|  |
| --- |
|  |

|  |
| --- |
|  |

IPA von Remo Kessler

Zentralisierte Parameterverwaltung für eine Mikroservices-Architektur

Über dieses Dokument

Ablage

|  |  |
| --- | --- |
| Git Repository | https://github.com/kre-cmi/IPA-KRE |

Versionierung

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Datum | Autor | Status | Bemerkung |
| 0.1 | 22.02.2018 | Remo Kessler | In Arbeit | Erstellung des Dokuments inkl. Formatierungen und Ablage im GIt. |
|  |  |  |  |  |

Referenzierte Dokumente

|  |  |  |  |
| --- | --- | --- | --- |
| Dokumentenname | Version | Autor | Datum |
|  |  |  |  |
|  |  |  |  |

Verteiler

|  |  |  |  |
| --- | --- | --- | --- |
| Name / Vorname | Kurzzeichen | Organisation | Rolle |
| Remo Kessler | KRE | CM Informatik AG | Lehrling Informatik |

Konventionen

In diesem Dokument wurden die folgenden Konventionen verwendet:

|  |  |
| --- | --- |
| Was | Beschrieb |
| Gleichstellung | Im Zuge sprachlicher Vereinfachung wird innerhalb des vorliegenden Dokuments jeweils nur eine Form von Personenbezeichnungen (z.B. Projektleiterin, Mitarbeiter etc.) verwendet. Es ist stets auch die andere Form der entsprechenden Personenbezeichnung gemeint und miteingeschlossen. |
| Grün & ??? | Grün hinterlegter Text betrifft Punkte, welche noch besprochen werden müssen |
| Gelb & !!! | Gelb hinterlegter Text betrifft Punkte, welcher bei der Umsetzung speziell beachtet werden müssen. |

Inhaltsverzeichnis

[1.1 Thema und Zielsetzung 4](#_Toc509235286)

[1.2 Allgemeines zur Umgebung 4](#_Toc509235287)

[1.3 Endprodukt 4](#_Toc509235288)

[2.1 Titel der IPA 5](#_Toc509235289)

[2.2 Auslöser der Aufgabenstellung 5](#_Toc509235290)

[2.3 Grobanforderungen 5](#_Toc509235291)

[2.3.1 Parameterverwaltung 5](#_Toc509235292)

[2.3.1.1 Funktionale Anforderungen 5](#_Toc509235293)

[2.3.1.2 Nicht Funktionale Anforderungen 5](#_Toc509235294)

[2.3.2 «Getting Started» Dokumentation 6](#_Toc509235295)

[2.4 Mittel & Methoden 6](#_Toc509235296)

[2.4.1 Eingesetzte Mittel 6](#_Toc509235297)

[2.4.2 Git Workflow 6](#_Toc509235298)

[2.4.3 Kontroll-Tasks 6](#_Toc509235299)

[2.5 Vorkenntnisse 6](#_Toc509235300)

[2.6 Neue Lerninhalte 6](#_Toc509235301)

[3.1 Konventionen 7](#_Toc509235302)

[3.1.1 Dokumentation 7](#_Toc509235303)

[3.1.2 Code-Konventionen C# 7](#_Toc509235304)

[3.1.3 Code-Konventionen TypeScript 7](#_Toc509235305)

[3.1.4 Layout-Konventionen 8](#_Toc509235306)

[3.2 Vorgehensmodell 8](#_Toc509235307)

[3.2.1 IPERKA 8](#_Toc509235308)

[4.1 Umsysteme 9](#_Toc509235309)

[4.2 Abhängigkeiten 9](#_Toc509235310)

[4.3 Systemgrenzen 10](#_Toc509235311)

[4.4 Schnittstellen 11](#_Toc509235312)

[5.1 Einzeltaskerstellung für die Realisierung 12](#_Toc509235313)

[5.1.1 Speichern der Parameter 12](#_Toc509235314)

[5.1.2 Implementation des Validierungsmechanismus 12](#_Toc509235315)

[5.1.3 Zentralisierte Parameterverwaltung Kontrolltask 13](#_Toc509235316)

[5.1.4 Suchen eines Parameters 13](#_Toc509235317)

[5.1.5 GUI Tests 13](#_Toc509235318)

# Zweck des Dokuments

In diesem Dokument wird der Entwicklungsprozess der IPA «Zentralisierte Parameterverwaltung für eine Microservice-Architektur» beschrieben. Des Weiteren dient diese Dokumentation als Abschlussarbeit von Remo Kessler.

