|  |
| --- |
|  |

|  |
| --- |
|  |

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 10](#_Toc510514904)

[1.1 Thema und Zielsetzung 10](#_Toc510514905)

[1.2 Allgemeines zur Umgebung 10](#_Toc510514906)

[1.3 Endprodukt 10](#_Toc510514907)

[2 Rahmenbedingungen 11](#_Toc510514908)

[2.1 Titel der IPA 11](#_Toc510514909)

[2.2 Auslöser der Aufgabenstellung 11](#_Toc510514910)

[2.3 Grobanforderungen 11](#_Toc510514911)

[2.3.1 Parameterverwaltung 11](#_Toc510514912)

[2.3.1.1 Funktionale Anforderungen 11](#_Toc510514913)

[2.3.1.2 Nicht Funktionale Anforderungen 11](#_Toc510514914)

[2.3.2 «Getting Started» Dokumentation 12](#_Toc510514915)

[2.4 Vorkenntnisse 12](#_Toc510514916)

[2.5 Neue Lerninhalte 12](#_Toc510514917)

[2.6 Mittel & Methoden 12](#_Toc510514918)

[2.6.1 Eingesetzte Mittel 12](#_Toc510514919)

[2.6.2 Git Workflow 12](#_Toc510514920)

[2.6.3 Daily Business 12](#_Toc510514921)

[2.6.4 Dokumentablage 13](#_Toc510514922)

[2.6.5 Kontroll-Tasks 13](#_Toc510514923)

[3 Allgemeines 14](#_Toc510514924)

[3.1 Konventionen 14](#_Toc510514925)

[3.1.1 Dokumentation 14](#_Toc510514926)

[3.1.2 Code-Konventionen C# 14](#_Toc510514927)

[3.1.3 Code-Konventionen TypeScript 14](#_Toc510514928)

[3.1.4 Layout-Konventionen 15](#_Toc510514929)

[3.2 Vorgehensmodell 15](#_Toc510514930)

[3.2.1 IPERKA 15](#_Toc510514931)

[4 Umsysteme, Abhängigkeiten, Systemgrenzen und Schnittstellen 16](#_Toc510514932)

[4.1 Umsysteme 16](#_Toc510514933)

[4.2 Abhängigkeiten 16](#_Toc510514934)

[4.3 Systemgrenzen 17](#_Toc510514935)

[4.4 Schnittstellen 18](#_Toc510514936)

[5 Taskerstellung nach den Anforderungen 19](#_Toc510514937)

[5.1 Einzeltasks für die Dokumentation 19](#_Toc510514938)

[5.1.1 Getting Started Dokumentation 19](#_Toc510514939)

[5.2 Einzeltasks für die Realisierung 19](#_Toc510514940)

[5.2.1 Anzeige & Speichern der Parameter 19](#_Toc510514941)

[5.2.2 Implementation des Validierungsmechanismus 20](#_Toc510514942)

[5.2.3 Suchen eines Parameters 20](#_Toc510514943)

[5.2.4 Erstellung der Unit Tests und des Testkonzepts 20](#_Toc510514944)

[5.3 Einzeltaskerstellung der Kontrolltasks 21](#_Toc510514945)

[5.3.1 Durchführen der Tests 21](#_Toc510514946)

[6 Zeitplan 22](#_Toc510514947)

[7 Entscheide der einzelnen Tasks 23](#_Toc510514948)

[7.1 Umsetzungsreihenfolge 23](#_Toc510514949)

[7.2 Erstellung des Testkonzepts 23](#_Toc510514950)

[7.2.1 Anforderung 23](#_Toc510514951)

[7.2.2 Mögliche Lösungen 23](#_Toc510514952)

[7.2.3 Umsetzungsbeschreibung 24](#_Toc510514953)

[7.3 Anzeige & Speichern der Parameter 25](#_Toc510514954)

[7.3.1 Anforderung 25](#_Toc510514955)

[7.3.2 Mögliche Lösungen Speichern 25](#_Toc510514956)

[7.3.3 Umsetzungsbeschreibung Speichern 26](#_Toc510514957)

[7.3.4 Mögliche Lösung Anzeige 26](#_Toc510514958)

[7.3.5 Umsetzungsbeschreibung Anzeige 29](#_Toc510514959)

[7.4 Implementation des Validierungsmechanismus 30](#_Toc510514960)

[7.4.1 Anforderung 30](#_Toc510514961)

[7.4.2 Mögliche Lösungen 30](#_Toc510514962)

[7.4.3 Umsetzungsbeschreibung 30](#_Toc510514963)

[7.5 Suche eines Parameters 31](#_Toc510514964)

[7.5.1 Anforderung 31](#_Toc510514965)

[7.5.2 Mögliche Lösungen 31](#_Toc510514966)

[7.5.3 Umsetzungsbeschreibung 32](#_Toc510514967)

[7.6 Erstellung der Unittests 32](#_Toc510514968)

[7.6.1 Anforderung 32](#_Toc510514969)

[7.6.2 Umsetzungsbeschreibung 32](#_Toc510514970)

[7.7 Durchführen der Tests 32](#_Toc510514971)

[7.7.1 Anforderung 32](#_Toc510514972)

[7.7.2 Umsetzungsbeschreibung 32](#_Toc510514973)

[7.8 Getting Started Dokumentation 33](#_Toc510514974)

[7.8.1 Anforderung 33](#_Toc510514975)

[7.8.2 Umsetzungsbeschreibung 33](#_Toc510514976)

[8 Realisierung 34](#_Toc510514977)

[8.1 Erstellung des Testkonzepts 34](#_Toc510514978)

[8.2 Anzeige & Speichern der Parameter 35](#_Toc510514979)

[8.2.1 Frontend 35](#_Toc510514980)

[8.2.2 Backend 36](#_Toc510514981)

[8.2.3 Technische Umsetzung Backend 36](#_Toc510514982)

[8.3 Implementation des Validierungsmechanismus 37](#_Toc510514983)

[8.4 Suche eines Parameter 37](#_Toc510514984)

[9 Testing 38](#_Toc510514985)

[9.1 Testdurchgang 1 38](#_Toc510514986)

[9.2 Testdurchgang 2 39](#_Toc510514987)

[10 Code Dokumentation Backend 40](#_Toc510514988)

[10.1 Assembly: CMI.Contract.Parameter 40](#_Toc510514989)

[10.1.1 Klasse: ParameterHelper 40](#_Toc510514990)

[10.1.1.1 Methode: GetParameterListFromSetting 40](#_Toc510514991)

[10.1.1.2 Methode: ValidateParameter 41](#_Toc510514992)

[10.1.1.3 Methode: SaveSetting 42](#_Toc510514993)

[10.1.1.4 Methode: GetSetting 43](#_Toc510514994)

[10.1.1.5 Methode: InitialSaveParameter 43](#_Toc510514995)

[10.1.1.6 Methode: GetJsonStringOfSetting 44](#_Toc510514996)

[10.1.1.7 Methode: GetSettingPath 44](#_Toc510514997)

[10.1.1.8 Methode: CreateParameter 45](#_Toc510514998)

[10.1.1.9 Methode: GetType 46](#_Toc510514999)

[10.1.2 Klasse: ParameterBusHelper 46](#_Toc510515000)

[10.1.2.1 Methode: SubscribeGetEvent 46](#_Toc510515001)

[10.1.2.2 Methode: SubscribeSaveEvent 47](#_Toc510515002)

[10.1.3 Klasse: Parameter 47](#_Toc510515003)

[10.1.3.1 Property: Name 47](#_Toc510515004)

[10.1.3.2 Property: Type 47](#_Toc510515005)

[10.1.3.3 Property: Description 48](#_Toc510515006)

[10.1.3.4 Property: Value 48](#_Toc510515007)

[10.1.3.5 Property: Default 48](#_Toc510515008)

[10.1.3.6 Property: RegexValidation 48](#_Toc510515009)

[10.1.3.7 Property: Mandatory 48](#_Toc510515010)

[10.1.4 Interface: ISetting 48](#_Toc510515011)

[10.1.5 Klasse: BusConfigurator 48](#_Toc510515012)

[10.1.6 Ordner: Attributes 49](#_Toc510515013)

[10.1.6.1 Klasse: DefaultAttribute 49](#_Toc510515014)

[10.1.6.2 Klasse: DescriptionAttribute 49](#_Toc510515015)

[10.1.6.3 Klasse: MandatoryAttribute 49](#_Toc510515016)

[10.1.6.4 Klasse: ValidationAttribute 50](#_Toc510515017)

[10.1.7 Ordner: GetParameter 50](#_Toc510515018)

[10.1.7.1 Klasse: GetParameterEvent 50](#_Toc510515019)

[10.1.7.2 Klasse: GetParameterEventResponse 50](#_Toc510515020)

[10.1.7.3 Klasse: GetParameterRequest 50](#_Toc510515021)

[10.1.7.4 Klasse: GetParameterResponse 51](#_Toc510515022)

[10.1.8 Ordner: SaveParameter 51](#_Toc510515023)

[10.1.8.1 Klasse: SaveParameterEvent 51](#_Toc510515024)

[10.1.8.2 Klasse: SaveParameterEventResponse 51](#_Toc510515025)

[10.1.8.3 Klasse: SaveParameterRequest 52](#_Toc510515026)

[10.1.8.4 Klasse: SaveParameterResponse 52](#_Toc510515027)

[10.2 Assembly: CMI.Host.ExampleServiceA 52](#_Toc510515028)

[10.3 Assembly: CMI.Host.ExampleServiceB 52](#_Toc510515029)

[10.4 Assembly: CMI.Host.Parameter 52](#_Toc510515030)

[10.5 Assembly: CMI.Manager.ExampleServiceA 53](#_Toc510515031)

[10.5.1 Klasse: ExampleServiceA 53](#_Toc510515032)

[10.5.1.1 Property: ParameterBus 53](#_Toc510515033)

[10.5.1.2 Methode: Start 53](#_Toc510515034)

[10.5.1.3 Methode: Stop 54](#_Toc510515035)

[10.5.2 Klasse: ExampleSettingA 54](#_Toc510515036)

[10.5.2.1 Property: Date 54](#_Toc510515037)

[10.5.2.2 Property: EMailAdress 54](#_Toc510515038)

[10.5.2.3 Property: ServiceOn 54](#_Toc510515039)

[10.6 Assembly: CMI.Manager.ExampleServiceB 55](#_Toc510515040)

[10.6.1 Klasse: ExampleServiceB 55](#_Toc510515041)

[10.6.1.1 Property: ParameterBus 55](#_Toc510515042)

[10.6.1.2 Methode: Start 55](#_Toc510515043)

[10.6.1.3 Methode: Stop 56](#_Toc510515044)

[10.6.2 Klasse: ExampleSettingB 56](#_Toc510515045)

[10.6.2.1 Property: EndeDatum 56](#_Toc510515046)

[10.6.2.2 Property: EMailAdress 56](#_Toc510515047)

[10.6.2.3 Property: FehlerVerstecken 56](#_Toc510515048)

[10.6.2.4 Property: Sekunden 56](#_Toc510515049)

[10.7 Assembly: CMI.Manager.Parameter 57](#_Toc510515050)

[10.7.1 Klasse: GetParameterEventResponseConsumer 57](#_Toc510515051)

[10.7.1.1 Methode: Consume 57](#_Toc510515052)

[10.7.2 Klasse: GetParameterRequestConsumer 57](#_Toc510515053)

[10.7.2.1 Methode: Consume 57](#_Toc510515054)

[10.7.3 Klasse: ParameterRequestResponseHelper 58](#_Toc510515055)

[10.7.3.1 Property: Parameters 58](#_Toc510515056)

[10.7.3.2 Property: SavedSuccessfully 58](#_Toc510515057)

[10.7.4 Klasse: ParameterService 58](#_Toc510515058)

[10.7.4.1 Property: ParameterBus 58](#_Toc510515059)

[10.7.4.2 Methode: Start 59](#_Toc510515060)

[10.7.4.3 Methode: Stop 59](#_Toc510515061)

[10.7.5 Klasse: SaveParameterEventResponseConsumer 60](#_Toc510515062)

[10.7.5.1 Methode: Consume 60](#_Toc510515063)

[10.7.6 Klasse: SaveParameterRequestConsumer 61](#_Toc510515064)

[10.7.6.1 Consume 61](#_Toc510515065)

[10.8 Assembly: CMI.Web.Management 62](#_Toc510515066)

[10.8.1 Ordner: Controllers 62](#_Toc510515067)

[10.8.1.1 Klasse: ParameterController 62](#_Toc510515068)

[10.8.1.1.1 Methode: GetAllParameters 62](#_Toc510515069)

[10.8.1.1.2 Methode: SaveParameter 62](#_Toc510515070)

[10.8.2 Ordner: Helpers 63](#_Toc510515071)

[10.8.2.1 Klasse: BusHelper 63](#_Toc510515072)

[10.8.2.1.1 Property: ParameterBus 63](#_Toc510515073)

[10.8.2.1.2 Klasse: BusHelper 63](#_Toc510515074)

[11 Code Dokumentation Frontend 64](#_Toc510515075)

[11.1 Ordner: services 64](#_Toc510515076)

[11.1.1 Klasse: HttpService 64](#_Toc510515077)

[11.1.2 Klasse: ParameterService 64](#_Toc510515078)

