

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 Glt.
1.0	06.04.2018	Remo Kessler	Abgeschlossen	Fertige IPA Dokumentation

Referenzierte Dokumente

Dokumentenname	Version	Autor	Datum

Verteiler & Projektorganisation

Name / Vorname	Kurzzeichen	Organisation	Rolle
Remo Kessler	KRE	CM Informatik AG	Lehrling Informatik
Martin Tinner	TMA	CM Informatik AG	IPA Betreuer
Matthias Hess	MHE	CM Informatik AG	IPA Fachbetreuer
Patrick Schättin	-	Noser Engineering	IPA Hauptexperte
Reto Loser	-	Swisscom	IPA Nebenexperte

Konventionen

In diesem Dokument wurden die folgenden Konventionen verwendet:

Was	Beschrieb
Gleichstellung	Im Zuge sprachlicher Vereinfachung wird innerhalb des vorliegenden Dokuments jeweils nur eine Form von Personenbezeichnungen (z.B. Projektleiterin, Mitarbeiter etc.) verwendet. Es ist stets auch die andere Form der entsprechenden Personenbezeichnung gemeint und miteingeschlossen.

Inhaltsverzeichnis

1 Zweck des Dokuments	12
1.1 Thema und Zielsetzung	12
1.2 Allgemeines zur Umgebung	12
1.3 Endprodukt	12
2 Rahmenbedingungen	13
2.1 Titel der IPA	13
2.2 Auslöser der Aufgabenstellung	13
2.3 Grobanforderungen	13
2.3.1 Parameterverwaltung	13
2.3.1.1 Funktionale Anforderungen	13
2.3.1.2 Nicht Funktionale Anforderungen	13
2.3.2 «Getting Started» Dokumentation	14
2.4 Vorkenntnisse	14
2.5 Neue Lerninhalte	14
2.6 Mittel & Methoden	14
2.6.1 Eingesetzte Mittel	14
2.6.2 Git Workflow	14
2.6.3 Daily Business	14
2.6.4 Dokumentablage	15
2.6.5 Kontroll-Tasks	17
3 Allgemeines	18
3.1 Konventionen	18
3.1.1 Dokumentation	18
3.1.2 Code-Konventionen C#	18
3.1.3 Code-Konventionen TypeScript	18
3.1.4 Layout-Konventionen	19
3.1.5 Unittests-Konventionen	19
3.2 Vorgehensmodell	19
3.2.1 IPERKA	19
4 Umsysteme, Abhängigkeiten, Systemgrenzen und Schnittstellen	20
4.1 Umsysteme	20
4.2 Abhängigkeiten	20
4.3 Systemgrenzen	21

4.4	Schnittstellen	22
5	Tagesjournale	23
5.1	Tagesjournal vom 19.03.2018.....	23
5.2	Tagesjournal vom 20.03.2018.....	24
5.3	Tagesjournal vom 22.03.2018.....	25
5.4	Tagesjournal vom 23.03.2018.....	26
5.5	Tagesjournal vom 26.03.2018.....	27
5.6	Tagesjournal vom 27.03.2018.....	28
5.7	Tagesjournal vom 29.03.2018.....	29
5.8	Tagesjournal vom 03.04.2018.....	30
5.9	Tagesjournal vom 05.04.2018.....	31
5.10	Tagesjournal vom 06.05.2018.....	32
5.11	Allgemeines zum Tagesjournal	33
6	Taskerstellung nach den Anforderungen	34
6.1	Einzeltasks für die Dokumentation	35
6.1.1	Getting Started Dokumentation	35
6.2	Einzeltasks für die Realisierung	35
6.2.1	Anzeige & Speichern der Parameter	35
6.2.2	Implementation des Validierungsmechanismus	36
6.2.3	Suchen eines Parameters	36
6.2.4	Erstellung der Unit Tests und des Testkonzepts	36
6.3	Einzeltaskerstellung der Kontrolltasks.....	37
6.3.1	Durchführen der Tests	37
7	Zeitplan & Burn down.....	38
7.1	Zeitplan.....	38
7.2	Burn down	40
8	Entscheide der einzelnen Tasks	41
8.1	Umsetzungsreihenfolge.....	41
8.2	Erstellung des Testkonzepts	41
8.2.1	Anforderung	41
8.2.2	Mögliche Lösungen.....	41
8.2.2.1	Unittests.....	41
8.2.2.2	Integration Tests / e2e Tests	42
8.2.2.3	User-Testing	42
8.2.3	Umsetzungsbeschreibung	42

