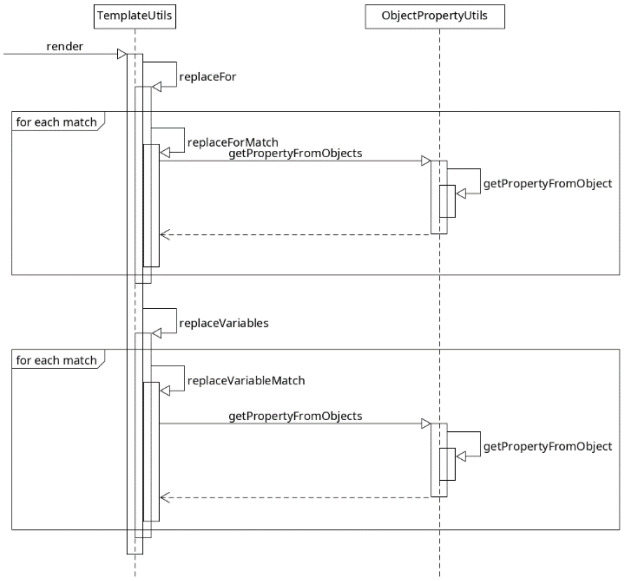
Das Subpackage „annotation“ sogt für die Auslösuing von Events über die „@HookEvent“ Annotation. Die „configure()“ Methode der Klasse „HookFeature“ wird beim Start des Backends für jede Rest-Methode in gesammten Projekt ausgeführt. Ist eine Methode mit der Annotation versehen wird dieser der „HookFilter“ als weitere Middleware hinzugefügt. Erfolgt nun ein Rest-Request an diese wird durch die „filter()“ Methode die „triggerHook“ Funktion des HookManagers aufgerufen.



Sequenzdiagramm



Im oben zu sehenden Sequenzdiagramm ist das Auslösen von Hooks und das Versenden von Daten abgebildet. Ein Sensor sendet neue Daten über die bestehende REST-API des SmartMonitoring Backends. Die dort enthaltene Funktion create() führt den Befehl HookManager.triggerHook() aus und löst damit die Hook aus. Innerhalb der Klasse HookManager werden als erstes alle eingetragenen Subscriptions, über den Funktionsaufruf loadSubscriptions() geladen. Anschließend werden die Hooks, mit den dazugehörigen Subscriptions ausgeführt und die Daten an einen externen Webservice gesendet. Das Übertragungsergebnis wird mit der Funktion saveResult() geloggt.



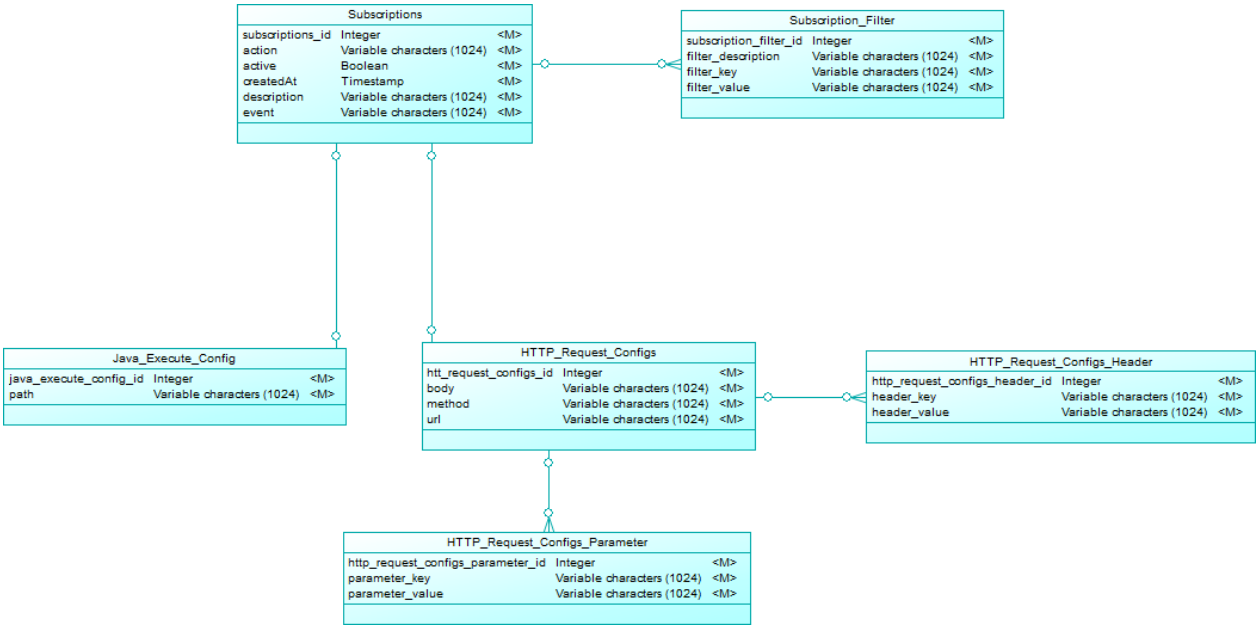
Die Klasse "TemplateUtils" übernimmt das Rendern eines beliebigen Template Strings. Dafür werden zunächst die Scheifen (mit replaceFor) und anschließend die Variablen (mit replaceVariables) ersetzt. Jeder gefundene Platzhalter wird mit Hilfe der Methoden "replaceForMatch()" und "replaceVariableMatch()" ersetzt. Zum Zugriff auf die Objekt Properties wird die "ObjectPropertyUtils" Klasse verwendet. Der Zugriff auf diese erfolgt über die "getPropertyFromObjects" Methode welcher zunächst alle Parameter übergeben werden. Diese wählt den richtigen Parameter aus und übergibt diesen an die innere Methode "getPropertyFromObject()". Nachdem alle Platzhalter ersetzt wurden, kann das Ergebnis zurückgegeben werden.

Datenbankdesign

Die Datenbank ist in einem eigenen Schema angelegt und kann damit losgelöst von der Datenbank des SmartMonitoring-Systems betrieben werden.

Die Tabelle "Subscriptions" speichert die angelegten Subscriptions. Jede Subscription kann keine, eine, oder mehrere Filter haben, welche in der Tabelle "Subscription_Filter" gespeichert werden.

Jede Subscription besitzt entweder eine "Java_Execute_Config" oder eine "HTTP_Request_Configs", wobei die "subscription_id" als Foreign Key in der jeweiligen Tabelle gespeichert wird. Eine HTTP_Request_Config kann keine, eine, oder mehrere Header oder Parameter besitzen. Die Entities "HTTP_Request_Configs_Parameter" und "HTTP_Request_Configs_Header" beinhalten die ID ihrer HTTP Request Configuration als Foreign Key.



Code-Ausschnitte

Frontend

Im Frontend nutzen wir die Komponente Present um alle angelegten Subscriptions anzuzeigen, im unten stehenden Ausschnitt laden wir mit der Komponente die Daten aus der Variable subscriptionData.

```
<div id="subscription-table" swa="swac_present FROM subscriptionData
TEMPLATE table_for_all_datasets"></div>
```

Die globale Variable subscriptionData wird mit Daten aus dem Backend gefüllt. Hierzu rufen wir die interne Schnittstelle auf und laden alle Subscriptions. Im Fehlerfall wird der Vorgang abgebrochen und der Benutzer erhält eine Benachrichtigung.

```
function getSubscriptions() {
        fetch(subscriptionListURL)
        .then(res => { if (!res.ok) { throw Error(response.statusText); }
else { return res.json(); } })
        .then((subscriptionArr) => {
          for (let i = 0; i < subscriptionArr.length; i++) {
                subscriptionData[i].Name = subscriptionArr[i].description;
                if (subscriptionArr[i].active) { subscriptionData[i].Aktiv =
activeCheckboxEnabled; } else { subscriptionData[i].Aktiv =
activeCheckboxDisabled; }
                subscriptionData[i].Bearbeiten =
subscriptionEditButton.replace(/{ID}/g, subscriptionArr[i].id);
        }}).catch(err => {
                SWAC_debug.addErrorMessage('hookmanagement.js', err + "
thrown when trying to get subscriptions:");
                sendMessageToUser("Fehler beim Laden der Subscriptions: " +
err);
});}
```