## Thema und Zielsetzung

Diese Projektarbeit befasst sich mit dem Thema von Microservices und wie man diese zentral verwalten kann. Dies unter Berücksichtigung der Microservice-Architektur.

Dies ist nun das Ausgangsproblem für diese Arbeit. Als Endprodukt soll eine funktionierende, zentrale Parameterverwaltung für das oben genannte Problem entstehen. Dies natürlich unter der Berücksichtigung von der Microservice-Architektur.

## Allgemeines zur Umgebung

Das Projekt wird in einem eigenen Repository unabhängig vom restlichen Projekt umgesetzt. Um das ganze möglichst zu vereinfachen sind sämtliche Funktionen, die nichts mit der IPA zu tun haben, nicht in diesem Repository enthalten. Im Anschluss des Projekts wird die Parameterverwaltung in die Entwicklungsumgebung und anschliessend in die Live-Umgebung eingebaut.

## Endprodukt

Nach Abschluss der Projektarbeit liegen folgende Produkte vor:

* Vollständige Dokumentation des Lösungsweges
* Arbeitsjournale
* Programm als Visaul Studio respektive Webstorms Projekte

# Rahmenbedingungen

## Titel der IPA

Zentralisierte Parameterverwaltung für eine Mikroservices-Architektur

## Auslöser der Aufgabenstellung

Im Projekt Viaduc, welches von der CM Informatik AG derzeit umgesetzt wird, geht es derzeit um die Erstellung einer Software, welche zur Aufgabe hat einen klassischen Lesesaal online verfügbar zu machen. Durch das die hohen und komplexen Anforderungen an den Backendbereich, entschloss man sich dieses mit einer Microservicearchitektur um zu setzen. Dies hat zur Folge, dass die Parametrierung dieser Services nicht zentral gespeichert werden kann, da sonst die Eigenständigkeit der Services nicht mehr gewährleistet wird und die Microservice-Architektur dann keine richtige Microservice-Architektur mehr wäre.

## Grobanforderungen

### Parameterverwaltung

Dies sind die Kriterien aus der Aufgabenstellung für die Parameterverwaltung.

#### Funktionale Anforderungen

* Der Administrator kann alle Parameter aller Systemdienste an einem Ort pflegen
* Das Auffinden des gewünschten Parameters wird vom System unterstützt
* Default-Werte sind optisch als solche identifizierbar
* Validierungsfehler werden optisch hervorgehoben
* Ein Hilfetext kann für jeden Parameter hinterlegt werden
* Die Validierung erlaubt die Kontrolle bei der Erfassung
* Die Validierung kann auch separat aufgerufen werden
* Der Validierungsmechanismus soll versch. Formate unterstützen
* Die PV erkennt selbstständig die parametrierbaren Dienste und ihre Werte

#### Nicht Funktionale Anforderungen

* Die Parameter werden im Service gespeichert, nicht zentral(!)
* Die Services funktionieren auch, wenn die Parameterverwaltung (PV) nicht läuft
* Die PV funktioniert rechnerübergreifend, ohne File-Sharing
* Es werden sprechende Namen für Klassen / Methoden verwendets
* Eine Beispielimplementierung im Projekt Viaduc ist vorhanden und einsehbar
* Der Programmierer braucht sich nicht um GUI Aspekte zu kümmern
* Ist es nachvollziehbar, warum gerade diese Lösung gewählt wurde? Was waren die Kriterien?
* Werden mögliche Fehler mit den entsprechenden Mitteln erkannt und behandelt? Mögliche Fehler: fehlerhafte Parameter, fehlende Parameter

### «Getting Started» Dokumentation

Dies sind die Kriterien aus der Aufgabenstellung an die «Getting Started» Dokumentation. Da dies eine Reine Dokumentation ist, macht es keinen Sinn in die Unterteilung Funktional und nicht Funktionale Anforderungen.