8.3	Anzeige & Speichern der Parameter	43
8.3.1	Anforderung	43
8.3.2	Mögliche Lösungen Speichern	43
8.3.2.1	Speichern der Parameter als Parametertyp im Json	43
8.3.2.2	Speichern der Parameter als generischer Typ im Json	43
8.3.3	Umsetzungsbeschreibung Speichern	44
8.3.4	Mögliche Lösung Anzeige.....	44
8.3.4.1	Ähnlich wie in Firefox / Waterfox die about:config	44
8.3.4.2	Ähnlich wie die Chrome Settings	45
8.3.4.3	Ähnlich wie in Visual Studio	46
8.3.5	Umsetzungsbeschreibung Anzeige	47
8.4	Implementation des Validierungsmechanismus	48
8.4.1	Anforderung	48
8.4.2	Mögliche Lösungen.....	48
8.4.2.1	Nur serverseitig im Service den Parameter selbst testen lassen	48
8.4.2.2	Regular Expression im Parameter	48
8.4.3	Umsetzungsbeschreibung	48
8.5	Suche eines Parameters	49
8.5.1	Anforderung	49
8.5.2	Mögliche Lösungen.....	49
8.5.2.1	Volltextsuche über alles.....	49
8.5.2.2	Suche auf die Parameternamen	49
8.5.2.3	Suche auf Parameternamen und Parameterwert	50
8.5.3	Umsetzungsbeschreibung	50
8.6	Erstellung der Unittests	50
8.6.1	Anforderung	50
8.6.2	Umsetzungsbeschreibung	50
8.7	Durchführen der Tests.....	50
8.7.1	Anforderung	50
8.7.2	Umsetzungsbeschreibung	50
8.8	Getting Started Dokumentation	51
8.8.1	Anforderung	51
8.8.2	Umsetzungsbeschreibung	51
9	Realisierung	52
9.1	Erstellung des Testkonzepts	52
9.2	Anzeige & Speichern der Parameter	53
9.2.1	Frontend	53

9.2.2 Backend.....	54
9.2.3 Technische Umsetzung Backend	54
9.2.4 Erstellte und Bearbeitete Klassen.....	55
9.3 Implementation des Validierungsmechanismus	56
9.3.1 Erstellte und bearbeitete Klassen	56
9.4 Suche eines Parameters	57
9.4.1 Erstellte und bearbeitete Klassen	57
9.5 Schreiben der Unittests	58
9.5.1 ParameterSerializerTests	58
9.5.2 ParameterValidationTests	59
9.5.3 Erstellte und bearbeitete Klassen	59
10Getting Started mit der Parameterverwaltung	60
10.1 Einen eigenen Parameter schreiben	60
10.1.1 Ein Setting existiert und nur ein Parameter soll hinzugefügt werden	60
10.1.2 Kein Setting existiert	60
10.2 Einen Parameter/ein Setting auslesen	61
10.3 Beispielsparameter.....	61
10.3.1 Unterstützte Parametertypen	61
10.3.2 Unterstützte Attributtypen	61
10.4 Resultat:	62
11Testing.....	63
11.1 Testdurchgang nach Task Anzeige & Speichern der Parameter	63
11.2 Testdurchgang nach Task Implementation des Validierungsmechanismus	64
11.3 Testdurchgang nach Task Suche eines Parameters	65
11.4 Testdurchgang nach Task Erstellung der Unittests.....	65
12Code Dokumentation Backend.....	66
12.1 Assembly: CMI.Contract.Parameter	66
12.1.1 Klasse: ParameterHelper.....	66
12.1.1.1 Methode: GetParameterListFromSetting	66
12.1.1.2 Methode: ValidateParameter	67
12.1.1.3 Methode: SaveSetting	68
12.1.1.4 Methode: GetSetting	69
12.1.1.5 Methode: InitialSaveParameter	69
12.1.1.6 Methode: GetJsonStringOfSetting.....	70
12.1.1.7 Methode: GetSettingPath	70
12.1.1.8 Methode: CreateParameter	71