Im Ausschnitt unten sind die Variablen zu sehen, welche uns das dynamische Laden der HTML-Elemente innerhalb der Tabelle ermöglichen. Die entsprechenden Werte werden zur Laufzeit für die Platzhalter {Placeholder} eingesetzt.

```
const activeCheckboxEnabled = '<input class="uk-checkbox uk-disabled"
type="checkbox" checked/>';
const activeCheckboxDisabled = '<input class="uk-checkbox uk-disabled"
type="checkbox"/>';
const subscriptionEditButton = "<button type='button' uk-toggle='target:
#subscription-detail' onclick='getSubscriptionDetails({ID})' class='uk-
button uk-button-default uk-button-primary' uk-tooltip='' style='width:
100%'>Bearbeiten</button>";
const subscriptionDeleteButton = "<button type='button' uk-toggle='target:
#delete-modal' onclick='setDeleteBTNEvent(\"deleteSubscription({ID})\")'
class='uk-button uk-button-default uk-button-primary' uk-tooltip=''
style='width: 100%'>L╪schen</button>";
```

Innerhalb des Frontend nutzen wir zur Löschung nur ein Modal, welches sich Abhängig vom Typ des Objekts unterschiedlich verhält. Mit Hilfe der Funktion setDeleteBTNEvent() können wir das Verhalten individuell anpassen, in dem wir die Funktion des Buttons auf die „Löschen“-Funktion des Objekts setzen.

```
function setDeleteBTNEvent(onClick) {
        let btn = document.getElementById("deleteBTN_Modal");
        btn.setAttribute("onClick", "javascript:" + onClick);
}
```

Ein Wrapper der UIkit.notification() Methode, mit der beliebige Benachrichtigungen an den Benutzer gesendet werden können, welche sich am oberen Bildschirmrand rechts sammeln und nach 5 Sekunden automatisch ausgeblendet werden. Der Wrapper wird genutzt, damit man lediglich eine Methode mit dem gewünschten Text aufrufen muss, ohne sich um andere Parameter wie Position und Verhalten der Benachrichtigung kümmern zu müssen. Zudem wird so sichergestellt, dass alle Benachrichtigungen an den Benutzer ein persistentes Aussehen und Verhalten aufweisen.

```
function sendMessageToUser(msg) {
        UIkit.notification({
                message: msg,
                status: 'primary',
                pos: 'top-right',
                timeout: 5000
        });
}
```

Nachdem eine Änderung an den Subscriptions vorgenommen wurde, kann mithilfe dieser Funktion die Darstellung aktualisiert werden. Zuerst werden die neuen Daten aus dem Backend geladen und in die Variable subscriptionData gespeichert, danach die veraltete Tabelle gelöscht. Zuletzt wird die SWAC-Present-Komponente mit den abgerufenen Daten erneut initialisiert, hierzu nutzen wir die Methode SWAC.detectRequestors(). Dies ist ein Workaround für die SWAC.reload() Methode, welche zum Zeitpunkt der Implementierung noch nicht zur Verfügung stand. Sobald SWAC.reload() implementiert wurde, sollte hier darauf zurückgegriffen werden, da das Löschen der alten Tabelle dann entfallen würde und die Tabelle einfach aktualisiert werden könnte.

```
function refreshSubscriptions() {
        getSubscriptions();
        let table = document.getElementById("subscription-outer");
        table.innerHTML = '';
        table.innerHTML = subscriptionTableHTML;
        SWAC.detectRequestors();
}
```

Untenstehend ein Ausschnitt aus der Funktion zum Löschen einer Subscription. Zuerst wird eine DELETE-Anfrage an die Route im Backend geschickt, mit der ID der zu löschenden Subscription als Parameter. Ist diese Anfrage erfolgreich, wird der Benutzer über das erfolgreiche Löschen der Subscription benachrichtigt und die Darstellung der Subscriptions wird aktualisiert. Fehler bei der Anfrage ans Backend führen zum Abbruch mit einer Benachrichtigung der Fehlerinformationen an den Benutzer.

```
function deleteSubscription(id) {
        fetch(subscriptionDeleteURL + "/" + id, { method: 'DELETE'
        }).then(() => {
                sendMessageToUser('Hook mit ID: ' + id + ' erfolgreich
gelöscht!');
                refreshSubscriptions();
        }).catch(err => {
                SWAC_debug.addErrorMessage('hookmanagement.js', err + "
thrown when trying to delete subscription. ID: " + id);
        });
}
```

Einlesen einer gegebenen Subscription aus dem Backend anhand ihrer ID. Die Subscriptiondetails werden aus der Antwort des Backends geparsed, anschließend wird das Subscription-Modal mit den zugehörigen Daten gefüllt und die Benutzeroberfläche dynamisch aufgebaut. Die Button-Eventistener werden auf die Funktion zum Speichern und Löschen der geladenen Subscription gesetzt. Fehler bei der Anfrage ans Backend führen zum Abbruch mit einer Benachrichtigung der Fehlerinformationen an den Benutzer.

```
function getSubscriptionDetails(id) {
        fetch(subscriptionDetailURL  + "/" + id)
        .then(res => {
                let subscriptionDetails = res.json();
                let subscriptionName =
document.getElementById("subscription_name");
                subscriptionName.value = subscriptionDetails.description;
                let subscriptionHTTPHeader =
document.getElementById("subscription_http_header");
                let header = subscriptionDetails.config.header;
                for (let i = 0; i < header.length; i++)
                {
                        let key = header[i].key; let value =
header[i].value;
                        if (header[i].key != null && header[i].key != "") {
// empty headers will be ignored
                                headerSwitcher +=
headerSwitcherHTML.replaceAll("{key}", key).replaceAll("{value}", value);
                                headerSwitcherContent +=
headerSwitcherContentHTML.replaceAll("index",
i.toString()).replaceAll("{key}", key).replaceAll("{value}", value);
                        }
                }
                subscriptionFilter.setAttribute("onClick", "javascript:" +
"getFilters(" + id + ")");
                subscriptionFilter.setAttribute("uk-toggle", "target:
#filter-edit");
                saveButton.setAttribute("onClick", "javascript:
saveSubscription(" + id + ")");
```

```
        }).catch(err => { SWAC_debug.addErrorMessage('hookmanagement.js',
err + " thrown when trying to get subscription details. ID: " + id); });
```

Da wir zur Änderung und zur Neuanlage einer Subscription das gleiche Modal benutzen, muss dieses bei Neuanlage zuerst geleert werden. Zudem wird eine Benachrichtigung für den Benutzer eingebaut, für den Fall, dass dieser Filter anlegen möchte bevor die Subscription erstellt wurde. Dies ist erst nach der ersten Speicherung, also Erstellung der Subscription, möglich.

```
function resetSubscriptionDetails() {
                let subscriptionName =
document.getElementById("subscription_name");
                subscriptionName.value = "";
                let subscriptionAction =
document.getElementById("subscription_actionswitcher");
                UIkit.switcher(subscriptionAction).show(1);
                subscriptionFilter.setAttribute("onClick", "javascript:
sendMessageToUser('Bitte Speichern, bevor Filter angelegt werden
können!')");
                subscriptionFilter.setAttribute("uk-toggle", "target:
locked");
                let saveButton =
document.getElementById("subscription_save");
                saveButton.setAttribute("onClick", "javascript:
createSubscription()");
}
```

Anlegen einer neuen Subscription; Daten werden aus den Eingabefeldern des Frontends übernommen und an das Backend per POST-Anfrage geschickt. HTTP-Header und -Parameter sollen in einem Schritt beliebig angelegt und gelöscht werden können (leere Felder werden gelöscht, alle anderen in die Datenbank übernommen). Bei erfolgreicher Anfrage an das Backend wird der Nutzer über den Erfolg benachrichtigt und die Subscription Übersicht aktualisiert. Fehler bei der Anfrage ans Backend führen zum Abbruch mit einer Benachrichtigung der Fehlerinformationen an den Benutzer.

```
function createSubscription() {
        let subscriptionJavaMethod =
document.getElementById("subscription_java_method");
        let path = subscriptionJavaMethod.value;
        let i = 0; let finished = false;
        while (!finished) {
                let subscriptionHeaderKey =
document.getElementById("subscription_header_key_" + i);
                let subscriptionHeaderValue =
document.getElementById("subscription_header_value_" + i);
                if (subscriptionHeaderKey != null && subscriptionHeaderValue
!= null) {
                        let headerKey = subscriptionHeaderKey.value;
                        let headerValue = subscriptionHeaderValue.value;
                        if (headerKey != "") { //ignore empty headers
                                requestJSON.config.header[i].key = headerKey;
                                requestJSON.config.header[i].value =
headerValue;
                        } i++;
                } else { finished = true; }
        }
```

```
        if (subscriptionHTTPRequest.className === "uk-active") { action =
"HTTPRequest";
        } else { action = "JavaExecute"; }
        requestJSON.action = action;
        fetch(subscriptionCreateURL, { method: "POST", headers: {"Content-
Type": "application/json", "Access-Control-Origin": "*"}, body:
JSON.stringify(requestJSON)
        }).then(function(response){
                sendMessageToUser("Successfully created subscription " +
description);
                refreshSubscriptions();
        })
        .catch(err => {
                SWAC_debug.addErrorMessage('hookmanagement.js', err + "
thrown when trying to create a subscription");
        });
}
```

