|  |
| --- |
|  |

|  |
| --- |
|  |

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. sProjektleiterin, 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 Zweck des Dokuments 6](#_Toc509817781)

[1.1 Thema und Zielsetzung 6](#_Toc509817782)

[1.2 Allgemeines zur Umgebung 6](#_Toc509817783)

[1.3 Endprodukt 6](#_Toc509817784)

[2 Rahmenbedingungen 7](#_Toc509817785)

[2.1 Titel der IPA 7](#_Toc509817786)

[2.2 Auslöser der Aufgabenstellung 7](#_Toc509817787)

[2.3 Grobanforderungen 7](#_Toc509817788)

[2.3.1 Parameterverwaltung 7](#_Toc509817789)

[2.3.1.1 Funktionale Anforderungen 7](#_Toc509817790)

[2.3.1.2 Nicht Funktionale Anforderungen 7](#_Toc509817791)

[2.3.2 «Getting Started» Dokumentation 8](#_Toc509817792)

[2.4 Vorkenntnisse 8](#_Toc509817793)

[2.5 Neue Lerninhalte 8](#_Toc509817794)

[2.6 Mittel & Methoden 8](#_Toc509817795)

[2.6.1 Eingesetzte Mittel 8](#_Toc509817796)

[2.6.2 Git Workflow 8](#_Toc509817797)

[2.6.3 Daily Business 8](#_Toc509817798)

[2.6.4 Dokumentablage 9](#_Toc509817799)

[2.6.5 Kontroll-Tasks 9](#_Toc509817800)

[3 Allgemeines 10](#_Toc509817801)

[3.1 Konventionen 10](#_Toc509817802)

[3.1.1 Dokumentation 10](#_Toc509817803)

[3.1.2 Code-Konventionen C# 10](#_Toc509817804)

[3.1.3 Code-Konventionen TypeScript 10](#_Toc509817805)

[3.1.4 Layout-Konventionen 11](#_Toc509817806)

[3.2 Vorgehensmodell 11](#_Toc509817807)

[3.2.1 IPERKA 11](#_Toc509817808)

[4 Umsysteme, Abhängigkeiten, Systemgrenzen und Schnittstellen 12](#_Toc509817809)

[4.1 Umsysteme 12](#_Toc509817810)

[4.2 Abhängigkeiten 12](#_Toc509817811)

[4.3 Systemgrenzen 13](#_Toc509817812)

[4.4 Schnittstellen 14](#_Toc509817813)

[5 Taskerstellung nach den Anforderungen 15](#_Toc509817814)

[5.1 Einzeltasks für die Dokumentation 15](#_Toc509817815)

[5.1.1 Getting Started Dokumentation 15](#_Toc509817816)

[5.2 Einzeltasks für die Realisierung 15](#_Toc509817817)

[5.2.1 Anzeige & Speichern der Parameter 15](#_Toc509817818)

[5.2.2 Implementation des Validierungsmechanismus 16](#_Toc509817819)

[5.2.3 Suchen eines Parameters 16](#_Toc509817820)

[5.2.4 Erstellung der Unit Tests und des Testkonzepts 16](#_Toc509817821)

[5.3 Einzeltaskerstellung der Kontrolltasks 17](#_Toc509817822)

[5.3.1 Durchführen der Tests 17](#_Toc509817823)

[6 Zeitplan 18](#_Toc509817824)

[7 Detailanalyse der erstellten Tasks 19](#_Toc509817825)

[7.1 Umsetzungsreihenfolge 19](#_Toc509817826)

[7.2 Erstellung des Testkonzepts 19](#_Toc509817827)

[7.2.1 Anforderung 19](#_Toc509817828)

[7.2.2 Mögliche Lösungen 19](#_Toc509817829)

[7.2.3 Umsetzungsbeschreibung 20](#_Toc509817830)

[7.3 Anzeige & Speichern der Parameter 21](#_Toc509817831)

[7.3.1 Anforderung 21](#_Toc509817832)

[7.3.2 Mögliche Lösungen Speichern 21](#_Toc509817833)

[7.3.3 Umsetzungsbeschreibung Speichern 22](#_Toc509817834)

[7.3.4 Mögliche Lösung Anzeige 22](#_Toc509817835)

[7.3.5 Umsetzungsbeschreibung Anzeige 25](#_Toc509817836)

[7.4 Implementation des Validierungsmechanismus 26](#_Toc509817837)

[7.4.1 Anforderung 26](#_Toc509817838)

[7.4.2 Mögliche Lösungen 26](#_Toc509817839)

[7.4.3 Umsetzungsbeschreibung 26](#_Toc509817840)

[7.5 Suche eines Parameters 27](#_Toc509817841)

[7.5.1 Anforderung 27](#_Toc509817842)

[7.5.2 Mögliche Lösungen 27](#_Toc509817843)

[7.5.3 Umsetzungsbeschreibung 28](#_Toc509817844)

[7.6 Erstellung der Unittests 28](#_Toc509817845)

[7.6.1 Anforderung 28](#_Toc509817846)

[7.6.2 Umsetzungsbeschreibung 28](#_Toc509817847)

[7.7 Durchführen der Tests 28](#_Toc509817848)

[7.7.1 Anforderung 28](#_Toc509817849)

[7.7.2 Umsetzungsbeschreibung 28](#_Toc509817850)

[7.8 Getting Started Dokumentation 29](#_Toc509817851)

[7.8.1 Anforderung 29](#_Toc509817852)

[7.8.2 Umsetzungsbeschreibung 29](#_Toc509817853)

[8 Realisierung 30](#_Toc509817854)

[8.1 Erstellung des Testkonzepts 30](#_Toc509817855)

[8.2 Anzeige & Speichern der Parameter 30](#_Toc509817856)

[9 Verzeichnisse 31](#_Toc509817857)

[9.1 Tabellenverzeichnis 31](#_Toc509817858)

[9.2 Glossar 31](#_Toc509817859)

[9.3 Abkürzungsverzeichnis 31](#_Toc509817860)

[9.4 Bildverzeichnis 31](#_Toc509817861)

[9.5 Quellenverzeichnis 31](#_Toc509817862)

# 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.

## Allgemeines zur Umgebung

Das Projekt wird in einem eigenen Repository unabhängig vom restlichen Kundenprojekt 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 Visual Studio respektive Webstorm 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 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, welches entgegen des Grundsatzes der Microservices-Architektur wäre, dass jeder Service unabhängig von einem anderen laufen kann. Wenn dies der Fall ist, hat man einen der grössten Vorteile der Microservices-Architektur.