12.1.1.9	Methode: GetType.....	72
12.1.2	Klasse: ParameterBusHelper	72
12.1.2.1	Methode: SubscribeGetEvent.....	72
12.1.2.2	Methode: SubscribeSaveEvent	73
12.1.3	Klasse: Parameter	73
12.1.3.1	Property: Name	73
12.1.3.2	Property: Type.....	73
12.1.3.3	Property: Description.....	74
12.1.3.4	Property: Value.....	74
12.1.3.5	Property: Default	74
12.1.3.6	Property: RegexValidation.....	74
12.1.3.7	Property: Mandatory.....	74
12.1.4	Interface: ISetting.....	74
12.1.5	Klasse: BusConfigurator	74
12.1.6	Ordner: Attributes	75
12.1.6.1	Klasse: DefaultAttribute	75
12.1.6.2	Klasse: DescriptionAttribute	75
12.1.6.3	Klasse: MandatoryAttribute	75
12.1.6.4	Klasse: ValidationAttribute	76
12.1.7	Ordner: GetParameter	76
12.1.7.1	Klasse: GetParameterEvent	76
12.1.7.2	Klasse: GetParameterEventResponse.....	76
12.1.7.3	Klasse: GetParameterRequest.....	76
12.1.7.4	Klasse: GetParameterResponse	77
12.1.8	Ordner: SaveParameter.....	77
12.1.8.1	Klasse: SaveParameterEvent.....	77
12.1.8.2	Klasse: SaveParameterEventResponse	77
12.1.8.3	Klasse: SaveParameterRequest	78
12.1.8.4	Klasse: SaveParameterResponse.....	78
12.2	Assembly: CMI.Host.ExampleServiceA	78
12.3	Assembly: CMI.Host.ExampleServiceB	78
12.4	Assembly: CMI.Host.Parameter	78
12.5	Assembly: CMI.Manager.ExampleServiceA.....	79
12.5.1	Klasse: ExampleServiceA.....	79
12.5.1.1	Property: ParameterBus	79
12.5.1.2	Methode: Start.....	79
12.5.1.3	Methode: Stop	80
12.5.2	Klasse: ExampleSettingA	80

12.5.2.1	Property: Date	80
12.5.2.2	Property: EMailAdress.....	80
12.5.2.3	Property: ServiceOn	80
12.6	Assembly: CMI.Manager.ExampleServiceB.....	81
12.6.1	Klasse: ExampleServiceB.....	81
12.6.1.1	Property: ParameterBus	81
12.6.1.2	Methode: Start.....	81
12.6.1.3	Methode: Stop	82
12.6.2	Klasse: ExampleSettingB	82
12.6.2.1	Property: EndeDatum	82
12.6.2.2	Property: EMailAdress.....	82
12.6.2.3	Property: FehlerVerstecken.....	82
12.6.2.4	Property: Sekunden.....	82
12.7	Assembly: CMI.Manager.Parameter	83
12.7.1	Klasse: GetParameterEventResponseConsumer	83
12.7.1.1	Methode: Consume	83
12.7.2	Klasse: GetParameterRequestConsumer	83
12.7.2.1	Methode: Consume	83
12.7.3	Klasse: ParameterRequestResponseHelper	84
12.7.3.1	Property: Parameters	84
12.7.3.2	Property: SavedSuccessfully.....	84
12.7.4	Klasse: ParameterService	84
12.7.4.1	Property: ParameterBus	84
12.7.4.2	Methode: Start.....	85
12.7.4.3	Methode: Stop	85
12.7.5	Klasse: SaveParameterEventResponseConsumer	86
12.7.5.1	Methode: Consume	86
12.7.6	Klasse: SaveParameterRequestConsumer	87
12.7.6.1	Consume	87
12.8	Assembly: CMI.Web.Management.....	88
12.8.1	Ordner: Controllers	88
12.8.1.1	Klasse: ParameterController	88
12.8.1.1.1	Methode: GetAllParameters.....	88
12.8.1.1.2	Methode: SaveParameter	88
12.8.2	Ordner: Helpers	89
12.8.2.1	Klasse: BusHelper.....	89
12.8.2.1.1	Property: ParameterBus.....	89
12.8.2.1.2	Klasse: BusHelper	89