Speichern einer Subscription per Anfrage an das Backend. Die Daten werden aus den Eingabefeldern des Frontends ausgelesen, um per PUT-Anfrage an das Backend gesendet zu werden. Sollten Header oder Parameter angelegt worden sein, die keine Daten enthalten, werden diese ignoriert. Bei Erfolg wird der Benutzer benachrichtigt, kommt es bei der Anfrage zu einem Fehler, kommt es zum Abbruch und der Nutzer wird darüber benachrichtigt.

```
function saveSubscription(id) {
        let subscriptionName =
document.getElementById("subscription_name");
        let description = subscriptionName.value;
        let i = 0; let finished = false;
        while (!finished) {
                let subscriptionHeaderKey =
document.getElementById("subscription_header_key_" + i);
                let subscriptionHeaderValue =
document.getElementById("subscription_header_value_" + i);
                if (subscriptionHeaderKey != null && subscriptionHeaderValue
!= null) {
                        let headerKey = subscriptionHeaderKey.value;
                        let headerValue = subscriptionHeaderValue.value;
                        if (headerKey != "") { // ignore empty headers
                                requestJSON.config.header[i].key = headerKey;
                                requestJSON.config.header[i].value =
headerValue;
                        } i++;
                } else { finished = true; }
        }
        if (subscriptionHTTPRequest.className === "uk-active") { action =
"HTTPRequest";
        } else { action = "JavaExecute"; }
        requestJSON.action = action;
        fetch(subscriptionUpdateURL + '/' + id, { method: "PUT", headers:
{"Content-Type": "application/json", "Access-Control-Origin": "*"}, body:
JSON.stringify(requestJSON)
        }).then(function(response){
                sendMessageToUser("Successfully updated subscription " +
description);
                refreshSubscriptions();
        }).catch(err => {
                SWAC_debug.addErrorMessage('hookmanagement.js', err + "
thrown when trying to create a subscription");});});
```

```
}
```

Fügt dynamisch einen neuen Header bei der Bearbeitung oder Erstellung einer HTTPRequest-Subscription hinzu und verschiebt den Fokus zu diesem Header. Bei der Speicherung werden nur Header berücksichtigt, die mit Inhalt gefüllt wurden.

```
function newHeader() {
	let i = 0; let finished = false;
	while (!finished) {
		let subscriptionHeaderKey =
document.getElementById("subscription_header_key_" + i);
		let subscriptionHeaderValue =
document.getElementById("subscription_header_value_" + i);
		if (subscriptionHeaderKey != null && subscriptionHeaderValue
!= null) { i++;
		} else { finished = true; }
	}
	headerSwitcherContent.insertAdjacentHTML('beforebegin',
headerNewSwitcherContent.replaceAll("{index}", i.toString()));
	let switcher =
document.getElementById("subscription_http_header").children[0];
	UIkit.switcher(switcher).show(i);
}
```

Lädt die Filter, die einer gegebenen Subscription zugeordnet sind aus dem Backend und füllt mit den Daten das Filter-Modal. Die Benutzerrollen werden zudem aus dem Backend geladen um die Optionen für die Benutzerfilterung dynamisch Anzeigen und Speichern zu können. Bei Erfolg wird der Benutzer benachrichtigt, kommt es bei der Anfrage zu einem Fehler, kommt es zum Abbruch und der Nutzer wird darüber benachrichtigt.

```
function getFilters(subId) {
	fetch(filterListURL.replace("{subId}", subId))
	.then((filterArr) => {
		const fet = fetch(parentUserURL).then(res =>
res.json()).then((parentList) => {
			parentUsers = parentList.parentList;
			for (let j = 0; j < parentUsers.length; j++) {
				parentOptions += "<option>" +
parentUsers[j].username + "(" + parentUsers[j].id + ")" + "</option>";
			}
		}).then(() => {
			for (let i = 0; i < filterArr.length; i++) {
				let key = filterArr[i].key;
				filterSwitcher.innerHTML +=
filterSwitcherHTML.replaceAll("{description}", description);
				let newfilter =
document.createElement("div");
				newfilter.innerHTML =
filterContentHTML.replaceAll("{description}",
description).replaceAll("{id}", id).replaceAll("{key}",
key).replaceAll("{value}", value).replaceAll("{subId}",
subId).replaceAll("{parentOptions}", parentOptions);
				filterContent.appendChild(newfilter);
				let switcher =
document.getElementById("filter_switcher_" + id);
```

```
                                if(key == userRole) { // it is userrole
filter
                                        UIkit.switcher(switcher).show(0);
                                        let filterRole =
document.getElementById("filter_role_" + id);
                                        filterRole.value = value;
                                } else { UIkit.switcher(switcher).show(1); }
                        }
                        filterSwitcher.innerHTML +=
filterSwitcherAddNewFilterHTML;
                        let newfilter = document.createElement("div");
                        newfilter.innerHTML =
filterContentEndHTML.replaceAll("{subId}",
subId).replaceAll("{parentOptions}", parentOptions);
                        filterContent.appendChild(newfilter);
        });}).catch(err => {
SWAC_debug.addErrorMessage('hookmanagement.js', err + " thrown when trying
to get filters for subscription: " + subId);});
}
```

Löschen eines Filters mithilfe einer DELETE-Anfrage ans Backend, benötigt werden hierfür die Subscription-ID und die ID des zu löschenden Filters als Parameter. Wurde der Filter erfolgreich gelöscht, wird der Benutzer hierüber benachrichtigt und die Anzeige der Filter wird aktualisiert. Kommt es bei der Anfrage zu einem Fehler, kommt es zum Abbruch und der Nutzer wird darüber benachrichtigt.

```
function deleteFilter(subId, id) {
        fetch(filerDeleteURL.replace("{subId}", subId) + "/" + id, {
method: 'DELETE' }).then(() => {
                sendMessageToUser('Filter mit ID: ' + id + ' erfolgreich
gelöscht!');
                refreshFilters(subId);
        }).catch(err => { SWAC_debug.addErrorMessage('hookmanagement.js',
err + " thrown when trying to delete filter"); })
}
```

Speichern eines existierenden Filters, übergeben werden die ID der übergeordneten Subscription und die ID des Filters. Die geänderten Daten des Filters werden aus den Eingabeelementen ausgelesen und per PUT-Anfrage an das Backend gesendet. Ist dies erfolgreich, wird der User über den Erfolg benachrichtigt und die aktualisierten Daten werden dargestellt. Gibt es Fehler bei der Anfrage, wird das Speichern abgebrochen und der User über den Fehler benachrichtigt.

```
function saveFilter(subId, id) {
        let filterSwitcherRole =
document.getElementById("filter_switcher_role_" + id);
        if (filterSwitcherRole.className === "uk-active") {
                key = userRole;
                let filterRole = document.getElementById("filter_role_" +
id);
                value = filterRole.value;
        }
        requestJSON.description = description;
        requestJSON.key = key;
        requestJSON.value = value;
```

```
        fetch(filterUpdateURL.replace("{subId}", subId) + '/' + id, {
method: "PUT", headers: {"Content-Type": "application/json", "Access-
Control-Origin": "*"}, body:  JSON.stringify(requestJSON)
        }).then(function(response){
                sendMessageToUser("Successfully updated filter " +
description);
                refreshFilters(subId);
        }).catch(err => { SWAC_debug.addErrorMessage('hookmanagement.js',
err + " thrown when trying to create a filter for: " + subId); });
}
```

Anlegen eines neuen Filters, mit der ID der übergeordneten Subscription identifiziert, wird per POST-Anfrage an das Backend gesendet. Wurde der Filter angelegt, wird der Nutzer über den Erfolg benachrichtigt, kommt es bei der Anfrage zu Fehlern, wird der Nutzer darüber benachrichtigt.

```
function createFilter(subId) {
        let filterNameNew = document.getElementById("filter_name_new");
        let description = filterNameNew.value;
        let filterSwitcherRoleNew =
document.getElementById("filter_switcher_role_new");
        if (filterSwitcherRoleNew.className === "uk-active") {
                key = userRole;
                let filterRole = document.getElementById("filter_role_new");
                value = filterRole.value; }
        requestJSON.description = description;
        requestJSON.key = key;
        requestJSON.value = value;
        fetch(filterCreateURL.replace("{subId}", subId), { method: "POST",
headers: {"Content-Type": "application/json", "Access-Control-Origin":
"*"}, body:  JSON.stringify(requestJSON)
        }).then(function(response){
                sendMessageToUser("Successfully created filter " +
description);
                refreshFilters(subId);
        }).catch(err => { SWAC_debug.addErrorMessage('hookmanagement.js',
err + " thrown when trying to create a filter for: " + subId); });
}
```