## 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
* Defaultwerte 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 Parameterverwaltung 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 Parameterverwaltun nicht läuft
* Die PV funktioniert rechnerübergreifend, ohne File-Sharing
* Es werden sprechende Namen für Klassen / Methoden verwendet
* 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

## Vorkenntnisse

* C# Programmierung während der Lehre bei der Greenshare AG / CM Informatik AG
* 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

## Mittel & Methoden

### Eingesetzte Mittel

* Visual Studio für die Backendprogrammierung in C#
* Webstorm für die Frontendprogrammierung in TypeScript
* Microsoft Office für die Erstellung der Dokumentationff
* officeatwork Vorlagen, um das CI/CD der Dokumentation einzuhalten
* Draw.io für das Zeichnen der Diagramme.
* 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 Datensicherung wird dennoch jeder Commit auch gleich gepusht. Zudem wird es jeden Tag eine Version geben.

### Daily Business

Am Ende jedes Tages gibt es den Tagesabschluss. Bei diesem wird das Arbeits- und Tagesjournal geschrieben. Nachdem das Journal fertig verfasst ist, wird es Martin Tinner, dem IPA Betreuer, gegeben. Er unterschreibt das Tagesjournal und gibt es mir wieder retour. Anschliessend mache ich ein Rundmail, in dem ich das Tagesjournal an Martin Tinner, Matthias Hess (technischer Betreuer) und Marco Zollinger (Projektleiter des Viaduc Projekts) schicke.

### Dokumentablage

|  |  |
| --- | --- |
|  | Client  In diesem Ordner befindet sich das Webstorm Projekt für den Client-Teil der IPA. Dieser liegt in derselben Struktur wie das Livesystem. |
| **Doc**  Hier drin befindet sich die Dokumentation der IPA, der Zeitplan, die Kriterien und die Vorlage für das Tagesjournal. |
| **Arbeits- & Tagesjournal**  Der Ordner beinhaltet sämtliche ausgefüllte Arbeits- und Tagesjournale. |
| **Diagramme**  Der Ordner enthält eine Kopie jedes Diagramms als PNG. |
| **Screenshot**  Sämtliche Screenshots, die in der Doku verwendet werden, werden auch in diesem Ordner abgelegt. |
| **Server**  In diesem Ordner liegt das ganze Visual Studio Projekt des Backend. |

Tabelle 1 - Dokumentablage

### Kontroll-Tasks

* Nach den IPERKA Phasen planen, entscheiden und realisieren wird das Dokument zum Gegenlesen gegeben. Am anschliessenden Tag wird jeweils eine Stunde eingerechnet, um zu besprechen und allfällige Fehler zu beheben.
* Jeweils am Ende einer Woche, wird die Anforderungsliste mit dem Ist-Stand verglichen.

# 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 officeatwork zu entnehmen |

Tabelle 2 - Konventionen Dokumentation

### Code-Konventionen C#

|  |  |
| --- | --- |
| Bereich | Vorgaben |
| Namensgebung Methoden/Variablen | Methoden- und Variablennamen sollen 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 |

Tabelle 3 - Code-Konventionen C#

### Code-Konventionen TypeScript

|  |  |
| --- | --- |
| Bereich | Vorgaben |
| Namensgebung Methoden/Variablen | Methoden- und Variablennamen sollen 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 anzulegen  KomponentenName.component.(ts | html | less)  Services sind im Services Verzeichnis abzulegen und via Dependency Injection mit Angular anzusteuern. |

Tabelle 4 - Code Konventionen TypeScript

### Layout-Konventionen

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

Tabelle 5 - Layout-Konventionen

## 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

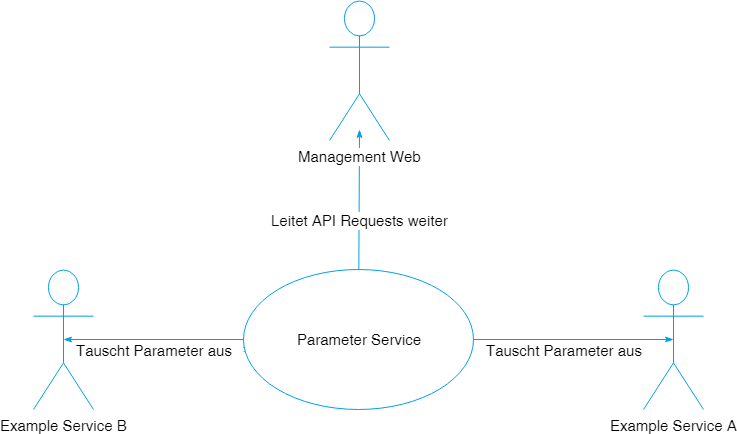
Das gewählte Arbeitsmodell IPERKA bietet sich für diese Projektarbeit an, da sie einen grossen Wert auf die Dokumentation legt. Dadurch, dass die Dokumentation bei dieser Arbeit im Vordergrund steht, kommt diese Stärke hier sehr gut zum Tragen. Die sechs Phasen von IPERKA gehen ineinander über. Welcher 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. In untenstehendem Diagramm sehen wir die Microservices, welche im Rahmen der IPA umgesetzt werden. Sie sind Dummy-Services und bieten verschiedene Parameter an.