* Bietet einen konzeptionellen Überblick
* Schritt für Schritt Anleitung für Einbindung
* Mind. 2 Beispiele für die Validierung
* Mind. 2 Screenshots der resultierenden Darstellung im GUI

## Mittel & Methoden

### Eingesetzte Mittel

* Visual Studio für die Backendprogrammierung in C#
* Webstorms für die Frontendprogrammierung in TypeScript
* MS Office für die Erstellung der Dokumentation
* Office at Work Vorlagen, um das CI/CD der Dokumentation ein zu halten
* Konventionen der Programmierung im Backend und Frontend wie bei der CM Informatik AG üblich

### Git Workflow

Das gesamte Projekt wird mit Git respektive Github versioniert und abgelegt. Da es im Projekt nur einen «Contributer» gibt, macht es wenig Sinn immer einen eigenen Branch pro Task zu erstellen, da man keinen Nutzen daraus gewinnt und Zeit verliert durchs Mergen etc. Wegen der Sicherheit der Daten wird dennoch jeder Commit auch gleich gepusht. Zudem wird es jeden Tag eine Version geben.

### Kontroll-Tasks

* Nach den IPERKA Phasen Planen, Entscheiden und Realisieren wird das Dokument zum gegenlesen gegeben. Am anschliessenden Tag, werden jeweils 45 Minuten eingerechnet, um zu besprechen und allfällige Fehler zu beheben.
* Jeweils am Ende einer Woche, wird die Anforderungsliste mit dem Ist Stand verglichen.

## Vorkenntnisse

* C# Programmierung während der Lehre bei der Greenshare AG / CM Informatik
* TypeScript Programmierung / LESS / Angular seit einem halben Jahr bei der CM Informatik AG im Projekt Viaduc
* HTML und CSS Kenntnisse aus der Berufsschule und dem Viaduc Projekt

## Neue Lerninhalte

* Microservices-Architektur
* RabbitMQ / MassTransit

# Allgemeines

## Konventionen

### Dokumentation

|  |  |
| --- | --- |
| Bereich | Vorgabe |
| Seitenumbruch | Alle Überschriften erster Stufe beginnen auf einer neuen Seite. Keine Überschrift steht ganz unten auf einer Seite. |
| CI/CD | Von OAW zu entnehmen |

### Code-Konventionen C#

|  |  |
| --- | --- |
| Bereich | Vorgaben |
| Namensgebung Methoden/Variablen | Methoden- und Variablennamen soll Prägnant und in Englisch sein zudem immer in Camel-Case Fachbegriffe in Deutsch |
| Private / Public | Private klein, Public gross |
| Methodenlänge | Methoden sollen nicht zu lang sein. |
| Service | Ein Service besteht aus dem Host, dem Manager und dem Contract.  Im Host befindet sich nur der Service selbst, im Manager die Logik und im Contract die gemeinsame Logik von Empfänger und Sender. |
| Projektnamensgebung | CMI.(was es ist: Host | Manager | Service | Web).ServiceName |

### Code-Konventionen TypeScript

|  |  |
| --- | --- |
| Bereich | Vorgaben |
| Namensgebung Methoden/Variablen | Methoden- und Variablennamen soll Prägnant und in Englisch sein zudem immer in Lower-Camel-Case Fachbegriffe in Deutsch |
| Private / Public | Private mit Underscore, Public nur Lower-Camel-Case |
| Methodenlänge | Methoden sollen nicht zu lang sein. |
| Component | Pro Component ist ein eigenes Verzeichnis an zu legen  KomponentenName.component.(ts | html | less)  Services sind im Services Verzeichnis ab zu legen und via Dependency Injection an zu steuern. |

### Layout-Konventionen

|  |  |
| --- | --- |
| Bereich | Vorgabe |
| CI / CD | CI / CD des Bundes ist ein zu halten. Sagt dieses nichts aus ist auf Bootstrap zurück zu greifen. Ist da ebenfalls nichts definiert, so muss ein Vorschlag gemacht werden. |
| Barrierefreiheit | Es muss alles mit der Tastatur bedienbar sein. |

## Vorgehensmodell

Das Projekt wird mit IPERKA umgesetzt. Die CM Informatik AG setzt in der Entwicklung auf Scrum. Dies ist jedoch bewusst anders gewählt, da die CM Informatik AG mit einem 1 Monatssprint arbeitet. Dies ist nicht Sinnvoll für ein Projekt von 3 Wochen Dauer. Zudem kommen die Vorteile der Agilen Entwicklung in einem in sich selbst Abgeschlossenen Projekt nicht zum Tragen. Da die IPA eine Projektarbeit unter idealen Bedingungen darstellt, muss auch nicht auf allfällige Anforderungsänderungen eingegangen werden.