[11.1.2.1 Methode: getAllParameters 64](#_Toc510515079)

[11.1.2.2 Methode: saveParameter 64](#_Toc510515080)

[11.1.2.3 Methode: \_createBaseUrl 65](#_Toc510515081)

[11.2 Ordner: highlight 65](#_Toc510515082)

[11.2.1 Klasse: HighlightComponent 65](#_Toc510515083)

[11.2.1.1 Property: highlight 65](#_Toc510515084)

[11.2.1.2 Property: text 65](#_Toc510515085)

[11.2.1.3 Methode: constructor 65](#_Toc510515086)

[11.2.1.4 Methode: getInnerHTML 66](#_Toc510515087)

[11.2.2 HTML: HighlightComponent 66](#_Toc510515088)

[11.3 Ordner: parameterManager 66](#_Toc510515089)

[11.3.1 Klasse: Parameter 66](#_Toc510515090)

[11.3.1.1 Property: name 66](#_Toc510515091)

[11.3.1.2 Property: value 67](#_Toc510515092)

[11.3.1.3 Property: type 67](#_Toc510515093)

[11.3.1.4 Property: mandatory 67](#_Toc510515094)

[11.3.1.5 Property: description 67](#_Toc510515095)

[11.3.1.6 Property: regexValidation 67](#_Toc510515096)

[11.3.1.7 Property: default 67](#_Toc510515097)

[11.4 Ordner: parameter 67](#_Toc510515098)

[11.4.1 Klasse: ParameterComponent 67](#_Toc510515099)

[11.4.1.1 Property: parameter 68](#_Toc510515100)

[11.4.1.2 Property: validationEvent 68](#_Toc510515101)

[11.4.1.3 Property: searchString 68](#_Toc510515102)

[11.4.1.4 Property: active 68](#_Toc510515103)

[11.4.1.5 Property: value 68](#_Toc510515104)

[11.4.1.6 Property: checked 68](#_Toc510515105)

[11.4.1.7 Property: validationError 69](#_Toc510515106)

[11.4.1.8 Property: \_onFocusChange 69](#_Toc510515107)

[11.4.1.9 Methode: constructor 69](#_Toc510515108)

[11.4.1.10 Methode: ngOnInit 69](#_Toc510515109)

[11.4.1.11 Methode: onValueChanged 70](#_Toc510515110)

[11.4.1.12 Methode: onFocus 70](#_Toc510515111)

[11.4.1.13 Methode: saveParameter 70](#_Toc510515112)

[11.4.1.14 Methode: cancelEdit 71](#_Toc510515113)

[11.4.1.15 Methode: \_isValid 71](#_Toc510515114)

[11.4.1.16 Methode: \_validateString 71](#_Toc510515115)

[11.4.1.17 Methode: getErrorClass 72](#_Toc510515116)

[11.4.1.18 Methode: getInputClass 72](#_Toc510515117)

[11.4.2 HTML: ParameterComponent 73](#_Toc510515118)

[11.4.3 Less: ParameterComponent 74](#_Toc510515119)

[11.5 Ordner: ParameterList 75](#_Toc510515120)

[11.5.1 Klasse: ParameterListComponent 75](#_Toc510515121)

[11.5.1.1 Property: loading 75](#_Toc510515122)

[11.5.1.2 Property: filteredParameters 75](#_Toc510515123)

[11.5.1.3 Property: \_allParameters 75](#_Toc510515124)

[11.5.1.4 Property: validationEvent 75](#_Toc510515125)

[11.5.1.5 Property: searchString 75](#_Toc510515126)

[11.5.1.6 Property: searchedStringUpToDate 76](#_Toc510515127)

[11.5.1.7 Methode: constructor 76](#_Toc510515128)

[11.5.1.8 Methode: getAllParameters 76](#_Toc510515129)

[11.5.1.9 Methode: onValueChanged 76](#_Toc510515130)

[11.5.1.10 Methode: emitValidationEvent 77](#_Toc510515131)

[11.5.1.11 Methode: searchParam 77](#_Toc510515132)

[11.5.2 HTML: ParameterListComponent 78](#_Toc510515133)

[11.5.3 Less: ParameterListComponent 79](#_Toc510515134)

[12 Verzeichnisse 80](#_Toc510515135)

[12.1 Glossar 80](#_Toc510515136)

[12.2 Tabellenverzeichnis 82](#_Toc510515137)

[12.3 Bildverzeichnis 83](#_Toc510515138)

[12.4 Abkürzungsverzeichnis 83](#_Toc510515139)

[12.5 Quellenverzeichnis 83](#_Toc510515140)

# 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 die Layoutrichtlinien 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

|  |  |
| --- | --- |
| Abbildung 1 - Struktur der 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. |
| Dokumentvorlage | 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 |
| Layoutrichtlinien | Richtlinien des Bundes ist ein zu halten. Sagt dieses nichts aus ist auf Bootstrap zurückzugreifen. Ist da ebenfalls nichts definiert, so muss selbst ein Vorschlag gemacht werden. |
| Barrierefreiheit | Es muss alles mit der Tastatur bedienbar sein. |

Tabelle 5 - Layout-Konventionen

### Unittests-Konventionen

|  |  |
| --- | --- |
| Bereich | Vorgabe |
| Assemblyname | ZuTestendesAssembly.Tests |
| Methodennamen | Upper Snake Case |

## 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

Abbildung 2 - 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 statt. 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.

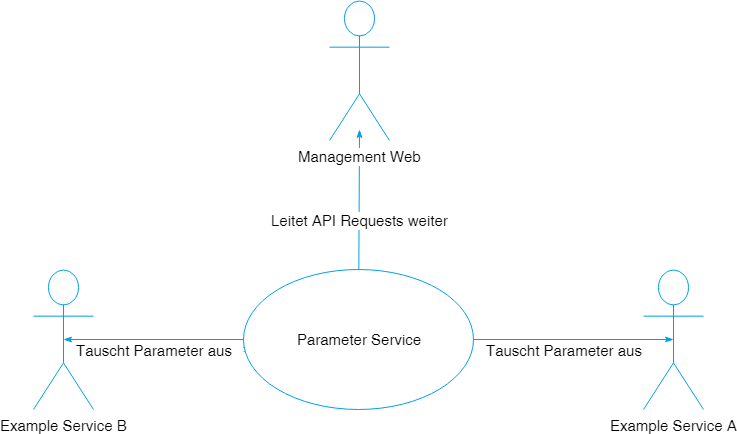


Abbildung 3 - Umsysteme

## 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 Ort, an welchem 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

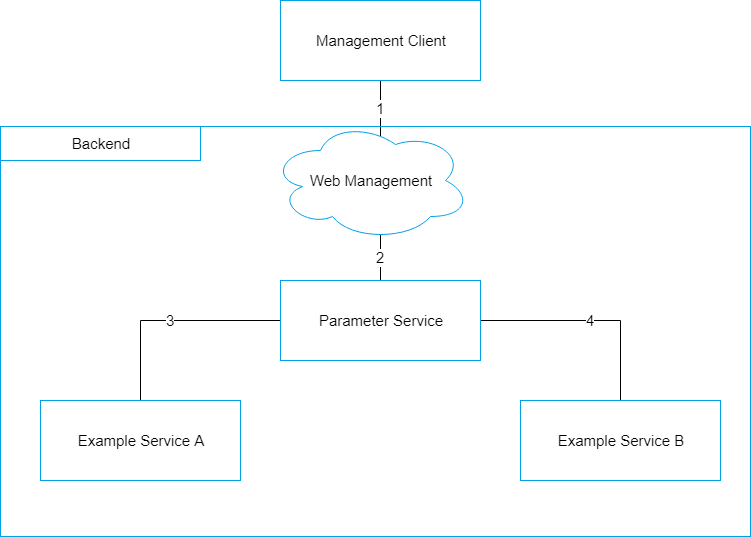


Abbildung 4 - Systemgrenzen

## 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:

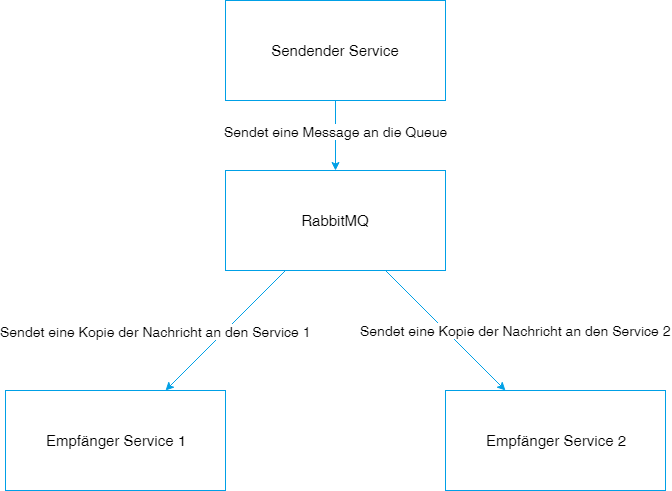


Abbildung 5 - RabbitMQ Event Kommunikation

Dieses Diagramm zeigt die Event-Kommunikation von RabbitMQ und MassTransit auf. Der Sender Service gibt Messages in die Queue und von der Queue aus gelangt dann eine Kopie der Nachricht an jeden Subscriber(Empfänger Service) gesendet. 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 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 die Layoutrichtlinien 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 realisierten Realisierung-Punkt umgesetzt werden.

**Gesamter geschätzter Aufwand:** 4h

# Zeitplan

# Entscheide der einzelnen 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, das macht was von ihr erwartet wird. Es sind zeitaufwendige Tests, die besonders für Layout and Anzeige gut zu gebrauchen ist, da eine Computer Tests wie z.B. sieht die Webseite optisch gut aus nicht testen kann.

|  |  |
| --- | --- |
| 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 nur 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 zum Schreiben ist, 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