## 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.

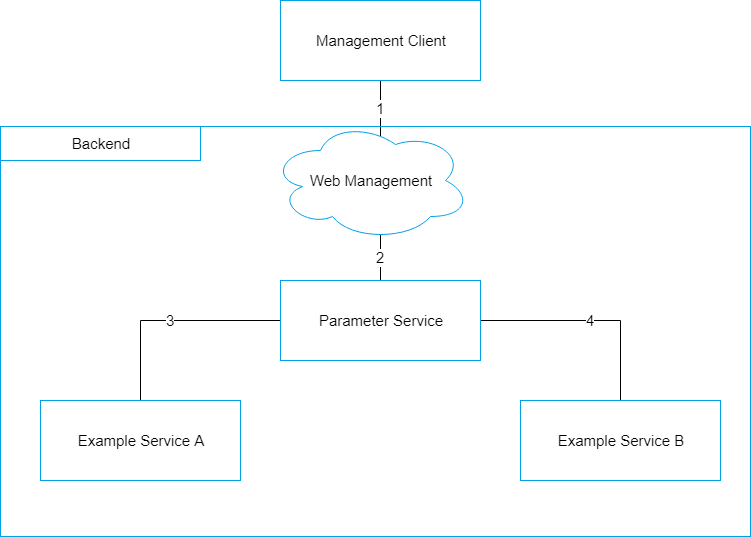
Im Fall dieser Arbeit ist dies die CMI.Contract.Parameter, welche als einzige eine Abhängigkeit hat, eine zentrale Rolle. In dieser DLL werden sämtliche Kommunikationsinterfaces für den Parameter Service definiert.

## 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.

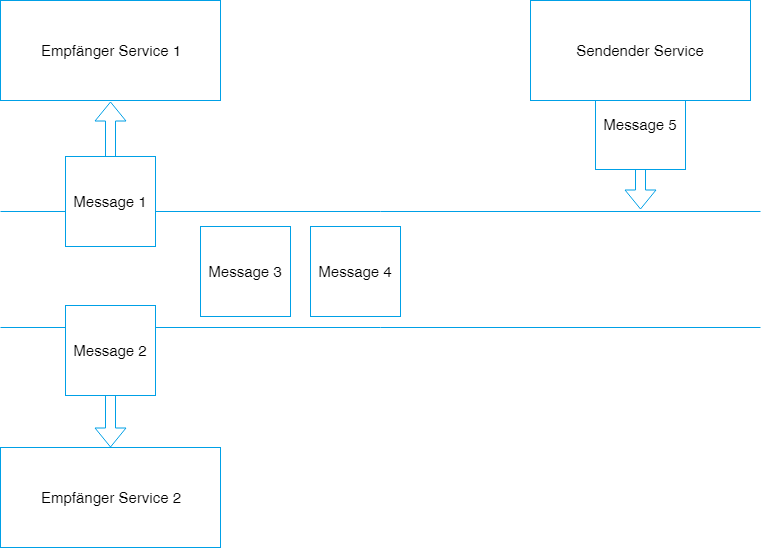
|  |  |
| --- | --- |
| Bezeichnung | Was es ist |
| Management Client | Der Web Client vom Projekt |
| Web Management | Der Webendpunkt, wo sich die APIs befinden |
| Parameter Service | Der Service, um welchen sich die IPA dreht |
| Example Service A & B | Diese zwei Services stellen zwei beliebige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

Tabelle 6 - Systemgrenze Erklärungen Diagramm



## Schnittstellen

Die Kommunikation entsteht durch RabbitMQ und MassTransit. Mit diesen beiden Frameworks ist es möglich verschiedene Kommunikations-Queues zu erstellen. Diese werden dann nach dem First in First out Prinzip abgearbeitet. Das gesamte Backend kommuniziert über solche RabbitMQ Queues. Die Kommunikation ist wie folgt gezeigt sich vorzustellen.



Dieses Diagramm zeigt die Event-Kommunikation von RabbitMQ und MassTransit auf. Der Sender Service gibt Messages in die Queue (der Balken in der Mitte) und von der Queue aus gelangen diese dann an den entsprechenden Empfänger Service und werden so abgearbeitet. Einmal erfolgreich abgearbeitet, werden die Messages aus der Queue entfernt.

# Taskerstellung nach den 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 |

Tabelle 7 - Getting Started Dokumentation Ziele

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

Tabelle 8 - Getting Started Dokumentation Anforderungen

**Geschätzter Aufwand**: 2h

## Einzeltasks für die Realisierung

### Anzeige & Speichern der Parameter

|  |
| --- |
| Erfüllte Ziele |
| Ein Hilfetext kann für jeden Parameter hinterlegt werden |
| Defaultwerte 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 |

Tabelle 9 - Anzeige & Speichern der Parameter Ziele

|  |
| --- |
| 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. |

Tabelle 10 - Anzeige & Speichern der Parameter Anforderungen

**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 |

Tabelle 11 - Implementation des Validierungsmechanismus Ziele

|  |
| --- |
| 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 Defaultwerte genügen. |

Tabelle 12 - Implementation des Validierungsmechanismus Anforderungen

**Geschätzter Aufwand**: 8h

### Suchen eines Parameters

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

Tabelle 13 - Suchen eines Parameters Ziele

|  |
| --- |
| 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. |

Tabelle 14 - Suchen eines Parameters Anforderungen

**Geschätzter Aufwand**: 8h

### Erstellung der Unit Tests und des Testkonzepts

|  |
| --- |
| Anforderung |
| Für Funktionen, welche Daten verarbeiten oder validieren, muss ein mindestens ein Unit-Test geschrieben werden. |