Backend

HookManager.triggerHook()

```
 public void triggerHook(String event, LinkedHashMap<String, Object>
parameters) {
```

Die „triggerHook()“ Methode des HookManagers ist der Eintrittspunkt beim Auslösen von Subscriptions. Zunächst werden die Subscriptions des zugehörigen Events aus der Datenbank geladen. Für jede der geladenen Subscriptions werden alle konfigurierten SubscriptionFilter überprüft.

```
        for (Subscription subscription : subscriptions) {
            boolean execute = subscription.isActive();
            for (SubscriptionFilter f : subscription.getFilter()) {
```

Zum einen kann nach Nutzer-Rollen gefiltert werden. Hierzu wird der „authtoken“ aus den Cookies geladen. Diese müssen sich als „cookies“ in den Parametern befinden. Dieser Token wird durch die UserManager Library überprüft und das User Object zurückgegeben. Stimmt die ID des Users oder die des Parent-Users mit der des Filters überein kann die Hook ausgeführt werden.

```
String authtoken = cookies.get("authtoken").getValue();
um = uconf.getUsermanager();
User user = um.getUser(authtoken);
if (!Objects.equals(user.getId().toString(), f.getValue()) &&
                                    !(user.getParent() != null &&
Objects.equals(user.getParent().getId(), f.getValue()))) {
                                execute = false;
                                break;
                            }
```

Der zweite mögliche Filter-Typ kann jeden beliebigen Parameter überprüfen indem er ihn mit einem festgelegten Wert vergleicht.

```
Object value = ObjectPropertyUtils.getPropertyFromObjects(parameters,
f.getKey());
if (!f.getValue().equals(value)) {
        execute = false;
}
```

Trifft einer diese Filter nicht zu, wird die Subscription nicht ausgeführt.

```
if (execute) {
                    this.executeHook(subscription, parameters);
                }
```

ObjectPropertyUtils.getPropertyFromObject()

```
public static Object getPropertyFromObject(Object obj, List<String>
keyParts) {
```

Dies ist wichtigste Methode zum Zugriff auf Propertries eines beliebigen Objektes. Als „keyParts“ werden hier die Namen der Propertries als String übergeben. Durch jedes weitere Element in dieser Liste wird „tiefer“ in das Objekt zugegriffen.

```
for (int pos = 0; pos < keyParts.size(); pos++) {
```

Wenn ein Element aus einem Array-Zugriff im Format „[0]“ besteht, wird überprüft ob es sich dabei um eine Klasse handelt, die das Iterable Interface implementiert. Dieses wird anschließend bis zu der gewünschten Stelle durchlaufen.

```
int index = Integer.parseInt(nextAttribute);
                    if (currentObj instanceof Iterable) {
                        Iterable iterable = (Iterable)currentObj;
                        var iterator = iterable.iterator();
                        for (int i = 0; i < index; i++) {
                            iterator.next();
                        }
                        currentObj = iterator.next();
                    }
```

Der eigentliche Zugriff auf die Properties des Objektes erfolgt über die Library „org.apache.commons.beanutils.PropertyUtils“.

```
currentObj = PropertyUtils.getProperty(currentObj, nextAttribute);
```

HookFeature.configure()

Diese Methode wird beim Start des Backend im Payara Webserver für jede registrierte REST-Methode aufgerufen.

```
public void configure(ResourceInfo ri, FeatureContext ctx) {
        Method method = ri.getResourceMethod();
        if (method.getAnnotation(HookEvent.class) == null) {
            // Method does not have the HookEvent annotation
            return;
        }

        HookFilter filter = new HookFilter();
        filter.setHookManager(hookmanager);
        filter.setMethod(method);
        ctx.register(filter);
    }
```

Wenn die jeweilige Methode mit der „@HookEvent“ Annotation versehen ist, wird der Methode die Middleware „HookFilter“ hinzugefügt. Diese wird anschließend vor jedem Zugriff auf die Resource ausgeführt. Darauf folgt die Ausführung der „triggerHook()“ Methode des HookManagers.

```
public void filter(ContainerRequestContext crc) throws IOException {
    String methodName = method.getDeclaringClass().getCanonicalName() + "."
+ method.getName();

    var parameter = new LinkedHashMap<String, Object>();
    parameter.put("crc", crc);
    parameter.put("cookies", crc.getCookies());
    hookmanager.triggerHook(methodName, parameter);
}
```

SubscriptionResource.list()

Durch diese REST Schnittstelle werden alle aktuell Konfigurierten Subscriptions durch einen GET Request auf der „/subscriptions“ Route zurückgegeben. Zunächst werden mit Hilfe des EntityManagers die Subscriptions aus der Datenbank gelesen. Anschließend werden diese in ein JSON-Objekt konvertiert und dem HTTP-Response angehangen.

```
@GET
@Path("")
@Produces(MediaType.APPLICATION_JSON)
public Response list() throws NotSupportedException, SystemException,
RollbackException, HeuristicMixedException, HeuristicRollbackException {

    List<Subscription> subscriptionList = this.em.createQuery("SELECT c
FROM Subscription c").getResultList();

    JsonArrayBuilder builder = Json.createArrayBuilder();

    for (Subscription subscription : subscriptionList) {
        JsonObject subscriptionObj = buildSubscriptionJson(subscription);
        builder.add(subscriptionObj);
    }

    var rb = Response.status(Response.Status.OK);
    rb.entity(builder.build());
    return rb.build();
}
```

TemplateUtils

```
public static String render(String input, HashMap<String, Object>
parameters) {
    input = replaceFor(input, parameters);
    input = replaceVariables(input, parameters);

    return input;
}
```

Zum rendern der HTTP Body Templates wird die „render()“ Funktion in der TemplateUtils Klasse verwendet. Diese unterstützt die Ersetzung von Parametern „###parameter###“ und Schleifen ###for list###. Die Schleifen werden hier zuerst ersetzt. Da diese über einen Regex gesucht werden, sind aktuell keine geschachtelten Schleifen möglich.

```
private static String replaceFor(String input, HashMap<String, Object>
parameters) {
   Pattern pattern = Pattern.compile("###for (.*?)###(.*)###end###",
Pattern.DOTALL);
    Matcher matcher = pattern.matcher(input);
    while(matcher.find()) {
        final MatchResult matchResult = matcher.toMatchResult();
        final String replacement = replaceForMatch(matchResult,
parameters);
        input = input.substring(0, matchResult.start()) +
                  replacement + input.substring(matchResult.end());
        matcher.reset(input);
    }
    return input;
}
```

Gefundene Ausschnitte werden von „replaceForMatch()“ ersetzt. Diese ersetzt ebenfalls die inneren Platzhalter durch neue Platzhalter mit Indices.

```
for (String match : matches) {
    String replacement = "###" + path + "[" + index + "]." + match + "###";
    innerString = innerString.replace("###" + match + "###", replacement);
}
```

Dadurch wird

```
###for list###
###name###
###end###
```

zu

```
###list[0].name###
###list[1].name###
```

Die Funktion „replaceVariables()“ übernimmt nun die Ersetzung der restlichen Platzhalter. Dazu verwendet diese für jeden gefundenen Platzhalter die „getPropertyFromObject()“ Methode der Klasse „ObjectPropertyUtils“.