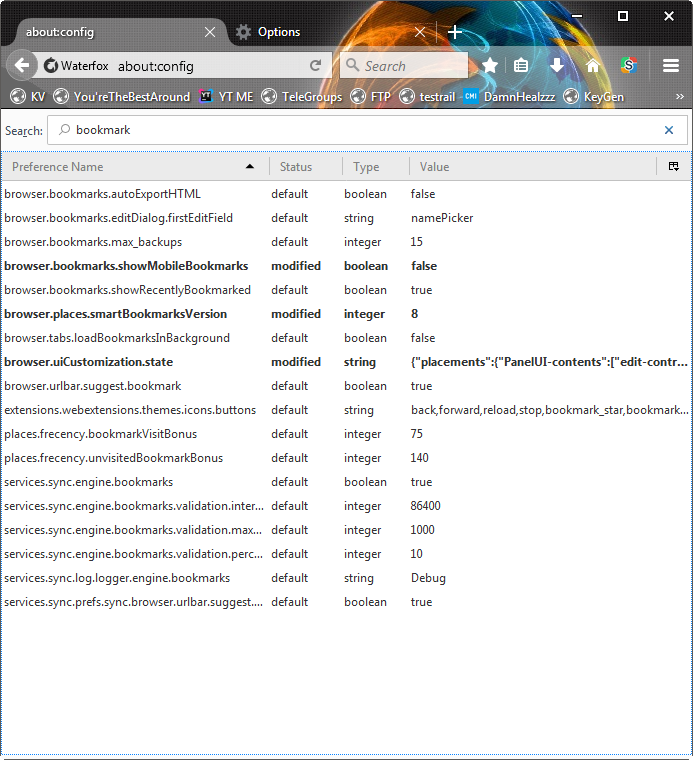


Abbildung 6 - Firefox/Waterfox Einstellungen

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

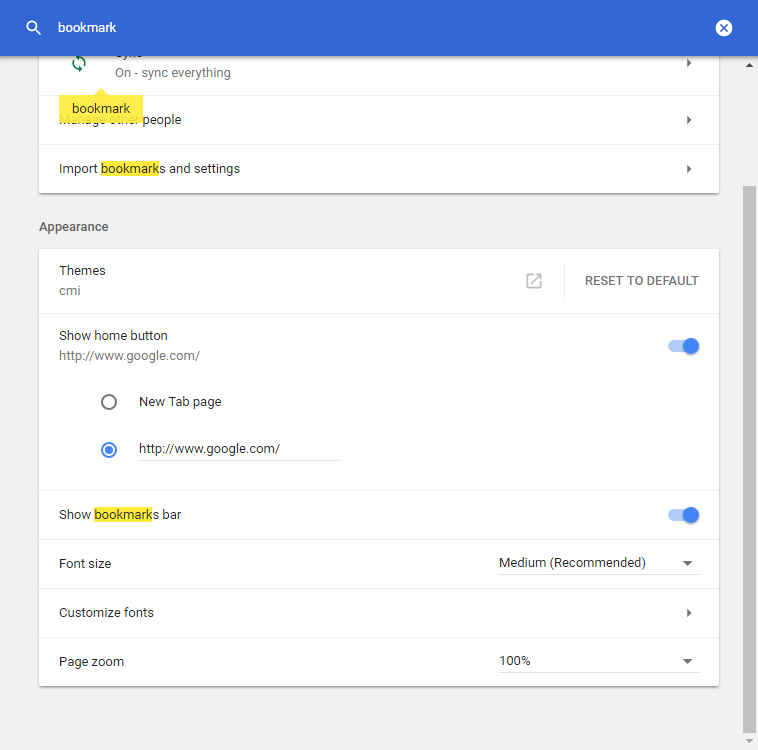


Abbildung 7 - Chrome Einstellungen

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

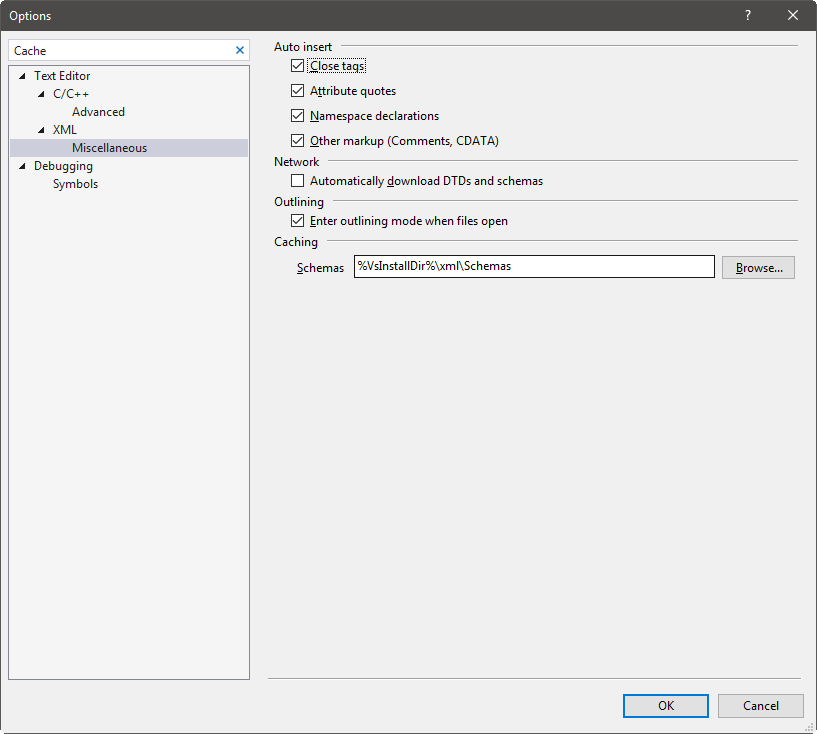


Abbildung 8 - Visual Studio Einstellungen

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.

Damit eine ideale Lösung gestaltet werden kann wird die Parameternamensgebung von Firefox, die Gruppierung von Visual Studio via den Parameternamen und die Suche wie in Chrome verwendet. Wobei die Gruppierung optional vorgenommen werden kann, wenn noch Zeit dafür ist.

Ein grobes Konzept könnte wie folgt aussehen:

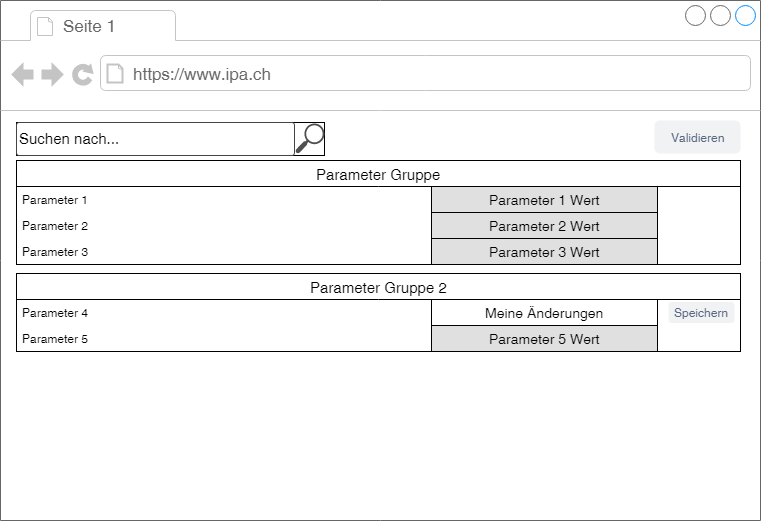


Abbildung 9 - Mockup

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. Ansonsten 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. |
| Layoutrichtlinien wurde eingehalten |
| Es können keine falschen Parameter gespeichert werden |

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

### Frontend

Die Umsetzung im Frontend ist einfach aufgebaut und intuitiv. Man hat einen Parameter pro Zeile. Wird der Fokus auf das Inputfeld gelegt, so wird der Bearbeitungsmodus gesetzt und der «Speichern»- respektive «Abbrechen»-Knopf erscheint. Diese verfügen über die vermutete Funktionalität. Speichern speichert den Parameter, Abbrechen bricht ab und setzt den Wert des Parameters auf den vom Server geladenen Wert. Dieses Ergebnis wird erreicht, indem das Inputfeld an eine separate Variable gebunden wird. So wird das Serverobjekt nur verändert, wenn gespeichert wurde. Springt man aus dem Inputfeld heraus, so wird ebenfalls der ungespeicherte Text zurückgesetzt. Der Hinweistext des Parameters ist als Title-Attribut des Parameters ausgegeben. So wird mit dem Hinweistext beim Mouseover angezeigt. Es kann jeweils nur genau ein Parameter im Bearbeitungsmodus sein. Dies wird via einem «static Subject» erreicht. Dies erlaubt das Senden einer Nachricht an alle Parameter. In dem Empfänger wird der Bearbeitungsmodus auf jedem deaktiviert. Der Event wird auf dem OnFocus Event des Inputfelds ausgelöst. Nach dem auslösen des Events, wird der aktuelle Parameter in Bearbeitung gesetzt.

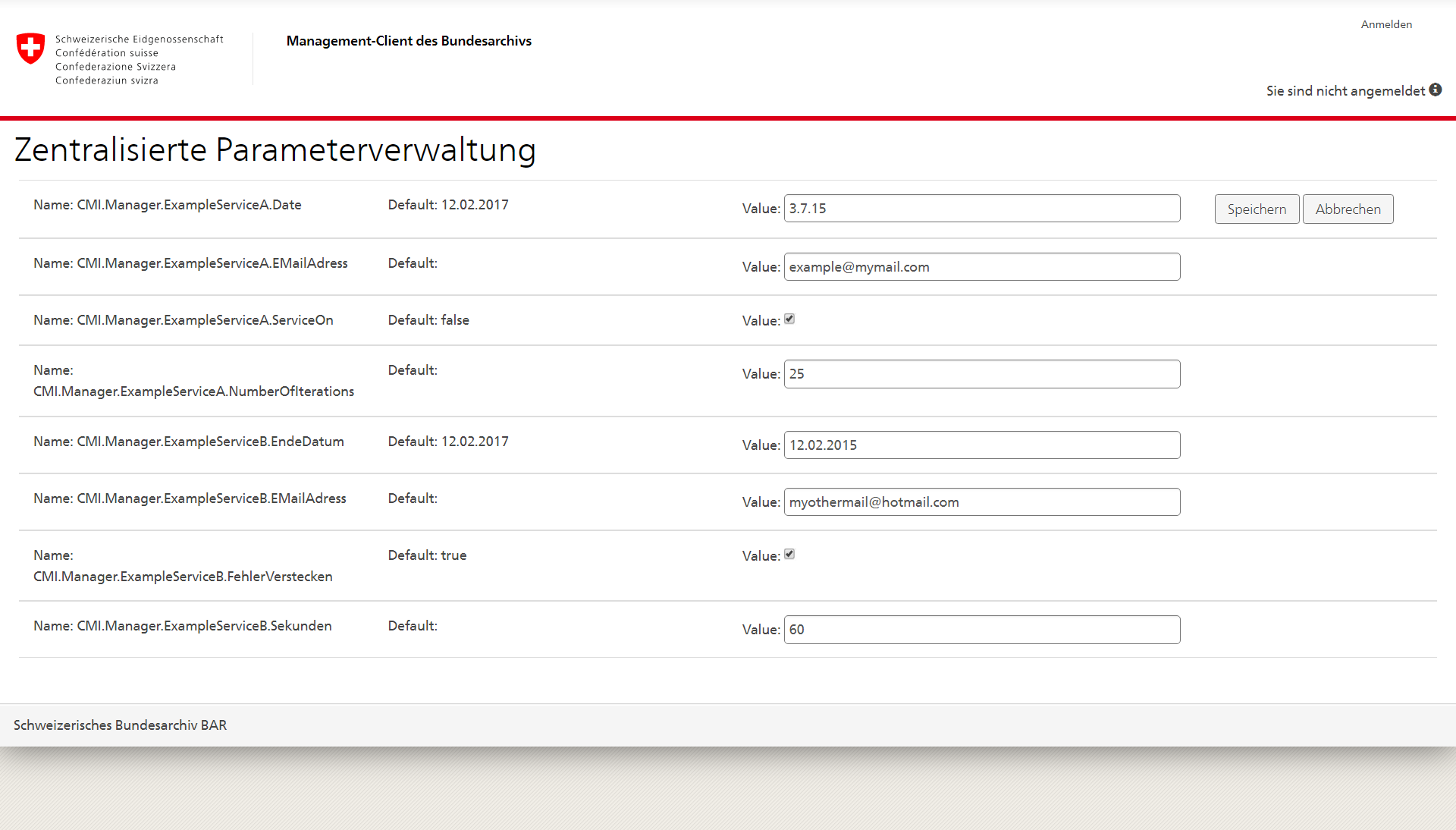


Abbildung 10 - Screenshot umgesetzte Lösung

### Backend

Der Backendbereich ist Zuständig für das Speichern und Verwalten der Parameter. Es können nur alle Parameter geladen werden und nur 1 Parameter auf einmal gespeichert werden. Die ganze Kommunikation sieht wie folgt aus:

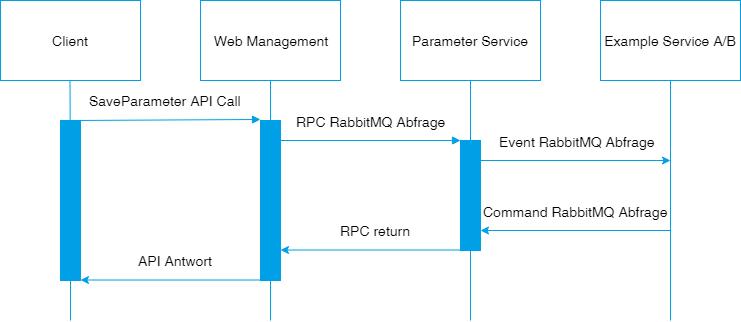


Abbildung 11 - Sequenzdiagramm Servicekommunikation beim «Save»

Example Service A und B werden beide Asynchron angesprochen. Sie melden sich beide beim Parameter Service. Dieser wartet zwischenzeitlich. Nach dem der Parameterservice 400 Millisekunden gewartet hat, gibt dieser alle gesammelten Parameter weiter an den API Controller im Web Management. Anschliessend wird die Liste von Parametern an den Client zurückgesendet.

### Technische Umsetzung Backend

Wird ein Parameter geladen wird die API aufgerufen. Die API holt sich anschliessend via RabbitMQ alle Parameter.

Dies geschieht, mit einem selbst gebautem Serialisierer. Dies ist alles in eine statische Helper Klasse ausgelagert, im CMI.Contract.Parameter. So hat jeder Service, der einen Parameter zum Parametrieren hat, auf die Helperklasse Zugriff.

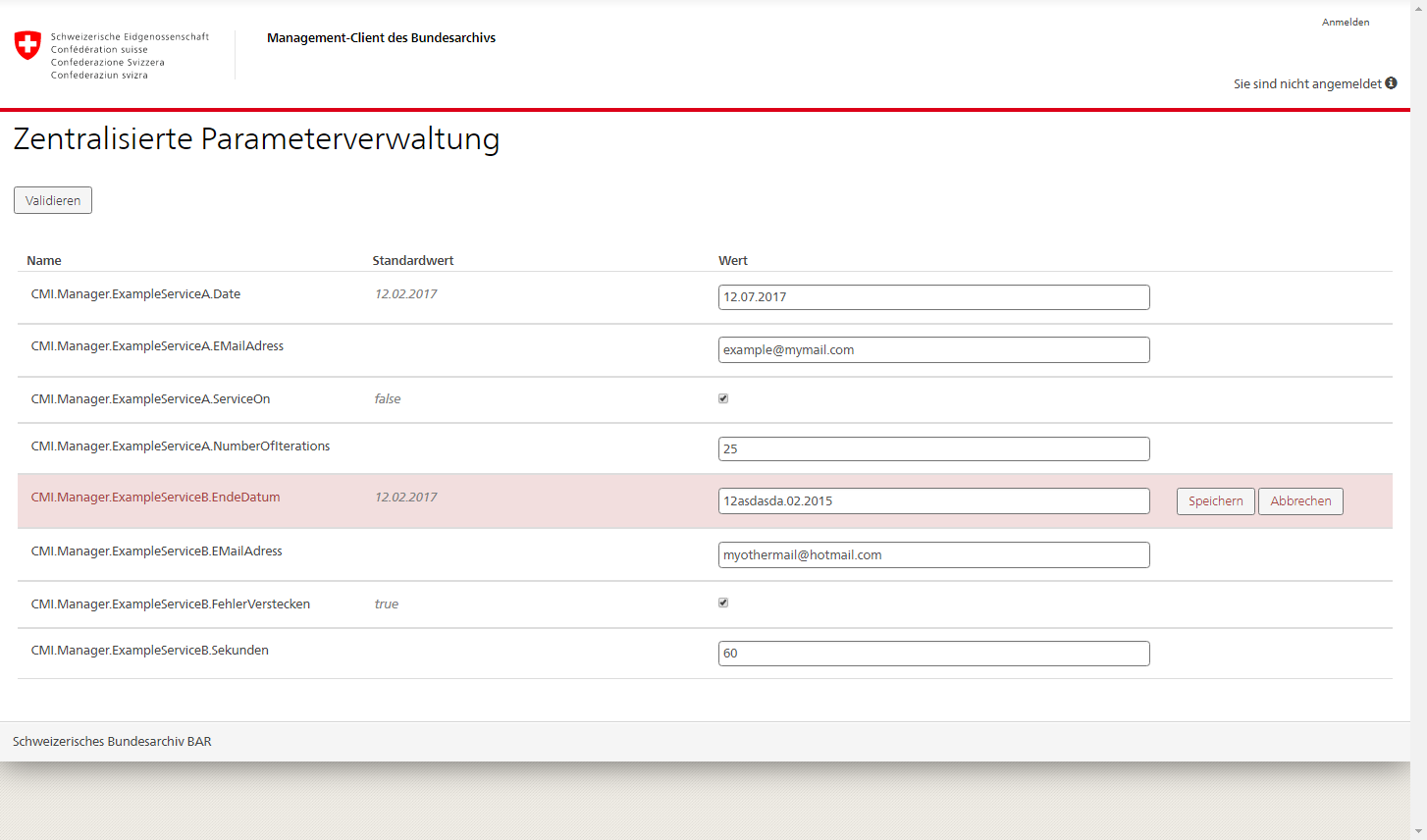
Die erstellten und erweiterten Klassen zu diesem Punkt sind:

**Backend**

* CMI.Contract.Parameter
  + Attributes
    - DefaultAttribute
    - DescriptionAttribute
    - MandatoryAttribute
    - ValidationAttribute
  + GetParameter
    - GetParameterEventResponse
    - GetParameterResponse
  + SaveParameter
    - SaveParameterEvent
    - SaveParameterEventResponse
    - SaveParameterRequest
    - SaveParameterResponse
  + ISetting
  + Parameter
  + ParameterHelper
* CMI.Manager.ExampleServiceA
  + ExampleSettingA
  + ExampleServiceA
* CMI.Manager.ExampleServiceB
  + ExampleSettingB
  + ExampleServiceB
* CMI.Manager.Parameter
  + GetParameterEventResponseConsumer
  + GetParameterRequestConsumer
  + ParameterRequestResponseHelpers
  + ParameterService
  + SaveParameterEventResponseConsumer
  + SaveParameterRequestConsumer
* CMI.Web.Management
  + Controllers
    - ParameterController

## Implementation des Validierungsmechanismus

Der Validierungsmechanismus ist Client und Serverseitig umgesetzt. Es wird beides Mal die gleiche Validierung vorgenommen. Dies ist so umgesetzt um unnötigen Traffic auf den Server zu vermeiden, da kein Request abgesetzt wird, solange es noch Fehler hat. Ist ein Fehler gefunden wird dieser rot hinterlegt. Die Validierung wird beim druck des Validieren Buttons ausgelöst über alle Parameter, oder nachdem auf speichern gedrückt wird der gespeicherte Parameter.



## ****Suche eines Parameters****

Die Suche ist nur im Frontend umgesetzt.

## Schreiben der Unittests

Die Unittests testen nur den CMI.Contract.Parameter Assembly. Darin wird hauptsächlich die ParameterHelper Klasse getestet, da diese alle Datenmanipulationen vornimmt. Die Unittests sind in zwei Klassen geschrieben. Eine um die Validierung zu testen, und eine um die Serialisierung und das Speichern zu testen. Für die Unittests wurden die Klassen ParameterSerializerTests und ParameterValidationTests geschrieben. Als Unittest-Setting wurde die Klasse TestSetting geschrieben.

### ParameterSerializerTests

|  |  |
| --- | --- |
| Test | Aussage |
| An\_empty\_setting\_can\_be\_serialized | Das Serialisieren eines neuen Settings funktioniert. |
| Value\_can\_be\_serialized\_correctly | Die Serialisierung des Werts des Settings funktioniert |
| Default\_can\_be\_serialized\_correctly | Die Serialisierung des Default-Attribut funktioniert. |
| Type\_can\_be\_serialized\_correctly | Die Serialisierung des Typen funktioniert. |
| Description\_can\_be\_serialized\_correctly | Die Serialisierung des Description-Attribut funktioniert. |
| Mandatory\_can\_be\_serialized\_correctly | Die Serialisierung des Mandatory-Attribut funktioniert. |
| Validation\_can\_be\_serialized\_correctly | Die Serialisierung des Validation-Attribut funktioniert. |
| Name\_can\_be\_serialized\_correclty | Die Serialisierung des Namens funktioniert. |
| Can\_be\_serialized\_saved\_and\_can\_then\_be\_deserialized\_and\_got | Das Speichern und Auslesen eines validen Parameters funktioniert. |

### ParameterValidationTests

|  |  |
| --- | --- |
| Test | Aussage |
| ParameterList\_Should\_be\_valid | Ein valider Parameter wird als richtig validiert. |
| Empty\_value\_and\_mandatory\_should\_not\_be\_valid | Ein leerer Pflichtparameter wird als falsch validiert. |
| None\_empty\_value\_and\_mandatory\_should\_be\_valid | Ein ausgefüllter Pflichtparameter wird als richtig validiert. |
| None\_empty\_value\_that\_is\_not\_conform\_should\_not\_be\_valide | Ein falsch ausgefüllter Parameter wird als falsch validiert. |
| None\_empty\_value\_that\_is\_conform\_should\_be\_valide | Ein richtig ausgefüllter Parameter wird als richtig validiert. |
| Empty\_value\_that\_is\_mandatory\_and\_has\_a\_regex\_defined\_should\_not\_be\_valide | Ein leer ausgefüllter Pflichtparameter mit Validierung wird als falsch validiert. |
| None\_empty\_conform\_value\_that\_is\_mandatory\_and\_has\_a\_regex\_defined\_should\_be\_valide | Ein richtig ausgefüllter Pflichtparameter mit Validierung wird als richtig validiert. |
| None\_empty\_none\_conform\_value\_that\_is\_mandatory\_and\_has\_a\_regex\_defined\_should\_not\_be\_valide | Ein falsch ausgefüllter Pflichtparameter mit Validierung wird als falsch validiert. |

# Testing

## Testdurchgang nach Task Anzeige & Speichern der Parameter

|  |  |
| --- | --- |
| Testfall | Resultat |
| Ein Benutzer kann einen Parameter einsehen | Funktioniert nachdem ein kleiner Bug mit Booleanwerten gefixt wurde. |
| Ein Benutzer kann einen Parameter speichern | Funktioniert |
| Ein Benutzer kann nach einem Parameter suchen und findet den entsprechenden Parameter | Noch nicht umgesetzt |
| Die gefundenen Parameter sind hervorgehoben | Noch nicht umgesetzt |
| Die Webapplikation ist mit der Tastatur bedienbar | Funktioniert |
| Der Titel des Parameters ist klar ersichtlich | Funktioniert |
| Der Wert des Parameters ist ersichtlich | Funktioniert |
| Ein allfälliger Standardwert des Parameters ist ersichtlich | Funktioniert |
| Parameter, die über ihn verfügen, können einen Hilfetext anzeigen. | Funktioniert |
| Die Unittests laufen fehlerfrei durch. | Noch nicht umgesetzt |
| Layoutrichtlinien wurde eingehalten | Noch nicht ganz, es klebt alles noch am oberen Rand. Ansonsten ist es eingehalten. |
| Es können keine falschen Parameter gespeichert werden | Noch nicht umgesetzt |

Tabelle 31 - Resultate Testdurchgang 1

Durch diesen Testdurchgang wurde noch ein Bug gefunden, welcher bei Booleanwerten auftrat. Wegen der Gross-/Kleinschreibung waren alle Checkboxen nie angekreuzt. Der Fehler wurde kurzerhand gefixt. Die Anpassung am CSS wurde ebenfalls noch im Rahmen der Korrekturen dieses Testes gemacht.

## Testdurchgang nach Task Implementation des Validierungsmechanismus

|  |  |
| --- | --- |
| Testfall | Resultat |
| Ein Benutzer kann einen Parameter einsehen | Funktioniert |
| Ein Benutzer kann einen Parameter speichern | Funktioniert |
| Ein Benutzer kann nach einem Parameter suchen und findet den entsprechenden Parameter | Noch nicht umgesetzt |
| Die gefundenen Parameter sind hervorgehoben | Noch nicht umgesetzt |
| Die Webapplikation ist mit der Tastatur bedienbar | Funktioniert nicht, der Bearbeitungsmodus wird nicht gesetzt. |
| Der Titel des Parameters ist klar ersichtlich | Funktioniert |
| Der Wert des Parameters ist ersichtlich | Funktioniert |
| Ein allfälliger Standardwert des Parameters ist ersichtlich | Funktioniert |
| Parameter, die über ihn verfügen, können einen Hilfetext anzeigen. | Funktioniert |
| Die Unittests laufen fehlerfrei durch. | Noch nicht umgesetzt |
| Layoutrichtlinien wurde eingehalten | Funktioniert |
| Es können keine falschen Parameter gespeichert werden | Funktioniert |

Tabelle 32 - Resultate Testdurchgang 2

Während der heutigen Entwicklungsarbeiten wurde eine zwischenzeitliche Änderung an dem Fokusevent vorgenommen. Dies musste ich nun wieder zurückbauen, da sonst das Fokussetzen via Tastatur nicht geht.

## Testdurchgang nach Task suche eines Parameters

|  |  |
| --- | --- |
| Testfall | Resultat |
| Ein Benutzer kann einen Parameter einsehen | Funktioniert |
| Ein Benutzer kann einen Parameter speichern | Funktioniert |
| Ein Benutzer kann nach einem Parameter suchen und findet den entsprechenden Parameter | Funktioniert |
| Die gefundenen Parameter sind hervorgehoben | Funktioniert |
| Die Webapplikation ist mit der Tastatur bedienbar | Funktioniert |
| Der Titel des Parameters ist klar ersichtlich | Funktioniert. |
| Der Wert des Parameters ist ersichtlich | Funktioniert |
| Ein allfälliger Standardwert des Parameters ist ersichtlich | Funktioniert |
| Parameter, die über ihn verfügen, können einen Hilfetext anzeigen. | Funktioniert |
| Die Unittests laufen fehlerfrei durch. | Noch nicht umgesetzt |
| Layoutrichtlinien wurde eingehalten | Funktioniert |
| Es können keine falschen Parameter gespeichert werden | Funktioniert |

Tabelle 33 - Resultate Testdurchgang 3

Es wurden in diesem Testdurchgang keine Fehler gefunden.

## Testdurchgang nach Task Erstellung der Unittests

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

Tabelle 34 - Resultate Testdurchgang 4

Es wurden in diesem Testdurchgang keine Fehler gefunden.

# Code Dokumentation Backend

## Assembly: CMI.Contract.Parameter

Die CMI.Contract.Parameter Assembly hat die Aufgabe sämtliche relevanten Informationen und Funktionen zur Verfügung zu stellen, welches die CMI.Manager.Parameter Assembly und die Assembly auf der anderen Seite der RabbitMQ brauchen.

### Klasse: ParameterHelper

Die Klasse bietet Helpermethoden zur Verarbeitung von Parametern zur Verfügung. Die Klasse ist generisch aufgebaut, sodass jede Klasse, die von ISetting erbt, hier verarbeitet, gespeichert und ausgelesen werden kann.