Tabelle 15 - Erstellung der Unit Tests und des Testkonzepts Anforderungen

**Gesamter geschätzter Aufwand:** 4h

## Einzeltaskerstellung der Kontrolltasks

### Durchführen der Tests

|  |
| --- |
| 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 verwendet |
| Ist es nachvollziehbar, warum gerade diese Lösung gewählt wurde? Was waren die Kriterien? |

Tabelle 16 - Durchführen der Tests Ziele

|  |
| --- |
| Anforderung |
| Überprüfung, ob alle oben genannten Ziele erfüllt wurden.  Zudem muss folgendes getestet werden.  **Frontend**   * 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 Tests durchlaufen, sobald diese geschrieben sind. |

Tabelle 17 - Durchführen der Tests Anforderungen

**Geschätzter Aufwand pro Durchgang**: 1h

Dieser Task muss 1 Mal nach jedem realisiertem Realisierung-Punkt umgesetzt werden.

**Gesamter geschätzter Aufwand:** 4h

# Zeitplan

# Detailanalyse der erstellten Tasks

## Umsetzungsreihenfolge

Ich brauche das Testkonzept, um die Tests, welche nicht durch Unittests abgedeckt werden können, zu testen. Ich brauche einen Parameter, um die Validierung schreiben zu können und die Suche baut ebenfalls auf der Validierung auf. Die Erstellung der Unittests kann erst mit der Fertigstellung des restlichen Source Codes abgeschlossen werden. Und zu Letzt die Getting Started Doku, da diese Screenshots und die konkrete Umsetzung benötigt, bevor sie geschrieben werden kann. Dies ergibt folgende Reihenfolge.

|  |  |
| --- | --- |
| Nr. | Task |
| 1 | Erstellung des Testkonzepts |
| 2 | Anzeige & Speichern der Parameter |
| 3 | Implementation des Validierungsmechanismus |
| 4 | Suche eines Parameters |
| 5 | Erstellung der Unittests |
| 6 | Durchführen der Tests |
| 7 | Getting Started |

Tabelle 18 - Umsetzungsreihenfolge

## Erstellung des Testkonzepts

### Anforderung

Das Testkonzept muss alle Test-Cases beinhalten. Es soll möglichst einfach zu testen sein. Zudem darf der Zeitaufwand beim Testen und beim Test schreiben nicht zu gross sein.

### Mögliche Lösungen

****Unittests****

Unittests sind klar definierte Tests, welche eine bestimmte Funktion testen. Sie sind einfach zu schreiben und schnell ausgeführt. Sie zeigen einem auf, wenn der Code etwas falsch gemacht hat. Man kann mit diesen Tests herausfinden, ob erwartete, unerwartete oder falsche Übergabewerte übergeben werden.

Sie sind technisch einfach, können aber GUI-Tests übernehmen.

|  |  |
| --- | --- |
| Kriterium | Punktzahl |
| Einfach auszuführen | 25 / 25 |
| Einfach zu schreiben / definieren | 15 / 25 |
| Zeitaufwand des Tests | 25 / 25 |
| Zeitaufwand des Schreibens der Test | 15 / 25 |
| **Total** | 80 / 100 |

Tabelle 19 - Lösungsmatrix Unittests

Integration Tests / e2e Tests

Integrationstests sind automatisierte Tests, die Businesslogik testen. Sie können verschiedene Teile des Systems testen. Sie sind relativ komplex, da man sich zuerst die ganze Datenstruktur aufbauen muss. Sie sind sehr Zeitaufwendig zu schreiben. E2e Tests ist das Äquivalent zu den Integrationstests für GUIs. Man kann mit ihnen einen Systemzustand simulieren, und schauen, ob dann das Richtige angezeigt werden würde. Für e2e Test fehlt mir jedoch die Erfahrung, da ich noch keine selbst geschrieben habe und mir nach kurzer Recherche auffiel, dass sie den Rahmen dieses Projektes übersteigen würden.

|  |  |
| --- | --- |
| Kriterium | Punktzahl |
| Einfach aus zu führen | 15 / 25 |
| Einfach zu schreiben / definieren | 10 / 25 |
| Zeitaufwand des Tests | 25 / 25 |
| Zeitaufwand des Schreibens der Test | 0 / 25 |
| **Total** | 50 / 100 |

Tabelle 20 - Lösungsmatrix Integration Tests / e2e Tests

User-Testing

Das User-Testing ist das Testen eines Benutzers von Hand, ob die Applikation, dass macht was von ihr erwartet wird. Es sind zeitaufwendige Tests, die besonders für Layout and Anzeige gut zu gebrauchen ist, da

|  |  |
| --- | --- |
| Kriterium | Punktzahl |
| Einfach aus zu führen | 25 / 25 |
| Einfach zu schreiben / definieren | 25 / 25 |
| Zeitaufwand des Tests | 5 / 25 |
| Zeitaufwand des Schreibens der Test | 25 / 25 |
| **Total** | 80 / 100 |

Tabelle 21 - Lösungsmatrix User-Testing

### Umsetzungsbeschreibung

Für das Testkonzept werde ich eine Mischung aus den zwei Siegern machen. Im Frontend sind die Anforderungen, die getestet werden müssen nicht mit Unit-Tests abdeckbar. Jedoch sind e2e Tests viel zu aufwendig. Im Backend macht ein User-Testing jedoch keinen Sinn, weshalb das Backend mit Unittests abgedeckt wird.

## Anzeige & Speichern der Parameter

### Anforderung

Die Parameter dürfen nicht zentral gespeichert werden. Die Microservices selbst müssen den ganzen Aufbau und die Konfiguration der Parameter selbst wissen. Der Parameterservice trägt diese dann zusammen. So muss jeder Service seine Parameter selbständig lesen und schreiben können.