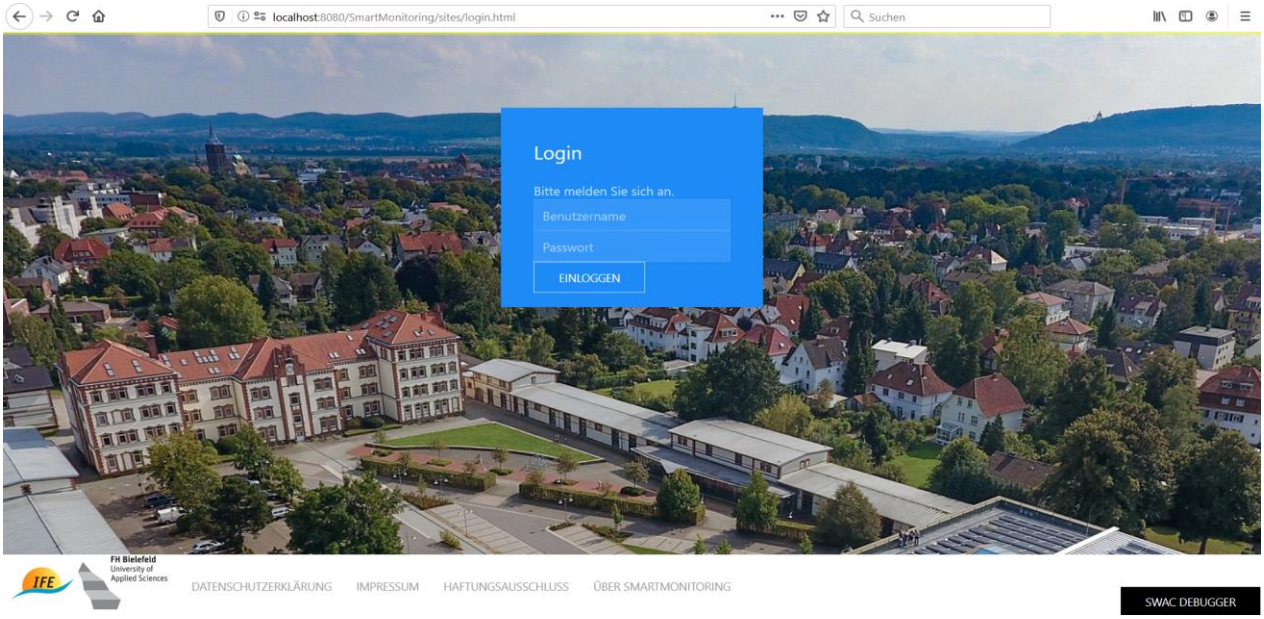
```
private static String replaceVariables(String input, HashMap<String,
Object> parameters) {
     Pattern pattern = Pattern.compile("###(.*?)###");
     Matcher matcher = pattern.matcher(input);
     while(matcher.find()) {
         final MatchResult matchResult = matcher.toMatchResult();
         final String replacement = replaceVariableMatch(matchResult,
parameters);
         input = input.substring(0, matchResult.start()) +
                 replacement + input.substring(matchResult.end());
         matcher.reset(input);
    }
    return input;
}

private static String replaceVariableMatch(MatchResult matchResult,
HashMap<String, Object> parameters) {
    String path = matchResult.group(1);
    return ObjectPropertyUtils.getPropertyFromObjects(parameters,
path).toString();
}
```

Benutzerhandbuch

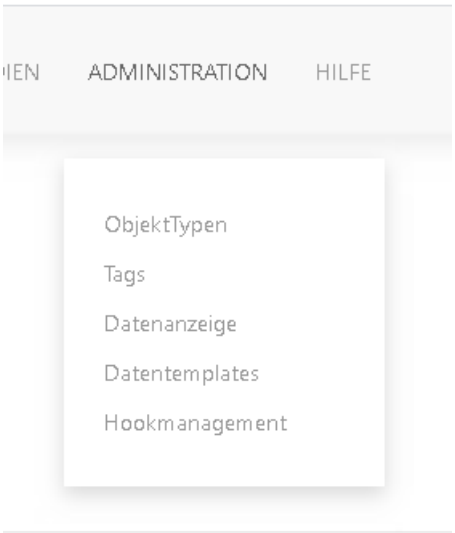
Bedienung

Login im SmartMonitoring

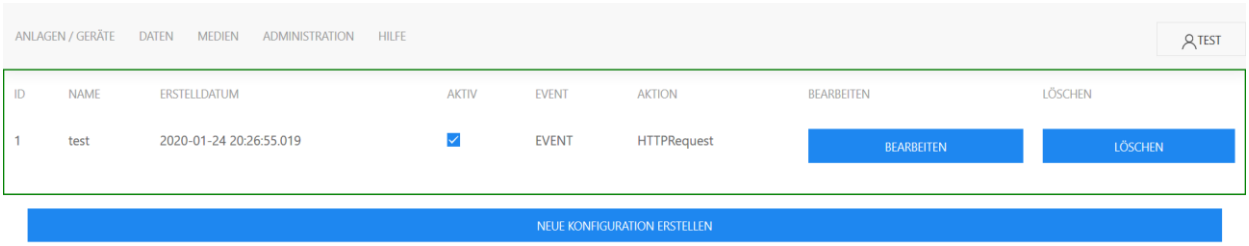


Um die Hookmanagement Seite zu erreichen muss sich der Benutzer zunächst bei der SmartMonitoring Homepage anmelden und die Seite Hookmanagement öffnen.

Nachdem sich der Benutzer angemeldet hat, kann die Hookmanagement Seite über die Navigationsleiste unter "Administration" aufgerufen werden.

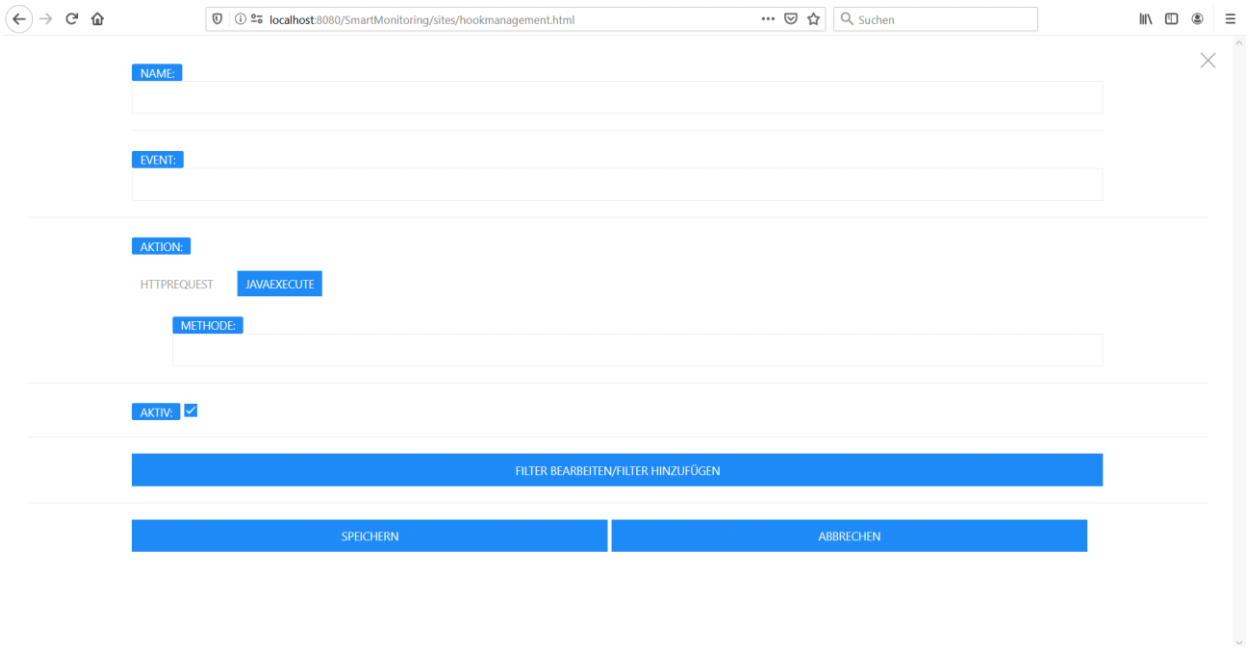


Auf der Hookmanagement Seite findet man eine Liste der bereits angelegten Hooks, sowie die Möglichkeit neue Hooks anzulegen. Hooks können nach dem Erstellen bearbeitet und gelöscht werden.



Anlegen neuer Hooks

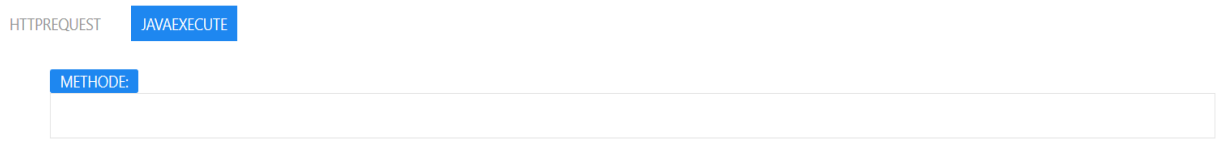
Um eine neue Hook zu registrieren, drückt der Benutzer zunächst auf den Button 'Neue Konfiguration Erstellen'.



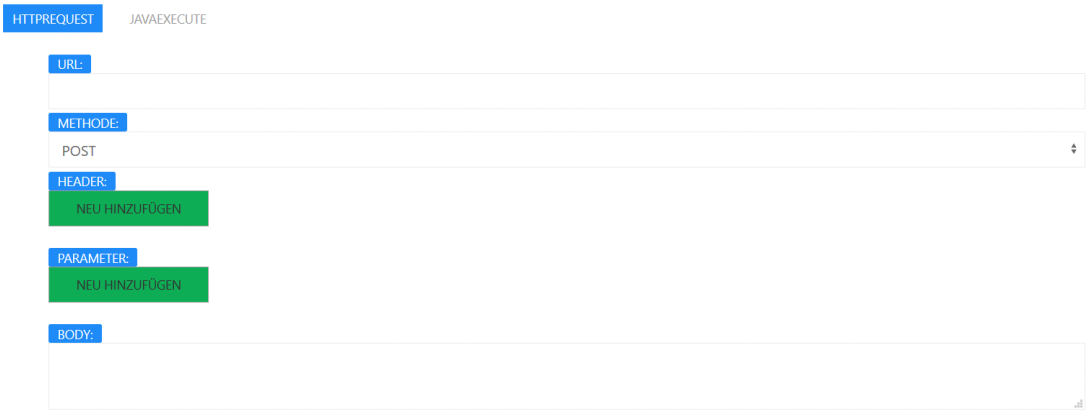
Als nächstes füllt der Benutzer die Eingabefelder mit den gewünschten Daten aus.