#### Methode: GetParameterListFromSetting

Diese Methode wandelt einen ISetting in eine Parameterliste um.

public static Parameter[] GetParameterListFromSetting (ISetting setting)

{

var paramList = new List<Parameter>();

var namePrefix = setting.GetType().Namespace;

foreach (var fieldInfo in setting.GetType().GetFields())

{

var param = CreateParameter(fieldInfo, setting, namePrefix);

if (param.Name != null)

{

paramList.Add(param);

}

else

{

throw new NullReferenceException();

}

}

return paramList.ToArray();

}

#### Methode: ValidateParameter

Diese Methoden Validieren einen Parameter. Sie geben true zurück, wenn der Parameter gültig ist und false, falls er nicht gültig ist. Die Methode kommt mit zwei Overloads. Einen für einen einzelnen Parameter und einen für eine Liste von Parametern.

public static bool ValidateParameter(Parameter parameter)

{

if (string.IsNullOrEmpty(parameter.Value) && parameter.Mandatory)

{

return false;

}

if (parameter.RegexValidation == null || parameter.Value == null)

{

return true;

}

var regex = new Regex(parameter.RegexValidation);

var match = regex.IsMatch(parameter.Value);

return match;

}

public static bool ValidateParameter(Parameter[] parameters)

{

return parameters.All(ValidateParameter);

}

#### Methode: SaveSetting

Die Methode speichert ein Setting auf dem Filesystem ab, nachdem alle Parameter validiert wurden. Der Rückgabewert ist, ob es geklappt hat, oder nicht.

public static bool SaveSetting(ISetting setting, Parameter[] parameters)

{

var path = GetSettingPath(setting);

var jsonString = string.Empty;

if (parameters == null)

{

if (!ValidateParameter(GetParameterListFromSetting(setting))) return false;

jsonString = GetJsonStringOfSetting(setting);

}

if (parameters != null)

{

if (!ValidateParameter(parameters)) return false;

jsonString = Newtonsoft.Json.JsonConvert.SerializeObject(parameters);

}

try

{

System.IO.File.WriteAllText(path, jsonString);

return true;

}

catch

{

return false;

}

}

#### Methode: GetSetting

Diese Methode gibt ein ISetting mit seinen Werten zurück. So ist der Aufruf mit new ExampleSettingA() z.B. möglich, um den ExampleSettingA zu bekommen im ExampleServiceA.

public static ISetting GetSetting(ISetting setting)

{

var path = GetSettingPath(setting);

if (!System.IO.File.Exists(path))

{

InitialSaveSetting(setting);

}

var jsonString = System.IO.File.ReadAllText(path);

var paramList = Newtonsoft.Json.JsonConvert.DeserializeObject<Parameter[]>(jsonString);

var namePrefix = setting.GetType().Namespace;

foreach (var fieldInfo in setting.GetType().GetFields())

{

var value = paramList.First(p => p.Name == namePrefix + "." + fieldInfo.Name)?.Value;

if (value != null)

{

fieldInfo.SetValue(setting, Convert.ChangeType(value, fieldInfo.FieldType));

}

}

return setting;

}

#### Methode: InitialSaveParameter

Diese Methode Initialisiert einen Parameter, der noch nicht vorhanden ist, indem ein leereres Setting eines ISettingtyps im entsprechenden Service gespeichert wird.

private static void InitialSaveSetting(ISetting setting)

{

var path = GetSettingPath(setting);

var jsonString = GetJsonStringOfSetting(setting);

System.IO.File.WriteAllText(path, jsonString);

}

#### Methode: GetJsonStringOfSetting

Diese Methode wandelt ein Setting in einen JSON-String um.

private static string GetJsonStringOfSetting(ISetting setting)

{

var paramList = GetParameterListFromSetting(setting);

var jsonString = Newtonsoft.Json.JsonConvert.SerializeObject(paramList);

if (jsonString == null)

{

throw new NullReferenceException();

}

return jsonString;

}

#### Methode: GetSettingPath

Diese Methode gibt den Pfad eines Parameters zurück, um ihn so in die richtige Assembly zu lesen, respektive ab zu legen.

private static string GetSettingPath(ISetting s)

{

var fullPath = s.GetType().Assembly.CodeBase;

var path = fullPath.Replace(fullPath.Split('/').Last(), "parameters.json");

var uri = new UriBuilder(path);

return Uri.UnescapeDataString(uri.Path);

}

#### Methode: CreateParameter

Diese Methode erstellt einen Parameter aufgrund eines ISetting und dessen FieldInfo. Der Prefix-String ist für die Assemblyzuordnung im Frontend.

private static Parameter CreateParameter(FieldInfo fieldInfo, ISetting setting, string prefix)

{

var param = new Parameter

{

Name = prefix + "." + fieldInfo.Name,

Value = fieldInfo.GetValue(setting)?.ToString(),

Type = GetType(fieldInfo.FieldType)

};

if (param.Name == null || param.Type == null) return null;

var attributes = fieldInfo.GetCustomAttributes(true);

foreach (var attribute in attributes)

{

var mandatoryAttribute = attribute as MandatoryAttribute;

var defaultAttribute = attribute as DefaultAttribute;

var validationAttribute = attribute as ValidationAttribute;

var descriptionAttribute = attribute as DescriptionAttribute;

if (mandatoryAttribute != null)

{

param.Mandatory = true;

}

if (defaultAttribute != null)

{

param.Default = defaultAttribute?.Default;

}

if (validationAttribute != null)

{

param.RegexValidation = validationAttribute?.Regex;

}

if (descriptionAttribute != null)

{

param.Description = descriptionAttribute?.Description;

}

}

return param;

}

#### Methode: GetType

Diese Methode gibt den HTML Input-Typ für den Client zurück, damit man sich nicht darum kümmern muss, wie der Parameter im Client angezeigt wird.

private static string GetType(Type type)

{

if (type.Name == "Boolean")

{

return "checkbox";

}

if (type.Name == "Int32" || type.Name == "Double" || type.Name == "Float" || type.Name == "Int64" || type.Name == "Long")

{

return "number";

}

return "text";

}

### Klasse: ParameterBusHelper

Diese Klasse stellt Hilfsfunktionen für die RabbitMQ-Kommunikation zur Verfügung. Sie beinhaltet eine Funktion für das Subscriben am Get-Event und eine für das Subscriben am Save-Event.

#### Methode: SubscribeGetEvent

Stellt die Funktionalität zur Verfügung, was man im Get-Event machen muss, damit ein Parameter vom Parameter Service ausgelesen werden kann.

public static void SubscribeGetEvent<T>(IBus parameterBus) where T:ISetting

{

var paramInstance = Activator.CreateInstance(typeof(T)) as ISetting;

paramInstance = ParameterHelper.GetSetting(paramInstance);

parameterBus.Publish(new GetParameterEventResponse { Parameters = ParameterHelper.GetParameterListFromSetting(paramInstance) });

}

#### Methode: SubscribeSaveEvent

Stellt die Funktionalität zur Verfügung, was man im Save-Event machen muss, damit ein Parameter vom Parameter Service gespeichert werden kann.

public static void SubscribeSaveEvent<T>(IBus parameterBus, Parameter paramToSave) where T:ISetting

{

var paramInstance = Activator.CreateInstance(typeof(T)) as ISetting;

paramInstance = ParameterHelper.GetSetting(paramInstance);

var paramAsParamList = ParameterHelper.GetParameterListFromSetting(paramInstance);

if(paramAsParamList.Any(p => p.Name == paramToSave.Name))

{

paramAsParamList.First(p => p.Name == paramToSave.Name).Value = paramToSave.Value;

parameterBus.Publish(new SaveParameterEventResponse { Success = ParameterHelper.SaveSetting(paramInstance, paramAsParamList) });

}

}

### Klasse: Parameter

Diese Klasse zeigt, wie ein Parameter auszusehen hat.

#### Property: Name

Der Name des Parameters.

public string Name { get; set; }

#### Property: Type

Der Typ des Parameters.

public string Type { get; set; }

#### Property: Description

Ein Hilfetext des Parameters.

public string Description { get; set; }

#### Property: Value

Der parametrierte Wert des Parameters.

public string Value { get; set; }

#### Property: Default

Die Standardeinstellung des Parameters.

public string Default { get; set; }

#### Property: RegexValidation

Der Validierungs-String des Parameters.

public string RegexValidation { get; set; }

#### Property: Mandatory

Gibt zurück, ob der Parameter zwingend ausgefüllt sein muss, oder nicht.

public bool Mandatory { get; set; } = false;

### Interface: ISetting

Dieses Interface dient als gemeinsamer Nenner aller Settings. Es verfügt über keine Member.

### Klasse: BusConfigurator

Aus dem Projekt Viaduc kopiert. Diese Klasse erstellt den RabbitMQ Bus. Die Klasse enthält die ConfigureBus Methode, mit welcher man einen neuen RabbitMQ Bus erstellen kann.

### Ordner: Attributes

Dies ist ein Ordner er enthält sämtliche erstellten Attribute. Diese haben keine Logik, weshalb sie hier einfach als ganze Klassen aufgelistet werden.

#### Klasse: DefaultAttribute

Dieses Attribut ist für die Deklaration eines Standardwerts im Parameter.

[AttributeUsage(AttributeTargets.Field)]

public class DefaultAttribute : Attribute

{

public string Default { get; set; }

public DefaultAttribute(string value)

{

Default = value;

}

}

#### Klasse: DescriptionAttribute

Mit diesem Attribut kann ein Hinweistext dem Parameter mitgegeben werden.

[AttributeUsage(AttributeTargets.Field)]

public class DescriptionAttribute : Attribute

{

public string Description { get; set; }

public DescriptionAttribute(string description)

{

Description = description;

}

}

#### Klasse: MandatoryAttribute

Dieses Attribut zeigt an, wenn gesetzt, dass dies ein Pflichtfeld ist.

[AttributeUsage(AttributeTargets.Field)]

public class MandatoryAttribute : Attribute

{

}

#### Klasse: ValidationAttribute

Mit diesem Attribut kann eine Regular Expression dem Parameter zur Validierung gegeben werden.

[AttributeUsage(AttributeTargets.Field)]

public class ValidationAttribute : Attribute

{

public string Regex { get; set; }

public ValidationAttribute(string regex)

{

Regex = regex;

}

}

### Ordner: GetParameter

Dies ist ein Ordner mit allen Interfaces für die Kommunikation von RabbitMQ für den Get-Befehl. Die Klassen waren ohne Members bereits vorhanden. Sie dienen nur als Deklaration der Schnittstelle der einzelnen Microservices. Die Member werden jeweils mitgeschickt. Die Klassennamen sind sprechend für sich.

#### Klasse: GetParameterEvent

public class GetParameterEvent

{

}

#### Klasse: GetParameterEventResponse

public class GetParameterEventResponse

{

public Parameter[] Parameters { get; set; }

}

#### Klasse: GetParameterRequest

public class GetParameterRequest

{

}

#### Klasse: GetParameterResponse

public class GetParameterResponse

{

public Parameter[] Parameters { get; set; }

}

### Ordner: SaveParameter

Dies ist ein Ordner mit allen Interfaces für die Kommunikation von RabbitMQ für den Save-Befehl. Sie dienen nur als Deklaration der Schnittstelle der einzelnen Microservices. Die Member werden jeweils mitgeschickt. Die Klassennamen sind so gewählt, dass klar sein sollte, was sie tun.

#### Klasse: SaveParameterEvent

public class SaveParameterEvent

{

public Parameter Parameter { get; set; }

public SaveParameterEvent(Parameter parameter)

{

Parameter = parameter;

}

}

#### Klasse: SaveParameterEventResponse

public class SaveParameterEventResponse

{

public bool Success { get; set; } = false;

}

#### Klasse: SaveParameterRequest

public class SaveParameterRequest

{

public Parameter Parameter { get; set; }

public SaveParameterRequest(Parameter parameter)

{

Parameter = parameter;

}

}

#### Klasse: SaveParameterResponse

public class SaveParameterResponse

{

public bool Success { get; set; } = false;

}

## Assembly: CMI.Host.ExampleServiceA

Dieses Projekt dient dazu den Windowsdienst zu starten und zu beenden. Es beinhaltet keine Logik und ist in der Vorbereitung auf die IPA bereits vorhanden gewesen.

## Assembly: CMI.Host.ExampleServiceB

Dieses Projekt dient dazu den Windowsdienst zu starten und zu beenden. Es beinhaltet keine Logik und ist in der Vorbereitung auf die IPA bereits vorhanden gewesen.

## Assembly: CMI.Host.Parameter

Dieses Projekt dient dazu den Windowsdienst zu starten und zu beenden. Es beinhaltet keine Logik und ist in der Vorbereitung auf die IPA bereits vorhanden gewesen.

## Assembly: CMI.Manager.ExampleServiceA

### Klasse: ExampleServiceA

Diese Klasse Steuert das Startverhalten des Example Services A. In der Startfunktion werden sämtliche Endpoints für RabbitMQ initialisiert.

#### Property: ParameterBus

private IBusControl ParameterBus { get; set; }

#### Methode: Start

Diese Methode definiert das Startverhalten des Service.

public void Start()

{

ParameterBus = BusConfigurator.ConfigureBus((cfg, host) =>

{

cfg.ReceiveEndpoint(host, "GetAllParametersA", ep =>

{

ep.Handler<GetParameterEvent>(context =>

{

ParameterBusHelper.SubscribeGetEvent<ExampleSettingA>(ParameterBus);

return Console.Out.WriteLineAsync("Get Parameters");

});

});

cfg.ReceiveEndpoint(host, "SaveParametersA", ep =>

{

ep.Handler<SaveParameterEvent>(context =>

{

ParameterBusHelper.SubscribeSaveEvent<ExampleSettingA>(ParameterBus, context.Message.Parameter);

return Console.Out.WriteLineAsync("Saved Parameter");

});

});

});

ParameterBus.Start();

}

#### Methode: Stop

public void Stop()

{

}

### Klasse: ExampleSettingA

Diese Klasse erbt vom ISetting Interface damit man den Parameter Serialisieren kann. Dies ist ein Testparameter zur Demonstration und zum Testen.

#### Property: Date

[Mandatory]

[Default("12.02.2017")]

public string Date;

#### Property: EMailAdress

[Description("Diese Email dient nur zu Demonstrationszwecken.")]

[Validation(@"([a-zA-Z0-9\_.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.])")]

public string EMailAdress;

#### Property: ServiceOn

[Default("false")]

public bool ServiceOn;

NumberOfIterations

[Validation(@"[0-9]|[0-9][0-9]")]

[Mandatory]

public int NumberOfIterations;

## Assembly: CMI.Manager.ExampleServiceB

### Klasse: ExampleServiceB

Diese Klasse Steuert das Starverhalten des Example Services B. In der Startfunktion werden sämtliche Endpoints für RabbitMQ initalisiert.