### Mögliche Lösungen Speichern

Speichern der Parameter als Parametertyp im Json

Mit der Newtonsoft Json Drittkomponente z.B., kann man eine C# Klasse zu einem Json serialisieren. Dies mit nur wenigen Zeilen Code. Ein solches Json File zu erweitern, heisst, dass man es clientseitig auch anpassen muss. Somit muss für eine Parameteranpassung auch eine Layoutanpassung erfolgen.

|  |  |
| --- | --- |
| Kriterium | Punktzahl |
| Einfach aus zu lesen | 25 / 25 |
| Einfach zu schreiben | 25 / 25 |
| Einfach erweiterbar | 5 / 25 |
| Fix definierte Form | 0 / 25 |
| **Total** | 55 / 100 |

Tabelle 22 - Lösungsmatrix Speichern der Parameter als Parametertyp im Json

Speichern der Parameter als generischer Typ im Json

Das Speichern der Parameter als generischer Typ bietet den Vorteil, dass man ein klar definiertes Json Schema hat. Dies ist dann in jedem Service gleich abgelegt und kann im Frontend dann für jeden Parameter gleichbehandelt werden. Der Nachteil dieser Variante ist, dass es komplizierter ist zum Schreiben, als ein Framework zu verwenden. Via C# Reflection kann ein Parameter generisch ausgelesen werden.

|  |  |
| --- | --- |
| Kriterium | Punktzahl |
| Einfach aus zu lesen | 10 / 25 |
| Einfach zu schreiben | 15 / 25 |
| Einfach erweiterbar | 25 / 25 |
| Fix definierte Form | 25 / 25 |
| **Total** | 75 / 100 |

Tabelle 23 - Lösungsmatrix Speichern der Parameter als generischer Typ im Json

### Umsetzungsbeschreibung Speichern

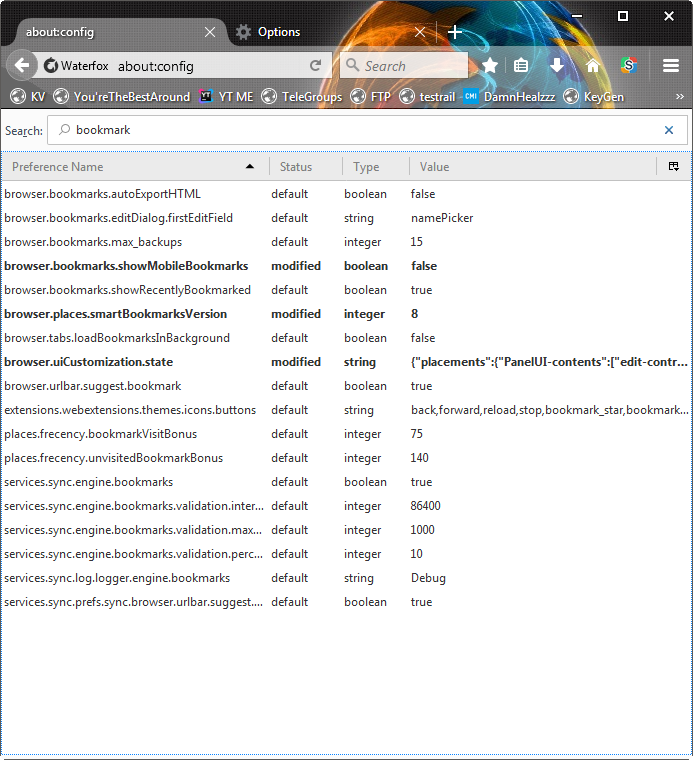
Das Speichern der Parameter als generischer Typ im Json scheint die bessere Option zu sein, da man klar definiert wie das Json auszusehen hat. Da jeder Service seine Parameter selbst speichert, ist es natürlich dem Service selbst überlassen. Jedoch ist zur Versendung der Parameter eine Vereinheitlichung notwendig. Der Aufbau der Versendungsklasse ist dann auch idealerweise der Aufbau des Jsons.

Das Json muss folgende Informationen des Parameters beinhalten:

* Name
* Value
* Typ
* Beschreibung
* Pflichtfeld
* Wie der Parameter zu validieren ist.

### Mögliche Lösung Anzeige

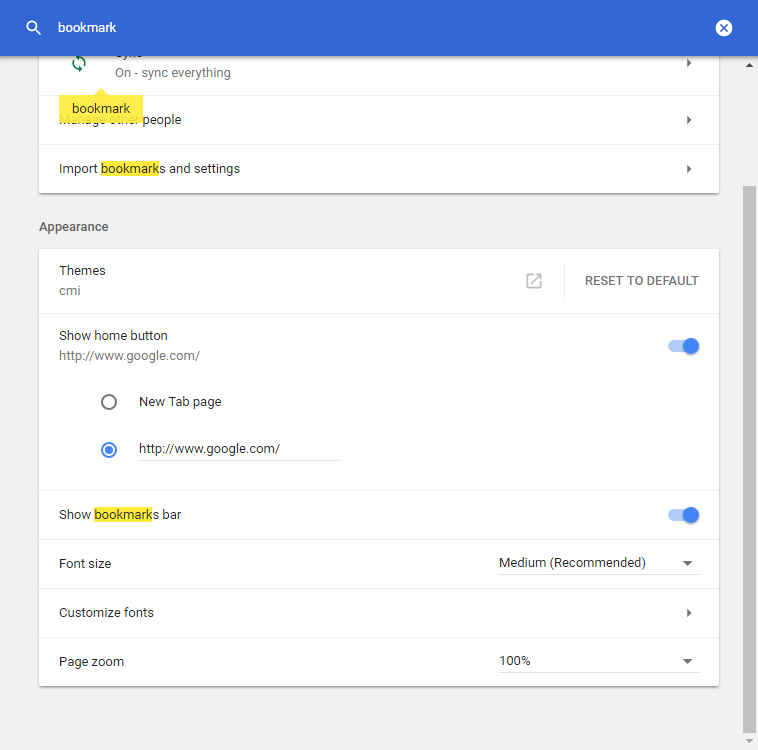
Ähnlich wie in Firefox / Waterfox die about:config



Man listet einfach alle Parameter. Mit einem Doppelklick auf den Parameter kann man diesen editieren und speichern.