12.9 Assembly: CMI.Contract.Parameter.Tests	89
12.9.1 Klasse: TestSetting	89
12.9.1.1 Property: TestFlag	89
12.9.1.1.1 Property: TestMailAdress	89
12.9.1.2 Property: TestDate	90
12.9.1.3 Property: TestNumber	90
12.9.1.4 Property: TestUnknownType	90
12.9.1.5 Property: TestMandatory	90
12.9.2 ParameterSerializerTests	90
12.9.2.1 Methode: An_empty_setting_can_be_serialized	90
12.9.2.2 Methode: Name_can_be_serialized_correctly	91
12.9.2.3 Methode: Type_can_be_serialized_correctly	91
12.9.2.4 Methode: Default_can_be_serialized_correctly	92
12.9.2.5 Methode: Value_can_be_serialized_correctly	92
12.9.2.6 Methode: Description_can_be_serialized_correctly	93
12.9.2.7 Methode: Mandatory_can_be_serialized_correctly	93
12.9.2.8 Methode: GetSettingPath	93
12.9.2.9 Methode: Validation_can_be_serialized_correctly	94
12.9.2.10 Methode: Can_be_serialized_saved_and_can_then_be_deserialized_and_got	95
12.9.3 Klasse: ParameterValidationTests	96
12.9.3.1 Methode: ParameterList_Should_be_valid	96
12.9.3.2 Methode: Empty_value_and_mandatory_should_not_be_valid	96
12.9.3.3 Methode: None_empty_value_and_mandatory_should_be_valid	97
12.9.3.4 Methode: None_empty_value_that_is_not_conform_should_not_be_valide	97
12.9.3.5 Methode: None_empty_value_that_is_conform_should_be_valide	97
12.9.3.6 Methode: Empty_mandatory_value_with_regex_defined_should_not_be_valide	98
12.9.3.7 Methode: None_empty_mandatory_conform_value_with_regex_defined_is_valide	98
12.9.3.8 Methode: None_empty_unconform_mandatory_value_with_regex_defined_is_not_valide ...	98
13 Code Dokumentation Frontend	99
13.1 Ordner: services	99
13.1.1 Klasse: HttpService	99
13.1.2 Klasse: ParameterService	99
13.1.2.1 Methode: getAllParameters	99
13.1.2.2 Methode: saveParameter	99
13.1.2.3 Methode: _createUrl	100
13.2 Ordner: highlight	100
13.2.1 Klasse: HighlightComponent	100