Es muss zwischen einer HTTP Request Konfiguration und Java Execute Konfiguration entschieden werden. Die Option "Aktiv" schaltet den Hook ein bzw. aus.

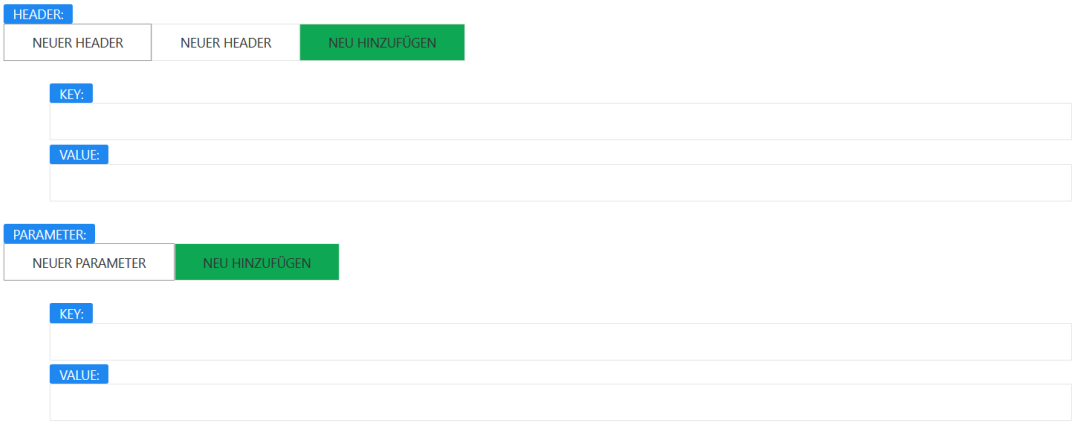
Eine JavaExecute Konfiguration kann in der folgenden Maske erstellt werden:



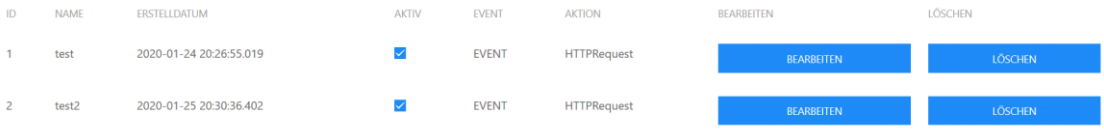
Eine HTTP-Request Konfiguration kann in der folgenden Maske erstellt werden:



Parameter und Header der HTTP Request können mit dem Button 'Neu Hinzufügen' hinzugefügt werden.



Angelegte Hooks bearbeiten

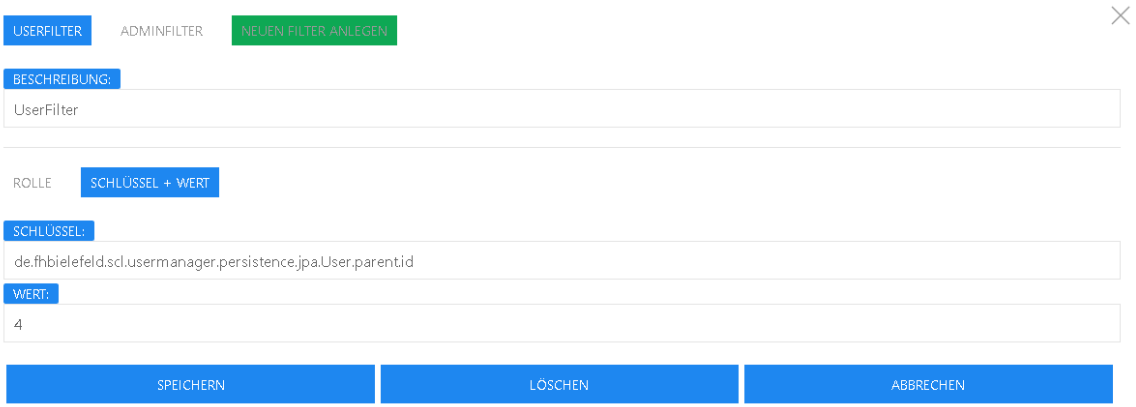


Existierende Hooks können einfach durch die Schaltfläche "Bearbeiten" geändert werden, welche in der Liste neben der jeweiligen Hook zu finden ist. Nachdem der Button gedrückt wurde, erscheint das Interface zum Bearbeiten von Hooks, welches dem Interface zum Registrieren von Hooks gleicht. Hier kann der Benutzer die gewünschten Attribute ändern und mit dem Button "Speichern" speichern.

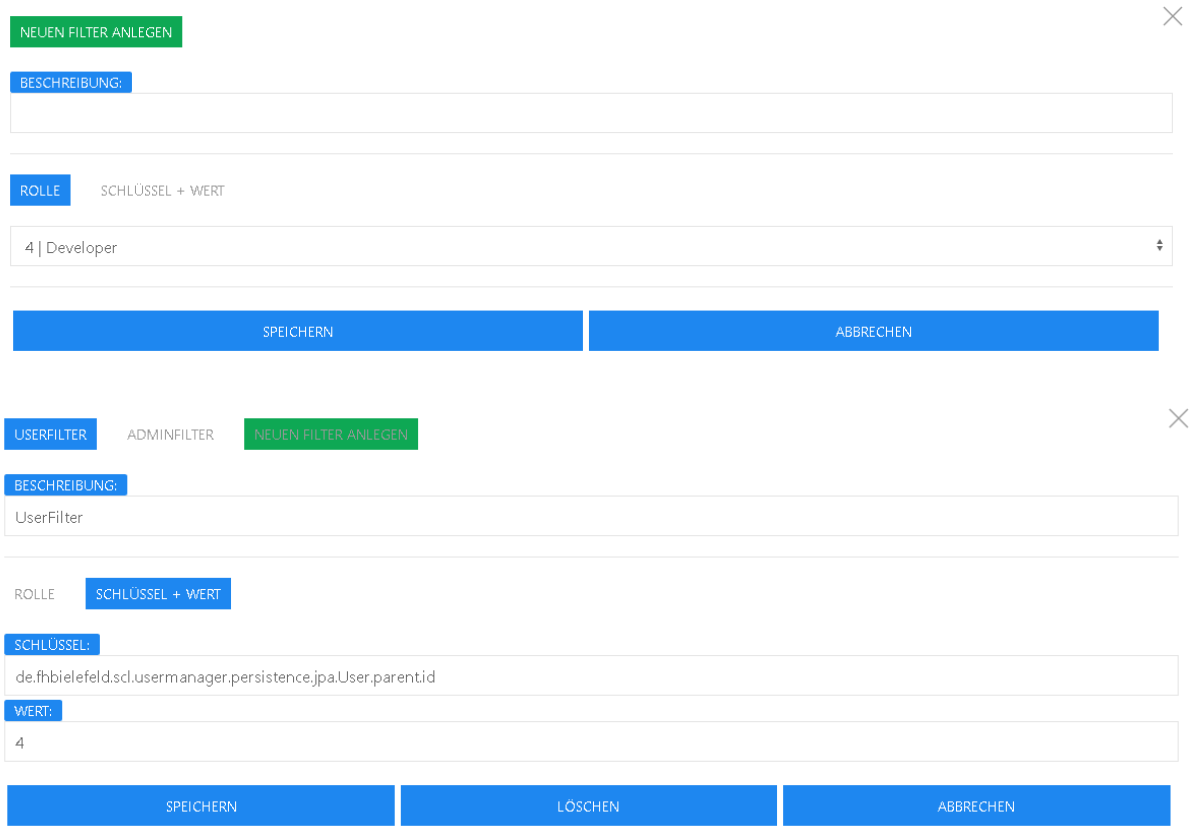
Angelegte Hooks löschen

Existierende Hooks können einfach durch die Schaltfläche "Löschen" gelöscht werden, welche in der Liste neben der jeweiligen Hook zu finden ist. Der Benutzer wird danach nach einer Bestätigung gefragt. Nach der Bestätigung wird die Angelegte Hook endgültig aus der Datenbank entfernt.