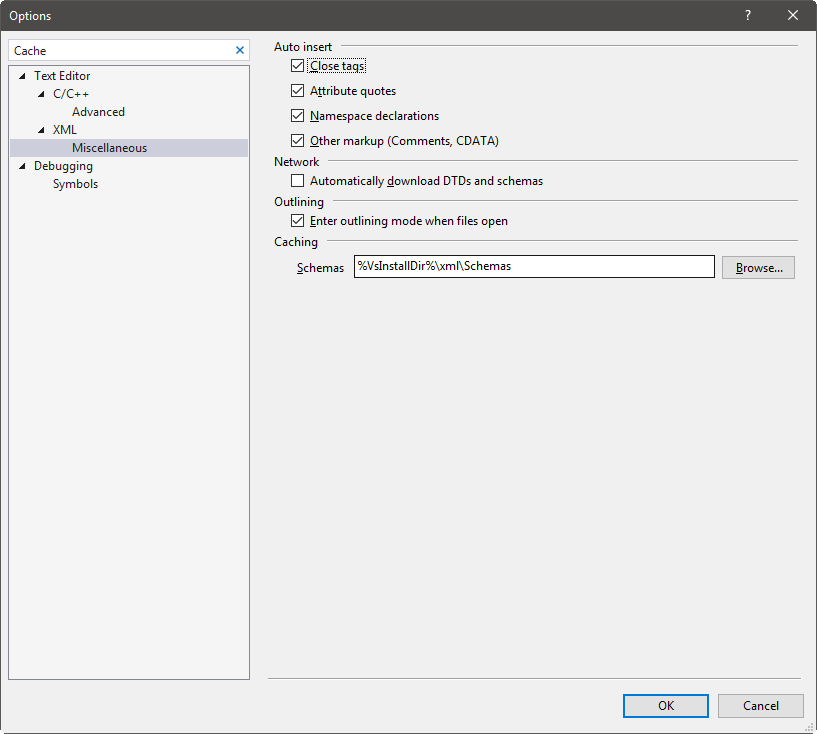
Dies hat den Vorteil, dass es einfach zu programmieren ist, da jeder Parameter gleich aussieht. Es gibt keine Möglichkeit die Parameter zu validieren, was man noch hinzufügen müsste. Die Parameter sind nicht gruppiert, es gibt aber eine Suchmöglichkeit, die das Ganze ziemlich angenehm zu bedienen macht.

Ähnlich wie die Chrome Settings



Die Chrome Settings Page erlaubt ebenfalls das Suchen nach einem Begriff. Es wird alles hervorgehoben, was gefunden wurde. Die Page ist strukturiert aufgebaut, jedoch nicht ganz so einfach um etwas zu finden, wenn man nicht genau weiss wo. Es ist modern aufgebaut und besitzt die Funktion alles auf den Defaultwert zu setzen. Der Standardwert wird oftmals mit (Recommended) gekennzeichnet.

Ähnlich wie in Visual Studio



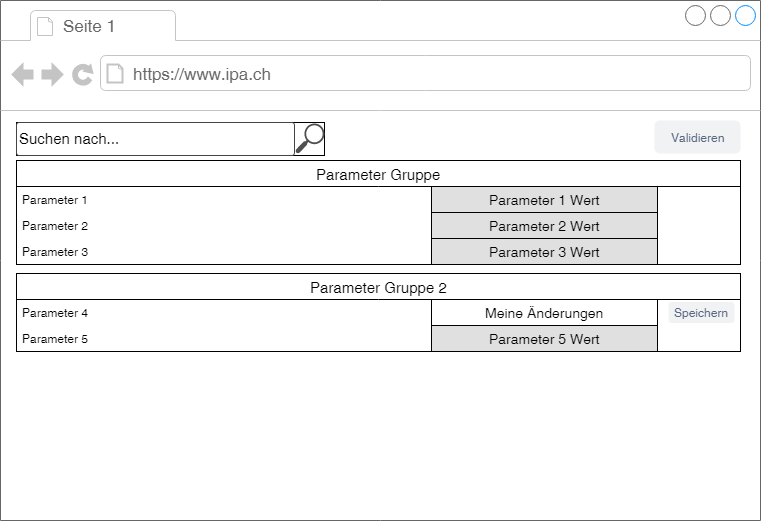
Visual Studio hat ebenfalls viele Parameter zu verwalten genau wie die Browser. In Visual Studio jedoch gibt es viele verschiedene Parameter, welche verschiedene Editoren braucht. Oftmals sind es auch nur Checkboxen. Es unterstützt das Auffinden von Parametern, jedoch nur schlecht. Das Gute an dieser Lösung ist, die hierarchische Gliederung der Parameter. Denn es kann so schnell eingegrenzt werden wo etwas sein sollte. Diese ist gut, wenn man sich mit dem Tool nicht so auskennt und nicht weiss, wie welcher Parameter genau heisst.

### Umsetzungsbeschreibung Anzeige

Alle der drei vorgestellten Möglichkeiten haben Vor- und Nachteile. Da im Projekt Bedingungen herrschen, denen keine der drei Lösungen entspricht, wird hier von allem das Beste genommen.

Es wird die Parameternamensgebung von Firefox, die Gruppierung von Visual Studio via der Parameternamensgebung und die Suche wie in Chrome verwendet um so eine ideale Lösung zu gestallten. Wobei die Gruppierung optional vorgenommen werden kann, wenn noch Zeit dafür ist.