13.2.1.1	Property: highlight	100
13.2.1.2	Property: text	100
13.2.1.3	Methode: constructor.....	100
13.2.1.4	Methode: getInnerHTML	101
13.2.2	HTML: HighlightComponent	101
13.3	Ordner: parameterManager.....	101
13.3.1	Klasse: Parameter	101
13.3.1.1	Property: name.....	101
13.3.1.2	Property: value	102
13.3.1.3	Property: type	102
13.3.1.4	Property: mandatory	102
13.3.1.5	Property: description	102
13.3.1.6	Property: regexValidation	102
13.3.1.7	Property: default	102
13.4	Ordner: parameter.....	103
13.4.1	Klasse: ParameterComponent.....	103
13.4.1.1	Property: parameter	103
13.4.1.2	Property: validationEvent	103
13.4.1.3	Property: searchString.....	103
13.4.1.4	Property: active	103
13.4.1.5	Property: value	103
13.4.1.6	Property: checked	104
13.4.1.7	Property: validationError.....	104
13.4.1.8	Property: _onFocusChange.....	104
13.4.1.9	Methode: constructor.....	104
13.4.1.10	Methode: ngOnInit.....	105
13.4.1.11	Methode: onValueChanged.....	105
13.4.1.12	Methode: onFocus.....	106
13.4.1.13	Methode: saveParameter	106
13.4.1.14	Methode: cancelEdit.....	106
13.4.1.15	Methode: _isValid.....	107
13.4.1.16	Methode: _validateString.....	107
13.4.1.17	Methode: getErrorClass	108
13.4.1.18	Methode: getInputClass	108
13.4.2	HTML: ParameterComponent.....	109
13.4.3	Less: ParameterComponent	110
13.5	Ordner: ParameterList.....	111
13.5.1	Klasse: ParameterListComponent	111

13.5.1.1	Property: loading	111
13.5.1.2	Property: filteredParameters.....	111
13.5.1.3	Property: _allParameters.....	111
13.5.1.4	Property: validationEvent	111
13.5.1.5	Property: searchString.....	111
13.5.1.6	Property: searchedStringUpToDate	112
13.5.1.7	Methode: constructor.....	112
13.5.1.8	Methode: getAllParameters.....	112
13.5.1.9	Methode: onValueChanged.....	112
13.5.1.10	Methode: emitValidationEvent.....	113
13.5.1.11	Methode: searchParam	113
13.5.2	HTML: ParameterListComponent	114
13.5.3	Less: ParameterListComponent	115
14	Schlusswort	116
14.1	Ausblick auf nach der Arbeit.....	116
14.2	Reflexion	116
14.3	Danksagung	116
15	Verzeichnisse.....	117
15.1	Glossar	117
15.2	Tabellenverzeichnis.....	119
15.3	Bildverzeichnis	121
15.4	Quellenverzeichnis	122

1 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.

1.1 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.

1.2 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.

1.3 Endprodukt

Nach Abschluss der Projektarbeit liegen folgende Produkte vor:

- Vollständige Dokumentation des Lösungsweges
- Arbeitsjournale
- Programm als Visual Studio respektive Webstorm Projekte

2 Rahmenbedingungen

2.1 Titel der IPA

Zentralisierte Parameterverwaltung für eine Mikroservices-Architektur

2.2 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 umzusetzen. 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 Mikroservices-Architektur wäre, dass jeder Service unabhängig von einem anderen laufen kann. Wenn dies der Fall ist, hat man einen der grössten Vorteile der Mikroservices-Architektur.

2.3 Grobanforderungen

2.3.1 Parameterverwaltung

Dies sind die Kriterien aus der Aufgabenstellung für die Parameterverwaltung.

2.3.1.1 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

2.3.1.2 Nicht Funktionale Anforderungen

- Die Parameter werden im Service gespeichert, nicht zentral(!)
- Die Services funktionieren auch, wenn die Parameterverwaltung nicht läuft
- Die PV funktioniert rechnerübergreifend, ohne File-Sharing
- Es werden sprechende Namen für Klassen / Methoden verwendet
- Die Einbindung der Parameterverwaltung soll mit der Angabe des Parametertyps und einem Minimum von weiterem Code möglich sein.
- 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

2.3.2 «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

2.4 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

2.5 Neue Lerninhalte

- Microservices-Architektur
- RabbitMQ / MassTransit

2.6 Mittel & Methoden

2.6.1 Eingesetzte Mittel

- Visual Studio für die Backendprogrammierung in C#
- Webstorm für die Frontendprogrammierung in TypeScript
- Microsoft Office für die Erstellung der Dokumentation
- 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

2.6.2 Git Workflow

Das gesamte Projekt wird mit Git respektive Github versioniert und abgelegt. Da es im Projekt nur einen «Contributor» 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.

2.6.3 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.