Filter bearbeiten



Filter können nach dem Anlegen einer Subscription in dem Modal zum Bearbeiten von Subscription erstellt werden. Die Schaltfläche „Neuen Filter anlegen“ öffnet die Maske zum Bearbeiten von Filtern.



Für jeden Filter muss eine Rolle angegeben werden. Diese kann mit einem Drop-Down Menü ausgewählt werden. Dazu muss für jeden Filter ein Schlüssel- und Wertpaar erstellt werden. Jeder Filter hat genau einen Schlüssel+Wert.

Ausblick

Offene Aufgaben

Die vom Auftraggeber gestellten Aufgaben wurden umgesetzt. Einzig die Auswertung der Übertragungsergebnisse konnte aus Zeitgründen nicht mehr umgesetzt werden, jedoch werden diese bereits in Form von Logging gespeichert. Die Vorarbeit für die Auswertung ist damit bereits erfolgt.

Mögliche Optimierungsmaßnahmen

Im Rahmen von Optimierungsmaßnahmen könnte beispielsweise die TemaplateEngine um komplexere Programmstrukturen erweitert werden. Beispielsweise wäre es vorstellbar Abfragestrukturen zu integrieren oder eigene Variablendeklaration einzubinden. Außerdem könnte man die Usability der TemplateEngine verbessern.

Eine Optimierung wäre auch im Bereich Java-Execute möglich. Hier muss aktuell, bei der Erstellung der Subscriptions, der fully-qualified-name (FQN) der Funktion angegeben werden. Da auszuführende Funktionen laut Anforderungen nur im lokalen Backend liegen können, wäre das automatische einsetzen des FQN möglich. Im Idealfall gäbe es eine Dropdown-Box im Frontend, in welcher alle Funktionen, gruppiert nach Klasse, selektierbar wären. Diese Erweiterung verursacht jedoch, aktuell noch, eine zu hohe Serverlast beim Laden der Dropdown-Box.

Im Frontend kann die Benutzerfreundlichkeit der „bearbeiten“ Modale verbessert werden. Hier mangelt es aktuell noch an Übersichtlichkeit. Vorstellbar wäre hier beispielsweise die Aufteilung der Bereiche Header-Konfiguration und Parameter-Konfiguration in eigene Tabs.

Für zukünftige Entwicklungen wäre die Integration von Unit Tests für das entwickelte Modul von Vorteil, da damit die problemfreie Funktion auch nach Änderungen gewährleistet werden kann.

Fazit

Das Projekt war für alle Mitglieder sehr lehrreich. Neben den fachlichen Herausforderungen mit denen wir konfrontiert wurden, mussten wir auch unsere Außerfachlichen Kompetenzen unter Beweis stellen. Die gesammelte Erfahrung wird uns bei zukünftigen Projekten mit großer Sicherheit helfen.

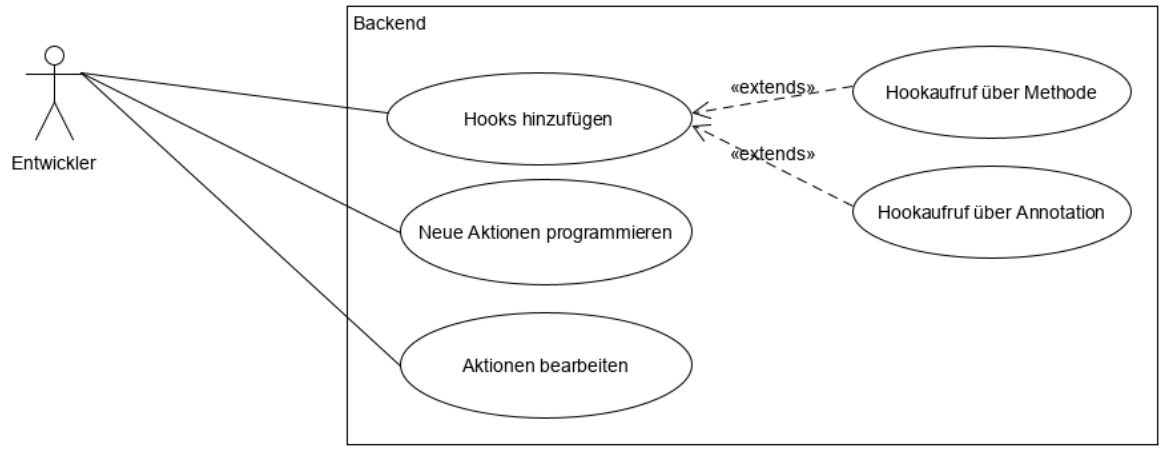
Besonders gut am Projektverlauf haben uns die variablen Projektthemen und der enge Austausch mit dem Auftraggeber gefallen.