#### Property: ParameterBus

Businstanz für die RabbitMQ Kommunikation.

private IBusControl ParameterBus { get; set; }

#### Methode: Start

Diese Methode definiert das Startverhalten des Service.

public void Start()

{

ParameterBus = BusConfigurator.ConfigureBus((cfg, host) =>

{

cfg.ReceiveEndpoint(host, "GetAllParametersB", ep =>

{

ep.Handler<GetParameterEvent>(context =>

{

ParameterBusHelper.SubscribeGetEvent<ExampleSettingB>(ParameterBus);

return Console.Out.WriteLineAsync("Get Parameters");

});

});

cfg.ReceiveEndpoint(host, "SaveParametersB", ep =>

{

ep.Handler<SaveParameterEvent>(context =>

{

ParameterBusHelper.SubscribeSaveEvent<ExampleSettingB>(ParameterBus, context.Message.Parameter);

return Console.Out.WriteLineAsync("Saved Parameter");

});

});

});

ParameterBus.Start();

}

#### Methode: Stop

public void Stop()

{

}

### Klasse: ExampleSettingB

Diese Klasse erbt vom ISetting Interface damit man den Parameter Serialisieren kann. Dies ist ein Testparameter zur Demonstration und zum Testen.

#### Property: EndeDatum

[Mandatory]

[Default("12.02.2017")]

[Validation(@"((0[1-9]|[12][0-9]|3[01])[- /.](0[1-9]|1[012])[-/.]\d\d\d\d)")]

public string EndeDatum;

#### Property: EMailAdress

[Description("Diese Email dient nur zu Demonstrationszwecken.")]

[Validation(@"([a-zA-Z0-9\_.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.])")]

public string EMailAdress;

#### Property: FehlerVerstecken

[Default("true")]

public bool FehlerVerstecken;

#### Property: Sekunden

[Validation(@"[0-9]|[0-5][0-9]")]

[Mandatory]

public int Sekunden;

## Assembly: CMI.Manager.Parameter

### Klasse: GetParameterEventResponseConsumer

Dies Klasse enthält die Logik, welche im Parameter Service ausgeführt wird, wenn eine Event-Antwort eines Get-Events im Parameter Service ankommt.

#### Methode: Consume

public async Task Consume(ConsumeContext<GetParameterEventResponse> context)

{

ParameterRequestResponseHelper.Parameters.AddRange(context.Message.Parameters);

await Console.Out.WriteLineAsync("Parameter Recived");

}

### Klasse: GetParameterRequestConsumer

Diese Klasse enthält die Logik, welche im Parameter Service ausgeführt wird, wenn ein GetAllParameter Request ausgelöst wird.

#### Methode: Consume

public Task Consume(ConsumeContext<GetParameterRequest> context)

{

ParameterRequestResponseHelper.Parameters = new List<Contract.Parameter.Parameter>();

ParameterService.ParameterBus.Publish(new GetParameterEvent());

Console.Out.WriteLineAsync("Get Event started");

Thread.Sleep(400);

if (ParameterRequestResponseHelper.Parameters.Count == 0)

{

Thread.Sleep(6000);

}

context.RespondAsync(new GetParameterResponse()

{

Parameters = ParameterRequestResponseHelper.Parameters.ToArray()

});

return Console.Out.WriteLineAsync("Get Event response sent");

}

}

### Klasse: ParameterRequestResponseHelper

Dies ist eine Helperklasse für die Eventkommunikation. In RabbitMQ gibt es keine Möglichkeit, eine Antwort eines Events für jeden Subscriber zu erhalten. Deshalb gibt einen Event vom zu parametrierenden Service aus einen Event zurück, welcher dann die Antworten dieser Services im Parameter Service speichern.

#### Property: Parameters

public static List<Contract.Parameter.Parameter> Parameters { get; set; } = new List<Contract.Parameter.Parameter>();

#### Property: SavedSuccessfully

public static bool? SavedSuccessfully { get; set; }

### Klasse: ParameterService

Diese Klasse Regelt das Startverhalten vom Parameterservice

#### Property: ParameterBus

Businstanz für die RabbitMQ Kommunikation.

public static IBusControl ParameterBus { get; set; }

#### Methode: Start

Diese Methode definiert das Startverhalten des Service.

public void Start()

{

ParameterBus = BusConfigurator.ConfigureBus((cfg, host) =>

{

cfg.ReceiveEndpoint("GetParameterQueue", ec =>

{

ec.Consumer(() => new GetParameterRequestConsumer());

});

cfg.ReceiveEndpoint("GetResponseParameterEventQueue", ec =>

{

ec.Consumer(() => new GetParameterEventResponseConsumer());

});

cfg.ReceiveEndpoint("SaveParameterQueue", ec =>

{

ec.Consumer(() => new SaveParameterRequestConsumer());

});

cfg.ReceiveEndpoint("SaveResponseParameterEventQueue", ec =>

{

ec.Consumer(() => new SaveParameterEventResponseConsumer());

});

});

ParameterBus.Start();

}

#### Methode: Stop

public void Stop()

{

ParameterBus.Stop();

}

### Klasse: SaveParameterEventResponseConsumer

Diese Klasse enthält die Logik, welche im Parameter Service ausgeführt wird, wenn eine Event-Antwort eines Save-Events im Parameter Service ankommt.

#### Methode: Consume

public async Task Consume(ConsumeContext<SaveParameterEventResponse> context)

{

ParameterRequestResponseHelper.SavedSuccessfully = context.Message.Success;

if (context.Message.Success)

{

await Console.Out.WriteLineAsync("Saved Successfully");

}

else

{

await Console.Out.WriteLineAsync("An Error has occured while saving!");

}

}

### Klasse: SaveParameterRequestConsumer

Dies ist eine Helperklasse für die Eventkommunikation. In RabbitMQ gibt es keine Möglichkeit, eine Antwort eines Events für jeden Subscriber zu erhalten. Deshalb gibt der Save-Event vom zu parametrierenden Service aus einen Event zurück, welcher dann die Antworten dieser Services im Parameter Service speichern.

#### Consume

public Task Consume(ConsumeContext<SaveParameterRequest> context)

{

ParameterRequestResponseHelper.SavedSuccessfully = null;

ParameterService.ParameterBus.Publish(new SaveParameterEvent(context.Message.Parameter));

Console.Out.WriteLineAsync("Save Event started");

Thread.Sleep(400);

if (ParameterRequestResponseHelper.SavedSuccessfully == null)

{

Thread.Sleep(6000);

}

context.RespondAsync(new SaveParameterResponse()

{

Success = ParameterRequestResponseHelper.SavedSuccessfully == true

});

return Console.Out.WriteLineAsync("Save Event response sent");

}

## Assembly: CMI.Web.Management

### Ordner: Controllers

#### Klasse: ParameterController

Diese Klasse enthält die API Controllers.

##### Methode: GetAllParameters

[Route(@"~/Controllers/GetAllParameters")]

[HttpGet]

public IHttpActionResult GetAllParameters()

{

var uri = new Uri(BusHelper.ParameterBus.Address, "GetParameterQueue");

var requestClient = BusHelper.ParameterBus.CreateRequestClient<GetParameterRequest, GetParameterResponse>(uri, TimeSpan.FromSeconds(20));

var result = requestClient.Request(new GetParameterRequest()).GetAwaiter().GetResult();

return Ok(result.Parameters);

}

##### Methode: SaveParameter

[Route(@"~/Controllers/SaveParameter")]

[HttpPost]

public IHttpActionResult SaveParameter(Parameter parameter)

{

var uri = new Uri(BusHelper.ParameterBus.Address, "SaveParameterQueue");

var requestClient = BusHelper.ParameterBus.CreateRequestClient<SaveParameterRequest, SaveParameterResponse>(uri, TimeSpan.FromSeconds(20));

var result = requestClient.Request(new SaveParameterRequest(parameter)).GetAwaiter().GetResult();

return Ok(result.Success);

}

### Ordner: Helpers

#### Klasse: BusHelper

Diese Klasse stellt die RabbitMQ Bus Instanz zur Kommunikation mit dem Parameterservice zur Verfügung.

##### Property: ParameterBus

Businstanz für die RabbitMQ Kommunikation.

public static IBusControl ParameterBus { get; }

##### Klasse: BusHelper

static BusHelper()

{

ParameterBus = BusConfigurator.ConfigureBus((cfg, host) =>

{

});

ParameterBus.Start();

}

# Code Dokumentation Frontend

## Ordner: services

Der Servicesordner beinhaltet sämtliche Services. Die Services werden via Angular Dependency Injection geladen.

### Klasse: HttpService

Dieser Service ist aus dem Viaduc Projekt übernommen. Der Service stellt die «get» und «post» Methode zur Verfügung.

### Klasse: ParameterService

Dieser Service ist für die API Aufrufe der Parameterverwaltung zuständig

#### Methode: getAllParameters

Diese Methode gibt alle Parameter zurück.

public async getAllParameters() {

let url = this.\_createBaseUrl() + '/GetAllParameters';

return await this.\_http.get<Parameter[]>(url, this.\_http.noCaching).toPromise();

}

#### Methode: saveParameter

Diese Methode speichert einen Parameter. Die Validierung geschieht in der ParameterComponent und im Backend.

public async saveParameter(param: Parameter) {

let url = this.\_createBaseUrl() + '/SaveParameter';

return await this.\_http.post<void>(url, param, this.\_http.noCaching).toPromise();

}

#### Methode: \_createBaseUrl

Diese Methode stellt die URL der Webseite zur Verfügung.

private \_createBaseUrl(): string {

let loc = window.location;

let port = isNaN(parseInt(loc.port, 10)) ? undefined : parseInt(loc.port, 10);

let baseUrl = '' + loc.protocol + '//' + loc.hostname + (port ? ':' + port : '') + '/ipa/Controllers';

return baseUrl;

}

## Ordner: highlight

### Klasse: HighlightComponent

Diese Component stellt die Funktionalität zur Verfügung, einen Text in einem anderen Text gelb hervorzuheben.

#### Property: highlight

Der Text, welcher hervorgehoben werden soll.

@Input()

public highlight: string;

#### Property: text

Der Text, der angezeigt werden soll.

@Input()

public text: string;

#### Methode: constructor

public constructor() {}

#### Methode: getInnerHTML

Diese Methode gibt den anzuzeigenden Text zurück.

public getInnerHTML(): string {

if (this.text && this.highlight) {

let position = this.text.toLowerCase().indexOf(this.highlight.toLowerCase());

if (position !== -1) {

let innerHTML: string = '',

replaceString = this.text.substr(position, this.highlight.length),

split = this.text.split(replaceString),

last = split.pop();

for (let s of split) {

innerHTML += s;

innerHTML += '<ins>' + replaceString + '</ins>';

}

innerHTML += last;

return innerHTML;

}

}

return this.text;

}

### HTML: HighlightComponent

<span [innerHTML]="getInnerHTML()">

</span>

## Ordner: parameterManager

### Klasse: Parameter

Diese Component ist für das Mapping des Parameter Typens mit dem des Backends.

#### Property: name

name: string;

#### Property: value

value: string;

#### Property: type

type: string;

#### Property: mandatory

mandatory: boolean;

#### Property: description

description: string;

#### Property: regexValidation

regexValidation: string;

#### Property: default

default: string;

## Ordner: parameter

### Klasse: ParameterComponent

Diese Component ist zuständig für die Anzeige, Speichern und das Validieren eines Parameters.

#### Property: parameter

Diese Property ist das Parameterobjekt.

@Input()

public parameter: Parameter;

#### Property: validationEvent

Dies ist der Event für die Validierungsfunktion via Button. Der Button befindet sich auf der ParameterListComponent.

@Input()

public validationEvent: EventEmitter<void> = new EventEmitter<void>();

#### Property: searchString

Dies ist der Suchtext vom Suchfeld.