Ein grobes Konzept könnte wie folgt aussehen:



Klickt man in ein Feld, so wird der Bearbeitungsmodus aktiviert und der Speichern-Button erscheint. Ist man mit den Änderungen fertig, klickt man auf «Speichern» und der Parameter wird gespeichert. Dies ist das gleiche Verhalten wie bei Firefox, nur dass in Firefox ein Pop-Up aufgeht. Dies ist jedoch für dieses Projekt nicht von Vorteil, falls man viele Parameter auf einmal editieren muss.

Der Hilfetext ist via Mouseover angezeigt und der Standardwert wird vor dem Parameterwert angezeigt.

Ist bei der Validierung ein Parameter nicht ok. So soll dieser rot hervorgehoben werden. Wird er mit Hilfe der Suche gefunden, so wird der Parametername gelb hervorgehoben.

## Implementation des Validierungsmechanismus

### Anforderung

Der Validierungsmechanismus muss mehrere verschiedene Formate zulassen. Zudem muss der Validierungsmechanismus auch separat, nicht beim Speichern aufgerufen werden können.

### Mögliche Lösungen

Nur serverseitig im Service den Parameter selbst testen lassen

|  |  |
| --- | --- |
| Kriterium | Punktzahl |
| Sicher gegen Angriffe (z.B. XSS) | 25 / 25 |
| Clientseitige Validierung möglich | 0 / 25 |
| Serverseitige Validierung möglich | 25 / 25 |
| Verschieden Formate unterstützt | 25 / 25 |
| **Total** | 75 / 100 |

Tabelle 24 - Lösungsmatrix nur Serverseitig im Service den Parameter selbst testen lassen

Regular Expression im Parameter

|  |  |
| --- | --- |
| Kriterium | Punktzahl |
| Sicher gegen Angriffe (z.B. XSS) | 25 / 25 |
| Clientseitige Validierung möglich | 25 / 25 |
| Serverseitige Validierung möglich | 25 / 25 |
| Verschieden Formate unterstützt | 25 / 25 |
| **Total** | 100 / 100 |

Tabelle 25 - Lösungsmatrix Regular Expression im Parameter

### Umsetzungsbeschreibung

Indem der Parameter einen Regex Ausdruck kennt, kann er sich jeweils selbst validieren. Hierbei spielt es keine Rolle, ob es auf dem Client oder dem Server validiert werden muss. Zudem ist Regex sehr flexibel und man kann alle formatbezogenen und wertbezogenen Validierungen damit machen (z.B. liegt die E-Mailadresse in einer gültigen Form vor, oder ist der Wert nicht grösser als 100).

## Suche eines Parameters

### Anforderung

Die Parameterverwaltung muss das Auffinden eines Parameters unterstützen. Die gefundenen Treffer sollen visuell hervorgehoben werden.

### Mögliche Lösungen

Volltextsuche über alles

Die Volltextsuche schaut jeden Parameter als Ganzes durch. Wird eine Übereinstimmung auf irgendeine Parameter-Property festgestellt, wird der Parameter in der Suche angezeigt.

|  |  |
| --- | --- |
| Kriterium | Punktzahl |
| Findet den Parameter, wenn man dessen Namen kennt | 25 / 25 |
| Findet den Parameter, wenn man dessen Wert kennt | 25 / 25 |
| Findet den Parameter, wenn man weiss, auf welchem Service definiert ist | 25 / 25 |
| Geringe Komplexität der Suche | 15 / 25 |
| **Total** | 90 / 100 |

Tabelle 26 - Lösungsmatrix Volltextsuche über alles

Suche auf die Parameternamen

Mit der Suche nur nach Parameternamen, kann jemand, der genau weiss, nach was er suchen muss sehr schnell fündig.

|  |  |
| --- | --- |
| Kriterium | Punktzahl |
| Findet den Parameter, wenn man dessen Namen kennt | 25 / 25 |
| Findet den Parameter, wenn man dessen Wert kennt | 0 / 25 |
| Findet den Parameter, wenn man weiss, auf welchem Service definiert ist | 25 / 25 |
| Geringe Komplexität der Suche | 25 / 25 |
| **Total** | 75 / 100 |

Tabelle 27 - Lösungsmatrix Suche auf die Parameternamen

Suche auf Parameternamen und Parameterwert

Mit der Suche nach Parameternamen und Parameterwert, behält man die Vorteile der Suche nach dem Parameternamen, fügt aber noch die Möglichkeit hinzu, z.B. Bei Personalwechsel eine E-Mailadresse in den Parametern schnell zu finden.

|  |  |
| --- | --- |
| Kriterium | Punktzahl |
| Findet den Parameter, wenn man dessen Namen kennt | 25 / 25 |
| Findet den Parameter, wenn man dessen Wert kennt | 25 / 25 |
| Findet den Parameter, wenn man weiss, auf welchem Service definiert ist | 25 / 25 |
| Geringe Komplexität der Suche | 15 / 25 |
| **Total** | 90 / 100 |

Tabelle 28 - Lösungsmatrix Suche auf Parameternamen und Parameterwert

### Umsetzungsbeschreibung

Die Volltextsuche auf Parameternamen und Parameterwert scheint am sinnvollsten, da man mit der Volltextsuche über alles nach «max.muster» z.B. auch einen Parameter bekommt, bei dem der Defaultwert die E-Mailadresse «max.muster@example.com» wäre. Dies möchte man aber wahrscheinlich nicht finden. Eine serverseitige suche ist hier nicht notwendig, da kein Paging existiert. So kann man einfach auf dem Client die Daten neu anzeigen lassen. Optional kann bei genügend Zeit noch die ganze Parametergruppe angezeigt werden. Dies setzt jedoch voraus, dass genug Zeit für die Parametergruppe bereits gefunden wurde. Ansonst werden nur die einzelnen Parameter in einer Liste angezeigt.