2.6.4 Dokumentablage

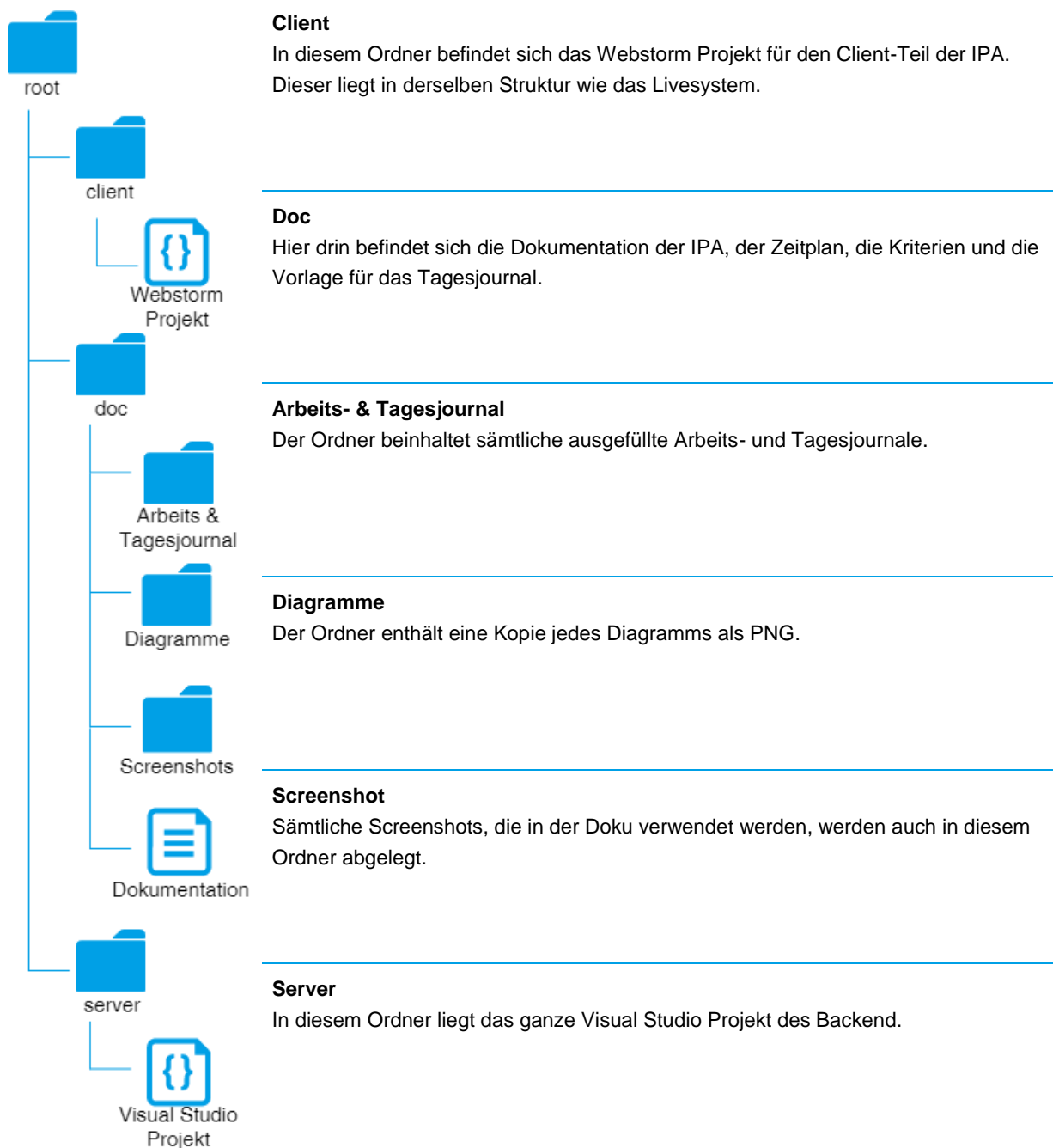


Abbildung 1 - Struktur der Dokumentablage

Tabelle 1 – Dokumentablage

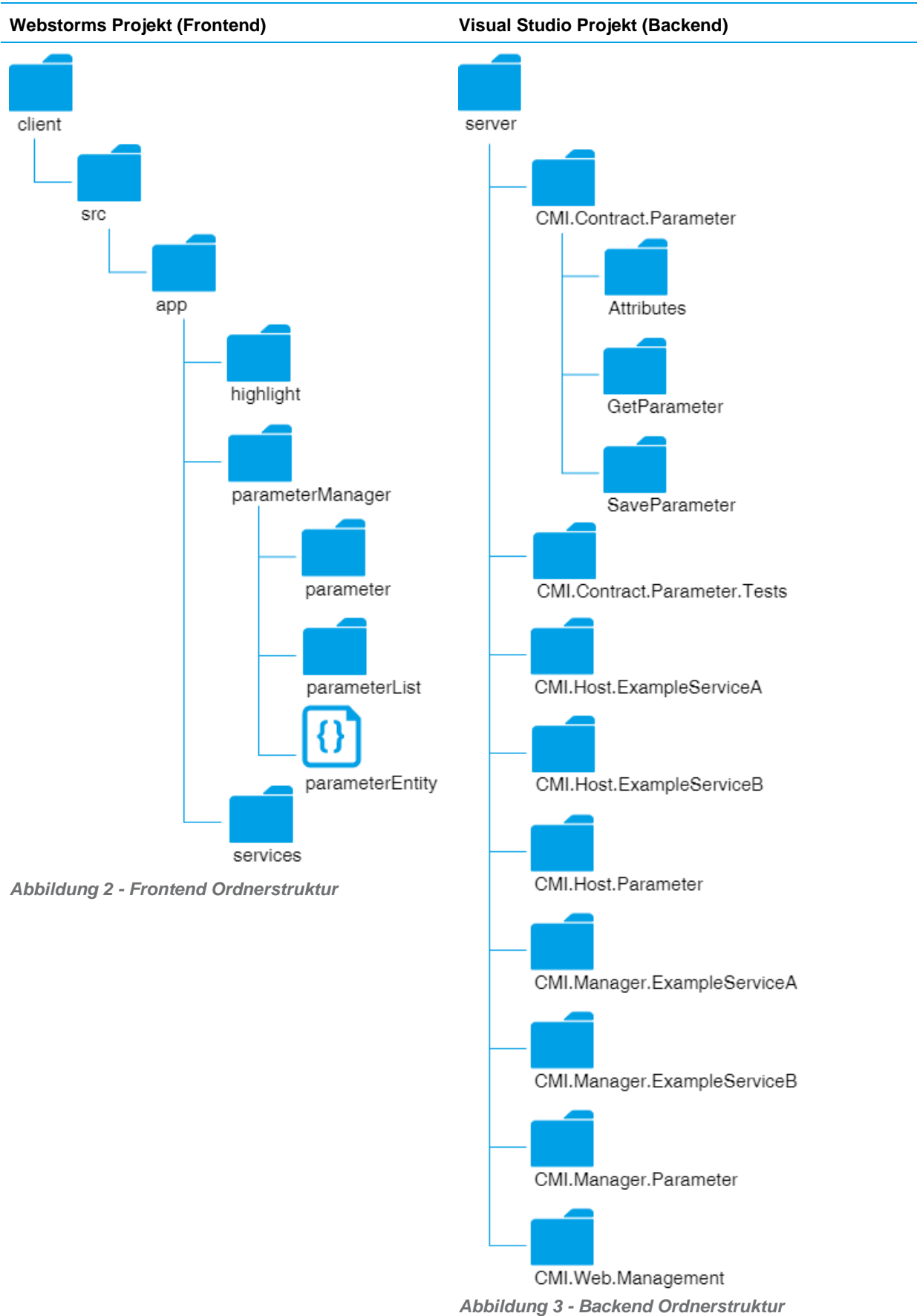


Abbildung 2 - Frontend Ordnerstruktur

Abbildung 3 - Backend Ordnerstruktur

Tabelle 2 - Source Code Ordnerstruktur

2.6.5 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.