@Input()

public searchString: string;

#### Property: active

Dieses Flag zeigt an, ob das Feld in Bearbeitung ist, oder nicht.

public active: boolean = false;

#### Property: value

Dies ist der Momentan geschriebene Wert im Parameterwertefeld. Es wird für die Abbrechen-Funktion benötigt.

public value: string;

#### Property: checked

Dieses Property ist das Value-Property für Boolean-Werte.

public checked: boolean;

#### Property: validationError

Dieses Flag steuert, ob der Parameter rot hinterlegt wird, oder nicht.

public validationError: boolean;

#### Property: \_onFocusChange

Dies ist der Event, welcher es ermöglicht, dass nur ein Parameter gleichzeitig in Bearbeitung sein kann.

private static \_onFocusChange: Subject<string> = new Subject();

#### Methode: constructor

constructor (private \_paramService: ParameterService) {

}

#### Methode: ngOnInit

Die ngOnInit Methode enthält die Logik, welche gleich nach dem Konstruktor folgt. Hier wird der Parameter befüllt und die Events Subscribed.

public ngOnInit() {

ParameterComponent.\_onFocusChange.subscribe((name) => {

if (name !== this.parameter.name) {

this.cancelEdit();

}

});

this.validationEvent.subscribe(() => {

this.validationError = !this.\_isValid();

});

this.value = this.parameter.value;

this.checked = this.parameter.value === 'True';

}

#### Methode: onValueChanged

Dies ist der Eventhandler des Value-Changed Event. Hier wird das Value- respektive Checked-Property gesetzt.

public onValueChanged(event: any) {

if (this.parameter.type === 'checkbox') {

this.checked = event.target.checked;

if (this.checked === (this.parameter.value === 'True')) {

this.active = false;

}

} else {

this.value = event.target.value;

}

}

#### Methode: onFocus

Dies ist der Eventhandler, der das Feld beim Fokus in Bearbeitung setzt. Die anderen Parameter werden zurückgesetzt.

public onFocus() {

ParameterComponent.\_onFocusChange.next(this.parameter.name);

this.active = true;

}

#### Methode: saveParameter

Diese Methode validiert und speichert den Parameter. Schlägt die Validierung Fehl, wird der Parameter rot hervorgehoben und der Parameter wird nicht gespeichert.

public saveParameter() {

this.validationError = !this.\_validateString(this.value);

if (this.validationError === false) {

if (this.parameter.type === 'checkbox') {

this.parameter.value = this.checked.toString();

} else {

this.parameter.value = this.value;

}

this.\_paramService.saveParameter(this.parameter).then( success => this.validationError = !success);

}

}

#### Methode: cancelEdit

Diese Methode bricht das Bearbeiten ab und setzt den Parameter wieder auf den Stand, der vom Server geholt wurde.

public cancelEdit() {

this.value = this.parameter.value;

if (this.parameter.type === 'checkbox') {

this.checked = this.parameter.value === 'True';

} else {

this.value = this.parameter.value;

}

this.active = false;

}

#### Methode: \_isValid

Diese Methode validiert den Parameter.

private \_isValid(): boolean {

return this.\_validateString(this.parameter.value);

}

#### Methode: \_validateString

Diese Methode validiert einen Text mit den Validierungen des Parameters.

private \_validateString(value: string): boolean {

if (!value && this.parameter.mandatory === true) {

return false;

}

if (this.parameter && this.parameter.regexValidation && value) {

let matches = value.match(this.parameter.regexValidation);

return (matches && matches[0] !== null);

} else {

return true;

}

}

#### Methode: getErrorClass

Diese Methode gibt die CSS Klasse zurück für den Parameter, ob der Parameter einen Fehler hat oder nicht.

public getErrorClass(): string {

return this.validationError ? 'parameter-list row alert-danger' : 'parameter-list row';

}

#### Methode: getInputClass

Diese Methode gibt die CSS Klasse für einen gefundenen Suchtreffer im Inputfeld zurück.

public getInputClass(): string {

if (this.value && this.searchString) {

if (this.value.toLowerCase().indexOf(this.searchString.toLowerCase()) !== -1) {

return 'form-control highlighted';

}

}

return 'form-control';

}

### HTML: ParameterComponent

<div \*ngIf="parameter" [class]="getErrorClass()" [title]="parameter.description ? parameter.description : ''">

<div class="col-md-3">

<cmi-viaduc-highlight [text]="parameter.name" [highlight]="searchString"></cmi-viaduc-highlight>

</div>

<var class="col-md-3">

**{{**parameter.default**}}**

</var>

<div class="col-md-4">

<input [class]="getInputClass()" [type]="parameter.type" (focus)="onFocus()" (change)="onValueChanged($event)" [checked]="checked" [value]="value">

</div>

<div \*ngIf="active" class="col-md-2">

<input type="button" class="btn" value="Speichern" (click)="saveParameter()">

<input type="button" class="btn" value="Abbrechen" (click)="cancelEdit()">

</div>

</div>

### Less: ParameterComponent

.parameter-list {

border-bottom: 1px solid #ddd;

border-top: 1px solid #ddd;

margin-left: 5px;

margin-right: 5px;

div, var {

.form-control[type='text'], .form-control[type='number'] {

display: initial;

}

.form-control[type='checkbox'] {

display: initial;

width: auto;

}

.highlighted {

background: #ff9;

color: #000;

text-decoration: none;

}

}

div[class\*=col-], var[class\*=col-]{

margin-top: 1em;

margin-bottom: 1em;

}

}

## Ordner: ParameterList

### Klasse: ParameterListComponent

Diese Component dient als Liste für die Parameter. Sie enthält das Suchenfeld und den Validierungsbutton.

#### Property: loading

Diese Property verhindert, dass die Page vor dem Eintreffen der Daten geladen wird und solange ein Ladebalken angezeigt wird.

public loading: boolean = true;

#### Property: filteredParameters

Diese Property enthält alle momentan angezeigten Parameter.

public filteredParameters: Parameter[] = [];

#### Property: \_allParameters

Diese Property enthält alle vom Server geladenen Parameter.

private \_allParameters: Parameter[] = [];

#### Property: validationEvent

Dies ist der Validierungsevent.

public validationEvent: EventEmitter<void> = new EventEmitter<void>();

#### Property: searchString

Dies ist das Inputbindig des Suchfelds.

public searchString: string = '';

#### Property: searchedStringUpToDate

Dieses Flag zeigt an, ob der Suchstring der aktuellste ist, oder ob er seit der letzten Suche verändert wurde.

public searchedStringUpToDate: boolean;

#### Methode: constructor

Im Konstruktor werden alle Parameter geladen.

constructor(private \_params: ParameterService) {

this.getAllParameters();

}

#### Methode: getAllParameters

Diese Methode holt alle Parameter vom Server und setzt die Liste aller Parameter und der angezeigten Parameter auf den Stand des Servers.

public async getAllParameters() {

this.\_params.getAllParameters().then(response => {

this.\_allParameters = response;

this.filteredParameters = this.\_allParameters;

this.loading = false;

});

}

#### Methode: onValueChanged

Dies ist der Event, welcher ausgelöst wird, wenn der Text im Suchfeld angepasst wurde. Der Eventhandler setzt das searchString- und searchStringUpToDate-Attribut.

public onValueChanged(event: any) {

this.searchString = event.target.value;

if (this.searchString) {

this.searchedStringUpToDate = false;

}

}

#### Methode: emitValidationEvent

Dies ist der Eventauslöser für die Validierung.

public emitValidationEvent() {

this.validationEvent.emit();

}

#### Methode: searchParam

Diese Methode filtert die Parameter nach der Eingabe im Suchfeld. Name und Value des Parameters warden überprüft.

public searchParam() {

this.filteredParameters = [];

this.searchedStringUpToDate = true;

if (this.searchString !== '') {

this.filteredParameters = this.\_allParameters.filter((param) =>

param.name.toLowerCase().indexOf(this.searchString.toLowerCase()) !== -1 || param.value && param.value.toLowerCase().indexOf(this.searchString.toLowerCase()) !== -1

);

} else {

this.filteredParameters = this.\_allParameters;

}

}

### HTML: ParameterListComponent

<div \*ngIf="!loading">

<h1>Zentralisierte Parameterverwaltung</h1>

<div [class]="getClass">

Der Parameter wurde erfolgreich gespeichert.

</div>

<div class="row">

<div class="col-md-4">

<input type="text" class="form-control" [value]="searchString" (change)="onValueChanged($event)"/>

</div>

<div class="col-md-1">

<input type="button" (click)="searchParam()" value="Suchen" class="btn"/>

</div>

<div class="col-md-6">

</div>

<div class="col-md-1">

<input type="button" (click)="emitValidationEvent()" value="Validieren" class="btn"/>

</div>

</div>

<div class="row">

<b class="col-md-3">Name</b>

<b class="col-md-3">Standardwert</b>

<b class="col-md-4">Wert</b>

</div>

<div \*ngFor="let param of filteredParameters">

<cmi-viaduc-parameter [searchString]="searchedStringUpToDate ? searchString : ''" [parameter]="param" [validationEvent]="validationEvent"></cmi-viaduc-parameter>

</div>

</div>

<div \*ngIf="loading">

<cmi-blocker class="cmi-visible cmi-fixed cmi-center cmi-shadow">

<cmi-spinner></cmi-spinner>

</cmi-blocker>

</div>

### Less: ParameterListComponent

.btn {

margin-bottom: 1em;

}

.no-display {

display: none;

}

.row {

margin: 0;

.col-md-1 {

padding: 0;

}

.col-md-4 {

padding-right: 0;

}

}

cmi-blocker {

cmi-spinner {

width: 50%;

height: 50%;

overflow: auto;

margin: auto;

position: fixed;

left: 0;

right: 0;

bottom: 0;

top: 40%;

}

}

# Code Dokumentation Unittests

## Klasse: TestSetting

Die Klasse erbt von ISetting. Sie dient zum Testen mit Unittests.

### Property: TestFlag

[Default("false")]

[Description("Some Test")]

public bool TestFlag;

### Property: TestMailAdress

[Default("max.muster@supermail.ch")]

[Mandatory]

[Validation(@"([a-zA-Z0-9\_.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.])")]

public string TestMailAdress;

### Property: TestDate

[Validation(@"((0[1-9]|[12][0-9]|3[01])[- /.](0[1-9]|1[012])[- /.]\d\d\d\d)")]

public string TestDate;

### Property: TestNumber

[Description("Testwert")]

public int TestNumber;

### Property: TestUnknownType

public char TestUnknownType;

### Property: TestMandatory

[Mandatory]

public string TestMandatory;

## ParameterSerializerTests

Diese Klasse testet das Serialisieren, Deserialisieren, Speichern und Lesen der Parameter.

### Methode: An\_empty\_setting\_can\_be\_serialized

[TestMethod]

public void An\_empty\_setting\_can\_be\_serialized()

{

var testSetting = new TestSetting();

var parameterList = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.AreEqual(parameterList.Length, 6);

}

### Methode: Name\_can\_be\_serialized\_correclty

[TestMethod]

public void Name\_can\_be\_serialized\_correclty()

{

var testSetting = new TestSetting();

var pl = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsTrue(pl.Any(p => p.Name == "CMI.Contract.Parameter.Tests.TestDate"));

Assert.IsTrue(pl.Any(p => p.Name == "CMI.Contract.Parameter.Tests.TestMailAdress"));

Assert.IsTrue(pl.Any(p => p.Name == "CMI.Contract.Parameter.Tests.TestNumber"));

Assert.IsTrue(pl.Any(p => p.Name == "CMI.Contract.Parameter.Tests.TestMandatory"));

Assert.IsTrue(pl.Any(p => p.Name == "CMI.Contract.Parameter.Tests.TestFlag"));

Assert.IsTrue(pl.Any(p => p.Name == "CMI.Contract.Parameter.Tests.TestUnknownType"));

}

### Methode: Value\_can\_be\_serialized\_correctly

[TestMethod]

public void Value\_can\_be\_serialized\_correctly()

{

var testSetting = new TestSetting

{

TestDate = "03.04.2018",

TestFlag = true,

TestMailAdress = "testmail@mail.ch",

TestNumber = 20,

TestMandatory = "Test"

};

var pl = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestDate") && p.Value == "03.04.2018"));

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestFlag") && p.Value == "True"));

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestMailAdress") && p.Value == "testmail@mail.ch"));

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestNumber") && p.Value == "20"));

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestMandatory") && p.Value == "Test"));

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestUnknownType") && p.Value == new char().ToString()));

}

### Methode: Default\_can\_be\_serialized\_correctly

[TestMethod]

public void Default\_can\_be\_serialized\_correctly()

{

var testSetting = new TestSetting();

var pl = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestFlag") && p.Default == "false"));

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestMailAdress") && p.Default == "max.muster@supermail.ch"));

Assert.AreEqual(pl.Count(p => p.Default != null), 2);

Assert.AreEqual(pl.Count(p => p.Default == null), 4);

}

### Methode: Type\_can\_be\_serialized\_correctly

[TestMethod]

public void Type\_can\_be\_serialized\_correctly()

{

var testSetting = new TestSetting();

var pl = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestFlag") && p.Type == "checkbox"));

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestMailAdress") && p.Type == "text"));

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestDate") && p.Type == "text"));

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestUnknownType") && p.Type == "text"));

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestNumber") && p.Type == "number"));

}

### Methode: Description\_can\_be\_serialized\_correctly

[TestMethod]

public void Description\_can\_be\_serialized\_correctly()

{

var testSetting = new TestSetting();

var pl = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestFlag") && p.Description == "Some Test"));

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestNumber") && p.Description == "Testwert"));

Assert.AreEqual(pl.Count(p => p.Description != null), 2);

Assert.AreEqual(pl.Count(p => p.Description == null), 4);

}

### Methode: Mandatory\_can\_be\_serialized\_correctly

[TestMethod]

public void Mandatory\_can\_be\_serialized\_correctly()

{

var testSetting = new TestSetting();

var pl = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestMailAdress") && p.Mandatory));

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestMandatory") && p.Mandatory));

Assert.AreEqual(pl.Count(p => p.Mandatory), 2);

Assert.AreEqual(pl.Count(p => !p.Mandatory), 4);

}

### Methode: Validation\_can\_be\_serialized\_correctly

[TestMethod]

public void Validation\_can\_be\_serialized\_correctly()

{

var testSetting = new TestSetting();

var pl = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestMailAdress") && p.RegexValidation == @"([a-zA-Z0-9\_.]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.])"));

Assert.IsTrue(pl.Any(p => p.Name.EndsWith("TestDate") && p.RegexValidation == @"((0[1-9]|[12][0-9]|3[01])[- /.](0[1-9]|1[012])[- /.]\d\d\d\d)"));

Assert.AreEqual(pl.Count(p => p.RegexValidation != null), 2);

Assert.AreEqual(pl.Count(p => p.RegexValidation == null), 4);

}

### Methode: GetSettingPath

private static string GetSettingPath(ISetting s)

{

var fullPath = s.GetType().Assembly.CodeBase;

var path = fullPath.Replace(fullPath.Split('/').Last(), "setting.json");

var uri = new UriBuilder(path);

return Uri.UnescapeDataString(uri.Path);

}

### Methode: Can\_be\_serialized\_saved\_and\_can\_then\_be\_deserialized\_and\_got

[TestMethod]

public void Can\_be\_serialized\_saved\_and\_can\_then\_be\_deserialized\_and\_got()

{

var testSetting = new TestSetting

{

TestDate = "03.04.2018",

TestFlag = true,

TestMailAdress = "testmail@mail.ch",

TestNumber = 20,

TestMandatory = "Test"

};

try

{

Assert.IsTrue(ParameterHelper.SaveSetting(testSetting, null));

var newTestSetting = (TestSetting) ParameterHelper.GetSetting(new TestSetting());

Assert.AreEqual(newTestSetting.TestDate, testSetting.TestDate);

Assert.AreEqual(newTestSetting.TestFlag, testSetting.TestFlag);

Assert.AreEqual(newTestSetting.TestMailAdress, testSetting.TestMailAdress);

Assert.AreEqual(newTestSetting.TestNumber, testSetting.TestNumber);

Assert.AreEqual(newTestSetting.TestMandatory, testSetting.TestMandatory);

Assert.AreEqual(newTestSetting.TestUnknownType, testSetting.TestUnknownType);

}

finally

{

var path = GetSettingPath(testSetting);

if (System.IO.File.Exists(path))

{

System.IO.File.Delete(path);

}

}

}

## Klasse: ParameterValidationTests

Diese Klasse testet die Parametervalidation.