### IPERKA

Die sechs Phasen von IPERKA gehen in einander über. Welcher Task zu welchem schritt gehört ist im Zeitplan ersichtlich

# Umsysteme, Abhängigkeiten, Systemgrenzen und Schnittstellen

Das ganze System ist anhand der Vereinfachten Bedingungen während der IPA aufgezeigt, da das Livesystem zu gross und komplex ist für dieses Projekt. Die Services, die im Livesystem vorhanden sind, verhalten sich gleich, wie jene, die sich im IPA-System befinden.

## Umsysteme

Für das IPA-Projekt gibt es keine relevanten Umsysteme, da man sich innerhalb des Projektes Viaduc befindet. Die Kommunikation mit den Umsystemen findet im Viaduc über einzelne Microservices stat. Da diese aber in der IPA nicht enthalten sind, kommuniziert das System mit keinem Umsystem.

## Abhängigkeiten

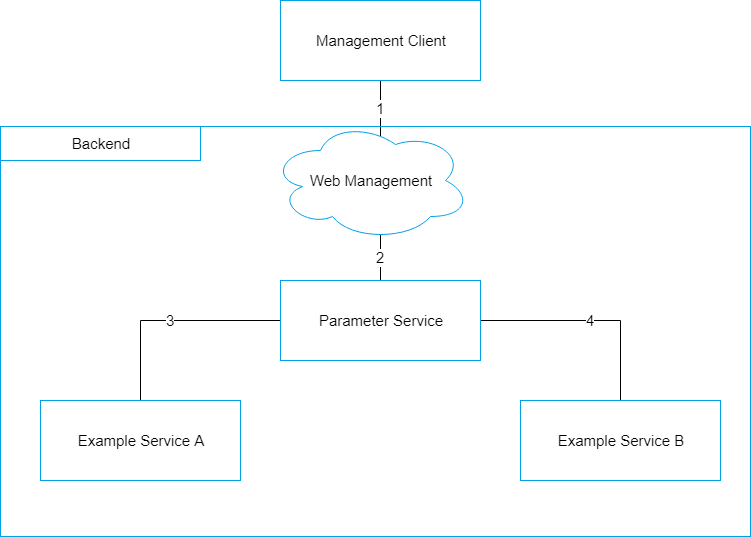
Aufgrund der Microservices-Architektur ist das ganze Programm so unabhängig wie es geht. Jeder Service ist in sich selbst gekapselt. Die einzelnen Services sind nur über die jeweilige Contract DLL miteinander verbunden.

Im Fall dieser Arbeit ist dies die CMI.Contract.Parameter, welche als einzige eine Abhängigkeit hat, eine Zentrale Rolle. In dieser DLL werden sämtliche Kommunikationsinterfaces für den Parameter Service definiert.

## Systemgrenzen

Im Projekt Viaduc haben wir verschiedenste Services. Diese sind im Rahmen dieser Projektarbeit durch die «Example Services» A und B repräsentiert. Der Web Managementteil dient nur zum Informationseingang Sämtliche Logik ist im Parameter Service zu schreiben.

|  |  |
| --- | --- |
| Bezeichnung | Was es ist |
| Management Client | Der Web Client vom Projekt |
| Web Management | Der Webendpunkt, wo sich die APIs befinden |
| Parameter Service | Der Service, welcher während der IPA |
| Example Service A & B | Diese zwei Services stellen zwei beliebige Viaduc Microservices dar |
| 1 | API Call und Response |
| 2, 3, 4 | RabbitMQ / MassTransit Kommunikation |



## Schnittstellen

Die Kommunikation entsteht durch RabbitMQ und MassTransit. Mit diesen beiden Frameworks können wir verschiedene Kommunikations-Queues erstellen. Diese werden dann nach dem First in First out Prinzip abgearbeitet.

# Detailanalyse der Anforderungen

## Einzeltasks für die Dokumentation

### Getting Started Dokumentation

|  |
| --- |
| Erfüllte Ziele |
| Bietet einen konzeptionellen Überblick |
| Schritt für Schritt Anleitung für Einbindung |
| Mind. 2 Beispiele für die Validierung |
| Mind. 2 Screenshots der resultierenden Darstellung im GUI |

|  |
| --- |
| Anforderung |
| Die Getting Started Dokumentation soll es nach der IPA erleichtern den anderen Projekt-Entwicklern selbst einen Parameter zu erstellen für die Parameterverwaltung |