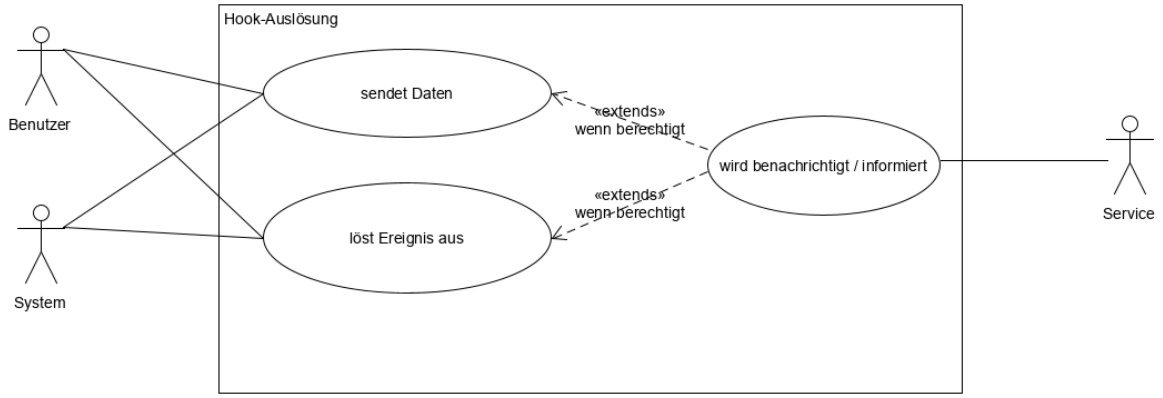
Anlagen

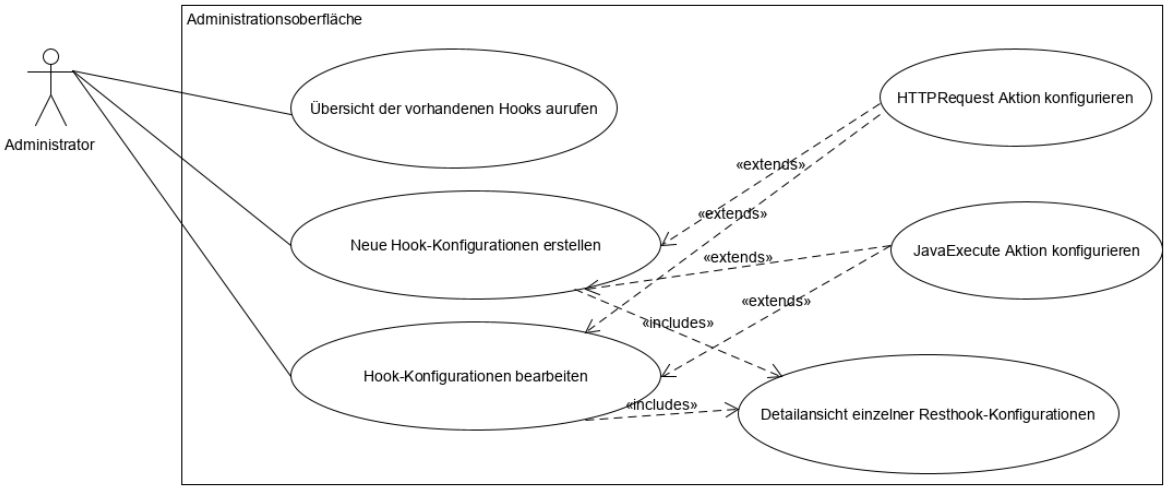
Arbeitsplan

Arbeitsplan	Benedikt Wiest	Steffen Dorsch	Luca Berneking	Julien Riechmann
KW 43	Vorbereitung Feinkonzept, Erstellung Sequenzdiagramme, Erstellung Klassendiagramm, Organisation	Definition der Use-Cases, Erstellung Use-Case-Diagramme, Erstellung UI-Mockups	Planung Architektur, Erstellung Klassendiagram, Erstellung Architekturübersicht	Planung Datenmodell, Erstellung ER-Diagramm
KW 44 (Feinkonzept)	Unterstützung bei Implementierung UI und Implementierung Subscriptions	Beginn Implementierung UI	Beginn Implenetierung Subscriptions	Beginn Implementierung Datenbank
KW 45	Unterstützung bei Implementierung UI und Beginn Implemtierung HookManager	Implementierung UI	Fertigstellung Subscriptions, Dokumentation und Tests	Fertigstellung Datenbank, Dokumentation
KW 46 (MS1)	Organisation, Implementierung HookManager und HookAction	Anbindung Frontend an bestehende Backend-Funktionalität	Unterstützung bei Implementierung HookManager und HookAction	Implementierung HookSubscriptionResource
KW 47	Organisation, Implementierung HookManager und HookAction	Fertigstellung UI	Implementierung Hook-Manager und HookAction	Implementierung HookSubscriptionResource
KW 48 (MS2)	Organisation, Unterstützung bei Implementierung ActionConfig	Implementierung Frontend Tests	Beginn Implenetierung Annotation	Beginn Implementierung ActionConfig
KW 49	Organisation, Unterstützung Implentierung SubscriptionFilter	Beginn Implementierung SubscriptionFilter	Implenetierung Annotation	Implementierung ActionConfig
KW 50 (MS3)	Organisation, Unterstützung Implentierung SubscriptionFilter	Fertigstellung SubscriptionFilter, Tests und Dokumenation	Fertigstellung Annotation Implementierung, Dokumentation und Tests	Fertigstellung ActionConfig
KW 51	Organisation, Refactoring, Dokumentationsarbeiten	Erstellung Benutzerhandbuch	Refactoring, Dokumentationsarbeiten	Refactoring, Dokumentationsarbeiten
KW 52	Urlaub	Urlaub	Urlaub	Urlaub
KW 1	Urlaub	Urlaub	Urlaub	Urlaub
KW 2	Organisation, Bugfixing	Bugfixing	Bugfixing, Testweise Integration	Bugfixing
KW 3	Organisation, Vorbereitung Übergabe, Beginn Erstellung Projektbericht	Vorbereitung Übergabe, Hotfixes, verbleibende Dokumentationsarbeiten	Vorbereitung Übergabe, Hotfixes, Kontrolle Git, Integration	Vorbereitung Übergabe, Hotfixes, verbleibendeDokumentationsarbeiten, Integration
KW 4 (Übergabe)	Erstellung Projektbericht, zusammenstellen der Dokumentationen	Unterstützung bei Erstellung Projektbericht	Unterstützung bei Erstellung Projektbericht	Unterstützung bei Erstellung Projektbericht
KW 5 (Projektbericht)				

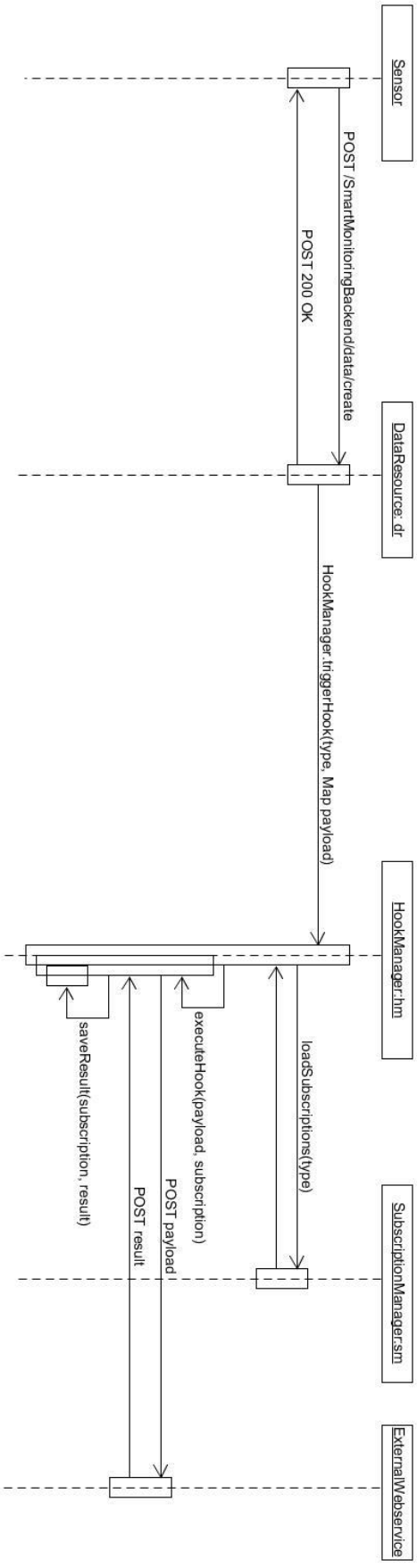
Use-Case-Diagramme

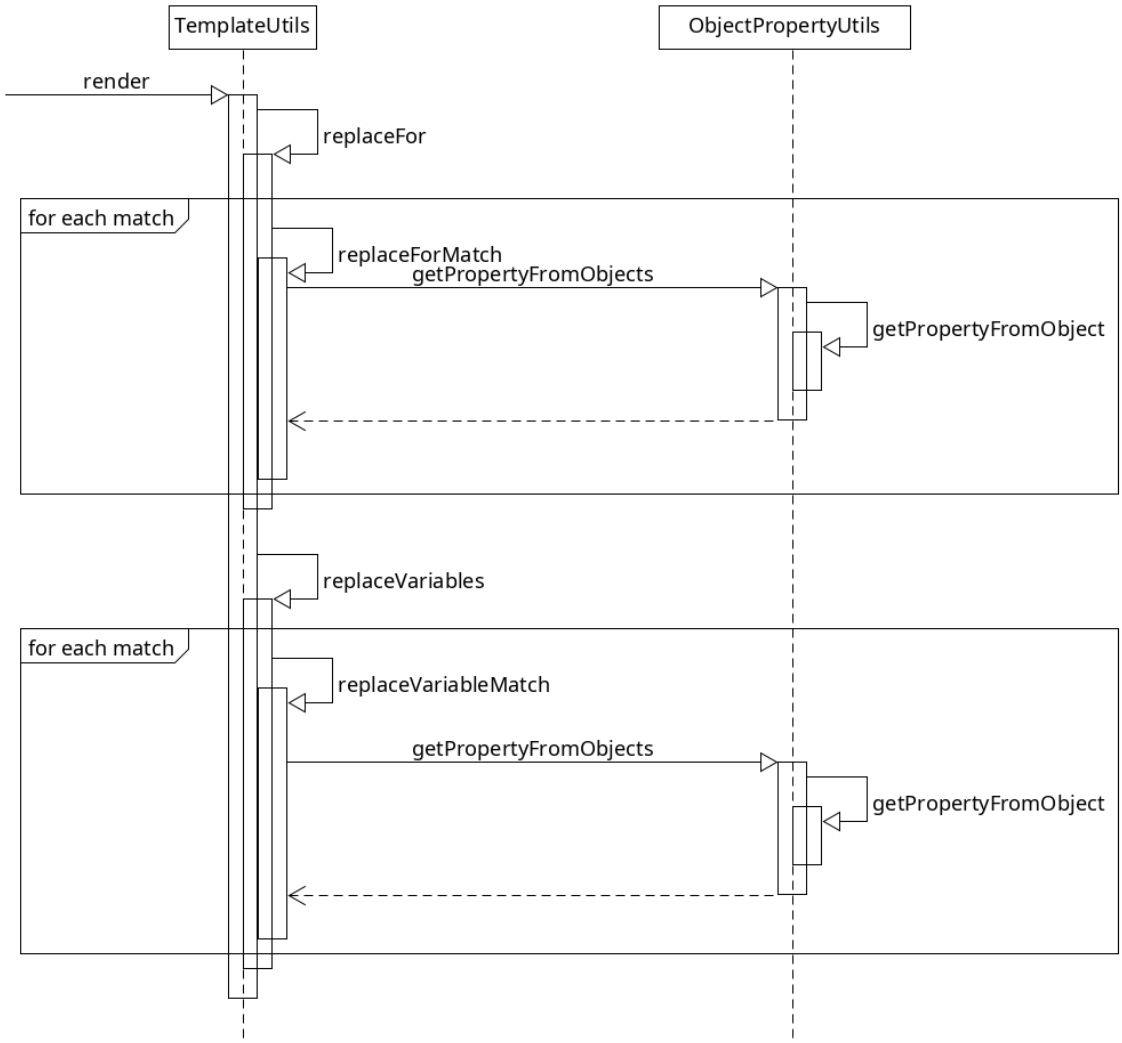






Sequenzdiagramme




TemplateUtils
ObjectPropertyUtils
render
replaceFor
for each match
replaceForMatch
getPropertyFromObjects
getPropertyFromObject
replaceVariables
for each match
replaceVariableMatch
getPropertyFromObjects
getPropertyFromObject

Klassendiagramm