### Methode: ParameterList\_Should\_be\_valid

[TestMethod]

public void ParameterList\_Should\_be\_valid()

{

var testSetting = new TestSetting

{

TestDate = "03.04.2018",

TestFlag = true,

TestMailAdress = "testmail@mail.ch",

TestNumber = 20,

TestMandatory = "Test"

};

var parameterList = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsTrue(ParameterHelper.ValidateParameter(parameterList));

}

### Methode: Empty\_value\_and\_mandatory\_should\_not\_be\_valid

[TestMethod]

public void Empty\_value\_and\_mandatory\_should\_not\_be\_valid()

{

var testSetting = new TestSetting();

var parameterList = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsFalse(ParameterHelper.ValidateParameter(parameterList.First(p => p.Name.EndsWith("TestMandatory"))));

}

### Methode: None\_empty\_value\_and\_mandatory\_should\_be\_valid

[TestMethod]

public void None\_empty\_value\_and\_mandatory\_should\_be\_valid()

{

var testSetting = new TestSetting();

testSetting.TestMandatory = "Test";

var parameterList = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsTrue(ParameterHelper.ValidateParameter(parameterList.First(p => p.Name.EndsWith("TestMandatory"))));

}

### Methode: None\_empty\_value\_that\_is\_not\_conform\_should\_not\_be\_valide

[TestMethod]

public void None\_empty\_value\_that\_is\_not\_conform\_should\_not\_be\_valide()

{

var testSetting = new TestSetting();

testSetting.TestDate = "This is no Date!";

var parameterList = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsFalse(ParameterHelper.ValidateParameter(parameterList.First(p => p.Name.EndsWith("TestDate"))));

}

### Methode: None\_empty\_value\_that\_is\_conform\_should\_be\_valide

[TestMethod]

public void None\_empty\_value\_that\_is\_conform\_should\_be\_valide()

{

var testSetting = new TestSetting();

testSetting.TestDate = "03.04.2018";

var parameterList = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsTrue(ParameterHelper.ValidateParameter(parameterList.First(p => p.Name.EndsWith("TestDate"))));

}

### Methode: Empty\_value\_that\_is\_mandatory\_and\_has\_a\_regex\_defined\_should\_not\_be\_valide

[TestMethod]

public void Empty\_value\_that\_is\_mandatory\_and\_has\_a\_regex\_defined\_should\_not\_be \_valide()

{

var testSetting = new TestSetting();

var parameterList = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsFalse(ParameterHelper.ValidateParameter(parameterList.First(p => p.Name.EndsWith("TestMailAdress"))));

}

### Methode: None\_empty\_conform\_value\_that\_is\_mandatory\_and\_has\_a\_regex\_defined\_should\_be\_valide

[TestMethod]

public void None\_empty\_conform\_value\_that\_is\_mandatory\_and\_has\_a\_regex\_defined \_should\_be\_valide()

{

var testSetting = new TestSetting();

testSetting.TestMailAdress = "max.muster@supermail.ch";

var parameterList = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsTrue(ParameterHelper.ValidateParameter(parameterList.First(p => p.Name.EndsWith("TestMailAdress"))));

}

### Methode: None\_empty\_none\_conform\_value\_that\_is\_mandatory\_and\_has\_a\_regex\_defined\_should\_not\_be\_valide

[TestMethod]

public void None\_empty\_none\_conform\_value\_that\_is\_mandatory\_and\_has\_a\_regex \_defined\_should\_not\_be\_valide()

{

var testSetting = new TestSetting();

testSetting.TestMailAdress = "This is not a mail adress!";

var parameterList = ParameterHelper.GetParameterListFromSetting(testSetting);

Assert.IsFalse(ParameterHelper.ValidateParameter(parameterList.First(p => p.Name.EndsWith("TestMailAdress"))));

}

# Verzeichnisse

## Glossar

|  |  |
| --- | --- |
| Begriff | Erklärung |
| Git / Github | Git ist ein Versionsverwaltungssystem und Github ist ein Cloud-Speicherort dafür. |
| Repository | Repository ist ein Github Projekt |
| Visual Studio | Programmierumgebung für C# in diesem Projekt |
| Webstorm | Programmierumgebung für TypeScript, HTML und Less in diesem Projekt |
| Backend | Serverteil des Programms, ist für die Datenverarbeitung im Hintergrund verantwortlich |
| RabbitMQ | Message Queue Tool zur Kommunikation der Microservices |
| Message Queue | Ein Stapel von Nachrichten von einem Service an den anderen. |
| MassTransit | C#.Net Anbindung für RabbitMQ |
| Contributer | Englisch der Beiträger. Im Kontext von Git / Github ein Entwickler |
| Branch | Einen Stand des Source Codes, der unabhängig vom restlichen Entwicklungsstand ist zur Implementierung eines Features ohne Einfluss von aussen. |
| Mergen | Zusammenführen von 2 Branchen |
| Commit | Änderungen ins Git übernehmen. |
| Camel-Case | Eine Möglichkeit etwas zu benennen. DieserSatzStehtInCamelCase |
| Lower-Camel-Case | Gleich wie Camel-Case nur kleiner Buchstabe am Anfang |
| Dependency Injection | Methode um ein die Abhängigkeit eines Objekts zu übergeben. So kann ein Service z.B. von allen genutzt werden, obwohl es nur eine Instanz gibt. |
| Angular | Clientframework für «Single Page Applikationen». |
| Bootstrap | CSS Framework |
| Scrum | Agile Projektmethode |
| API | Serverseitige Funktion. |
| Queue | Englisch für «Reihe». Eine Kommunikation-Queues im Kontext dieser Arbeit ist ein «Stapel» voller Anweisungen von RabbitMQ. |
| Frontend | Benutzeroberfläche und alles, was dafür berechnet werden muss. Interagiert mit dem Backend |
| Framework | Eine Programmierhilfe mit gewisser Funktionalität, sodass das Grundgerüst nicht selbst geschrieben werden muss. |
| E2E Tests | End-to-End Tests, womit das Frontend getestet werden kann. |
| Regex | Abkürzung für Regular Expression, ist eine Muster um etwas in einem Text zu finden. |
| officeatwork | Tool für die Vorlagenverwaltung |
| Paging | Ein Software design Pattern, in welchem man eine Liste von Daten in mehreren Schritten lädt. Z.B. bei einer Google-Suche kriegt man die Ergebnisse «gepaged» zurück. Sprich auf mehreren Seiten. |
| Mouseover | GUI Event beim drüberfahren der Maus über das Objekt. |
| GUI | Grafical User Interface, die Benutzerüberfläche |
| OnFocus | GUI Event beim Anklicken / Anspringen mit Tab eines Inputelements. |
| Overload | Gleiche Methode mit verschiedenen Übergabewerten |
| Prefix | Ein Voranstehender Text. |
| Flag | Ein Ja/Nein Wert |
| Inputbindig | Das binden einer Variable an ein Inputfeld |
| Sneak Case | Dieser\_satz\_ist\_in\_snake\_case\_geschrieben |

Tabelle 35 - Glossar

## Tabellenverzeichnis

[Tabelle 1 - Dokumentablage 13](#_Toc510514865)

[Tabelle 2 - Konventionen Dokumentation 14](#_Toc510514866)

[Tabelle 3 - Code-Konventionen C# 14](#_Toc510514867)

[Tabelle 4 - Code Konventionen TypeScript 14](#_Toc510514868)

[Tabelle 5 - Layout-Konventionen 15](#_Toc510514869)

[Tabelle 6 - Systemgrenze Erklärungen Diagramm 17](#_Toc510514870)

[Tabelle 7 - Getting Started Dokumentation Ziele 19](#_Toc510514871)

[Tabelle 8 - Getting Started Dokumentation Anforderungen 19](#_Toc510514872)

[Tabelle 9 - Anzeige & Speichern der Parameter Ziele 19](#_Toc510514873)

[Tabelle 10 - Anzeige & Speichern der Parameter Anforderungen 19](#_Toc510514874)

[Tabelle 11 - Implementation des Validierungsmechanismus Ziele 20](#_Toc510514875)

[Tabelle 12 - Implementation des Validierungsmechanismus Anforderungen 20](#_Toc510514876)

[Tabelle 13 - Suchen eines Parameters Ziele 20](#_Toc510514877)

[Tabelle 14 - Suchen eines Parameters Anforderungen 20](#_Toc510514878)

[Tabelle 15 - Erstellung der Unit Tests und des Testkonzepts Anforderungen 20](#_Toc510514879)

[Tabelle 16 - Durchführen der Tests Ziele 21](#_Toc510514880)

[Tabelle 17 - Durchführen der Tests Anforderungen 21](#_Toc510514881)

[Tabelle 18 - Umsetzungsreihenfolge 23](#_Toc510514882)

[Tabelle 19 - Lösungsmatrix Unittests 23](#_Toc510514883)

[Tabelle 20 - Lösungsmatrix Integration Tests / e2e Tests 24](#_Toc510514884)

[Tabelle 21 - Lösungsmatrix User-Testing 24](#_Toc510514885)

[Tabelle 22 - Lösungsmatrix Speichern der Parameter als Parametertyp im Json 25](#_Toc510514886)

[Tabelle 23 - Lösungsmatrix Speichern der Parameter als generischer Typ im Json 25](#_Toc510514887)

[Tabelle 24 - Lösungsmatrix nur Serverseitig im Service den Parameter selbst testen lassen 30](#_Toc510514888)

[Tabelle 25 - Lösungsmatrix Regular Expression im Parameter 30](#_Toc510514889)

[Tabelle 26 - Lösungsmatrix Volltextsuche über alles 31](#_Toc510514890)

[Tabelle 27 - Lösungsmatrix Suche auf die Parameternamen 31](#_Toc510514891)

[Tabelle 28 - Lösungsmatrix Suche auf Parameternamen und Parameterwert 32](#_Toc510514892)

[Tabelle 29 - Getting Started Dokumentation Anforderungen 33](#_Toc510514893)

[Tabelle 30 - Testfälle 34](#_Toc510514894)

[Tabelle 31 - Resultate Testdurchgang 1 38](#_Toc510514895)

[Tabelle 322 - Resultate Testdurchgang 2 39](#_Toc510514896)

[Tabelle 33 - Glossar 81](#_Toc510514897)

## Bildverzeichnis

[Abbildung 1 - Struktur der Dokumentablage](#_Toc510196267)

[Abbildung 2 - IPERKA](#_Toc510196268)

[Abbildung 3 - Umsysteme](#_Toc510196269)

[Abbildung 4 - Systemgrenzen](#_Toc510196270)

[Abbildung 5 - RabbitMQ Event Kommunikation](#_Toc510196271)

[Abbildung 6 - Firefox/Waterfox Einstellungen](#_Toc510196272)

[Abbildung 7 - Chrome Einstellungen](#_Toc510196273)

[Abbildung 8 - Visual Studio Einstellungen](#_Toc510196274)

[Abbildung 9 - Mockup](#_Toc510196275)

[Abbildung 10 - Screenshot umgesetzte Lösung](#_Toc510196276)

[Abbildung 11 - Sequenzdiagramm Servicekommunikation beim «Save»](#_Toc510196277)

## Quellenverzeichnis

|  |  |
| --- | --- |
| Quelle | Information |
| http://masstransit-project.com/ | Wie man RabbitMQ anspricht / Buskonfiguration |
| Projekt Viaduc Code | Grundgerüst der IPA |
| https://stackoverflow.com/ | Syntaxfragen von TypeScript und C# |
| https://angular.io/ | TypeScript Funktionalitätsfragen von TypeScript |
| https://msdn.microsoft.com/en-us/ | C# Reflection Syntaxfragen |