**Geschätzter Aufwand**: 2h

## Einzeltasks für die Realisierung

### Speichern der Parameter

|  |
| --- |
| Erfüllte Ziele |
| Ein Hilfetext kann für jeden Parameter hinterlegt werden |
| Default-Werte sind optisch als solche identifizierbar |
| Werden mögliche Fehler mit den entsprechenden Mitteln erkannt und behandelt? Mögliche Fehler: fehlerhafte Parameter, fehlende Parameter |
| Die Parameter werden im Service gespeichert, nicht zentral |
| Die PV erkennt selbstständig die parametrierbaren Dienste und ihre Werte |
| Der Administrator kann alle Parameter aller Systemdienste an einem Ort pflegen |

|  |
| --- |
| Anforderung |
| Die Parameterverwaltung muss die Parameter Speichern können. Dies muss zentral im Management Client geschehen. Zu diesem Zweck muss die Parameterverwaltung alle Parameter Abfragen und Validieren können.  Eine funktionierende Validierung ist zur Umsetzung dieses Tasks erforderlich. |

**Geschätzter Aufwand**: 12h

### Implementation des Validierungsmechanismus

|  |
| --- |
| Erfüllte Ziele |
| Die Validierung kann auch separat aufgerufen werden |
| Die Validierung erlaubt die Kontrolle bei der Erfassung |
| Validierungsfehler werden optisch hervorgehoben |
| Der Validierungsmechanismus soll versch. Formate unterstützen |
| Werden mögliche Fehler mit den entsprechenden Mitteln erkannt und behandelt? Mögliche Fehler: fehlerhafte Parameter, fehlende Parameter |

|  |
| --- |
| Anforderung |
| Die Parameterverwaltung muss die Richtigkeit der Daten überprüfen können. Diese soll bei Bedarf auch separat aufgerufen werden, sodass z.B. beim neuerfassen eines Service gleich hervorgehoben werden kann, was noch zu korrigieren ist und wo die Default-Werte genügen. |

**Geschätzter Aufwand**: 8h

### Suchen eines Parameters

|  |
| --- |
| Erfüllte Ziele |
| Das Auffinden des gewünschten Parameters wird vom System unterstützt |

|  |
| --- |
| Anforderung |
| Das System muss um Benutzerfreundlich zu bleiben dem Benutzer dabei behilflich sein, einen gewünschten Parameter schnell zu finden, da zurzeit noch ungewiss ist, wie viele Parameter wirklich in der Parameterverwaltung vorkommen werden. |

**Geschätzter Aufwand**: 8h

### Erstellung der Unit Tests

|  |
| --- |
| Anforderung |
| Für Funktionen, welche daten Verarbeiten oder Validieren, muss ein mindestens ein Unit-Test geschrieben werden. |

**Gesamter Geschätzter Aufwand:** 4h

## Einzeltaskerstellung der Kontroll Tasks

### Zentralisierte Parameterverwaltung Kontrolltask

|  |
| --- |
| Erfüllte Ziele |
| Die Services funktionieren auch, wenn die Parameterverwaltung (PV) nicht läuft |
| Die PV funktioniert rechnerübergreifend, ohne File-Sharing |
| Der Programmierer braucht sich nicht um GUI Aspekte zu kümmern |
| Eine Beispielimplementierung im Projekt Viaduc ist vorhanden und einsehbar |
| Es werden sprechende Namen für Klassen / Methoden verwendets |
| Ist es nachvollziehbar, warum gerade diese Lösung gewählt wurde? Was waren die Kriterien? |

|  |
| --- |
| Anforderung |
| Überprüfung ob alle oben genannten Ziele erfüllt wurden. |

**Geschätzter Aufwand**: 2h

### Durchführen der Tests

|  |
| --- |
| Anforderung |
| **Frontend**  Die GUI Tests müssen von Hand ausgeführt werden. Sie beinhalten das Testen von:   * der Barrierefreiheit (alles mit der Tastatur bedienbar?) * Ist das CI/CD vom Bund, oder falls dies nichts sagt, von Bootstrap eingehalten?   **Backend**  Auf dem Server müssen die Unit Tests durchlaufen, Sobald diese geschrieben sind. |

**Geschätzter Aufwand pro Durchgang**: 1h

Dieser Task muss 1 Mal nach jedem Realisiertem Realiseriungs-Punkt Umgesetzt werden.

**Gesamter Geschätzter Aufwand:** 4h