## Erstellung der Unittests

### Anforderung

Die Unittests müssen die wichtigsten Funktionalitäten des Programmes abdecken um Fehler zu vermeiden. Die Weitergabe von Daten via RabbitMQ/MassTransit kann mit dem zeitlich engen Rahmen dieses Projektes leider nicht getestet werden.

### Umsetzungsbeschreibung

Nur serverseitig müssen Unittests geschrieben werden, da clientseitig nur anzeigende Logik verbaut wird, bräuchte diese e2e Tests. Unittests sind zu wenig aussagekräftig für diesen Anwendungsfall, weshalb sich der Aufwand nicht lohnt. Die einzige clientseitig mit Unittests überprüfte Logik ist die der Clientservices. Da im Rahmen dieser Projektarbeit aber keine neuen Services dazugekommen sind, gibt es keine mit Tests abzudeckenden Clientlogik.

## Durchführen der Tests

### Anforderung

Die Tests müssen nach jedem beendeten Realisierungstask durchgeführt werden und protokolliert werden.

### Umsetzungsbeschreibung

Siehe Kapitel Testergebnisse.

## Getting Started Dokumentation

### Anforderung

Die Anforderungen an diese Dokumentation sind aus der Aufgabenstellung her genau definiert.

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

Tabelle 29 - Getting Started Dokumentation Anforderungen

### Umsetzungsbeschreibung

Das Getting Started ist sinnvollerweise ein separates Word, welches man dann so ablegen kann. Für die Projektdokumentation wird eine exakte Kopie des Getting Started auch noch in die Dokumentation eingefügt.

# Realisierung

## Erstellung des Testkonzepts

|  |
| --- |
| Testfall |
| Ein Benutzer kann einen Parameter einsehen |
| Ein Benutzer kann einen Parameter speichern |
| Ein Benutzer kann nach einem Parameter suchen und findet den entsprechenden Parameter |
| Die gefundenen Parameter sind hervorgehoben |
| Die Webapplikation ist mit der Tastatur bedienbar |
| Der Titel des Parameters ist klar ersichtlich |
| Der Wert des Parameters ist ersichtlich |
| Ein allfälliger Standardwert des Parameters ist ersichtlich |
| Parameter, die über ihn verfügen, können einen Hilfetext anzeigen. |
| Die Unittests laufen fehlerfrei durch. |

Tabelle 30 - Testfälle

Die Testfälle müssen jeweils ausgeführt werden. Ist die Funktionalität hinter einem Testfall noch nicht vorhanden, so wird dies festgehalten.

## Anzeige & Speichern der Parameter

# Verzeichnisse

## Tabellenverzeichnis

[Tabelle 1 - Dokumentablage 9](#_Toc509817751)

[Tabelle 2 - Konventionen Dokumentation 10](#_Toc509817752)

[Tabelle 3 - Code-Konventionen C# 10](#_Toc509817753)

[Tabelle 4 - Code Konventionen TypeScript 10](#_Toc509817754)

[Tabelle 5 - Layout-Konventionen 11](#_Toc509817755)

[Tabelle 6 - Systemgrenze Erklärungen Diagramm 13](#_Toc509817756)

[Tabelle 7 - Getting Started Dokumentation Ziele 15](#_Toc509817757)

[Tabelle 8 - Getting Started Dokumentation Anforderungen 15](#_Toc509817758)

[Tabelle 9 - Anzeige & Speichern der Parameter Ziele 15](#_Toc509817759)

[Tabelle 10 - Anzeige & Speichern der Parameter Anforderungen 15](#_Toc509817760)

[Tabelle 11 - Implementation des Validierungsmechanismus Ziele 16](#_Toc509817761)

[Tabelle 12 - Implementation des Validierungsmechanismus Anforderungen 16](#_Toc509817762)

[Tabelle 13 - Suchen eines Parameters Ziele 16](#_Toc509817763)

[Tabelle 14 - Suchen eines Parameters Anforderungen 16](#_Toc509817764)

[Tabelle 15 - Erstellung der Unit Tests und des Testkonzepts Anforderungen 16](#_Toc509817765)

[Tabelle 16 - Durchführen der Tests Ziele 17](#_Toc509817766)

[Tabelle 17 - Durchführen der Tests Anforderungen 17](#_Toc509817767)

[Tabelle 18 - Umsetzungsreihenfolge 19](#_Toc509817768)

[Tabelle 19 - Lösungsmatrix Unittests 19](#_Toc509817769)

[Tabelle 20 - Lösungsmatrix Integration Tests / e2e Tests 20](#_Toc509817770)

[Tabelle 21 - Lösungsmatrix User-Testing 20](#_Toc509817771)

[Tabelle 22 - Lösungsmatrix Speichern der Parameter als Parametertyp im Json 21](#_Toc509817772)

[Tabelle 23 - Lösungsmatrix Speichern der Parameter als generischer Typ im Json 21](#_Toc509817773)

[Tabelle 24 - Lösungsmatrix nur Serverseitig im Service den Parameter selbst testen lassen 26](#_Toc509817774)

[Tabelle 25 - Lösungsmatrix Regular Expression im Parameter 26](#_Toc509817775)

[Tabelle 26 - Lösungsmatrix Volltextsuche über alles 27](#_Toc509817776)

[Tabelle 27 - Lösungsmatrix Suche auf die Parameternamen 27](#_Toc509817777)

[Tabelle 28 - Lösungsmatrix Suche auf Parameternamen und Parameterwert 28](#_Toc509817778)

[Tabelle 29 - Getting Started Dokumentation Anforderungen 29](#_Toc509817779)

[Tabelle 30 - Testfälle 30](#_Toc509817780)

## Glossar

## Abkürzungsverzeichnis

## Bildverzeichnis

## Quellenverzeichnis