3 Allgemeines

3.1 Konventionen

3.1.1 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 3 - Konventionen Dokumentation

3.1.2 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 4 - Code-Konventionen C#

3.1.3 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 5 - Code Konventionen TypeScript

3.1.4 Layout-Konventionen

Bereich	Vorgabe
Layoutrichtlinien	Richtlinien des Bundes sind einzuhalten. 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 6 - Layout-Konventionen

3.1.5 Unittests-Konventionen

Bereich	Vorgabe
Assemblyname	Die Testassembly muss gleich wie die normale Assembly heissen mit der Endung «.Tests» als Suffix.
Methodennamen	Upper Snake Case

Tabelle 7 - Unittests-Konventionen

3.2 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.

3.2.1 IPERKA



Abbildung 4 - 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.

4 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.

4.1 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.

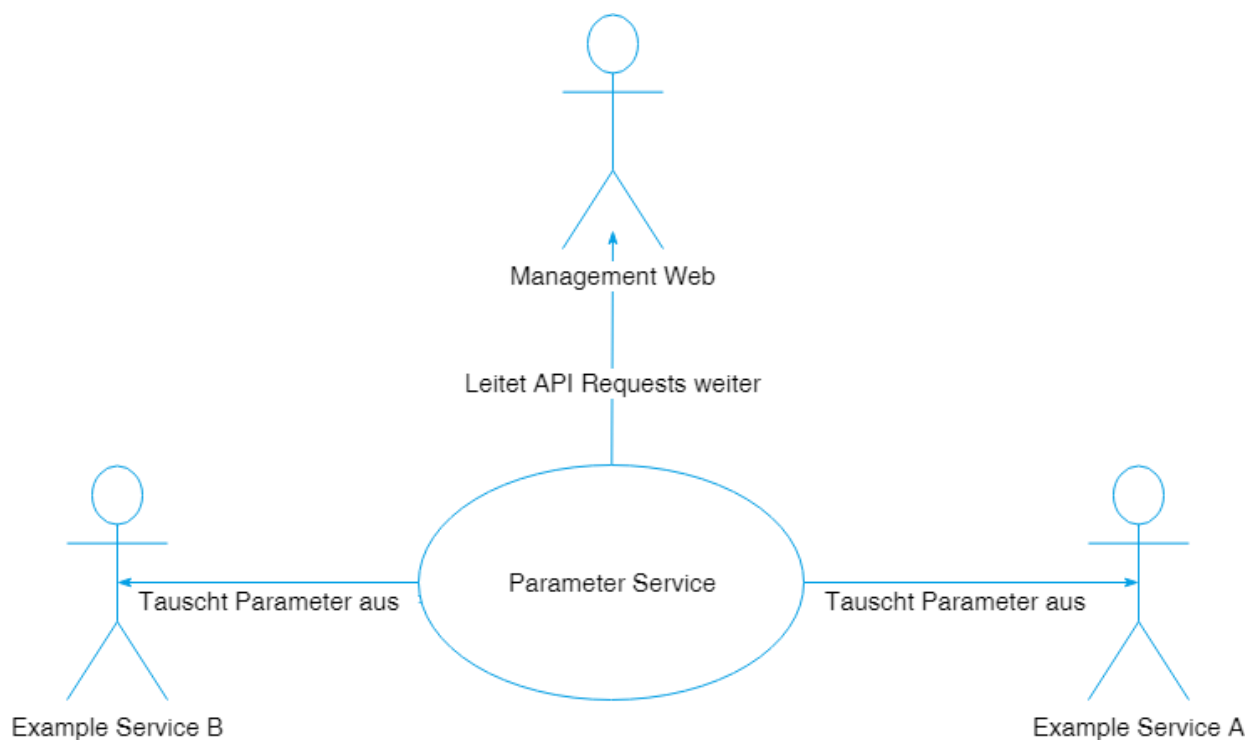


Abbildung 5 - Umsysteme

4.2 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.

4.3 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 8 - Systemgrenze Erklärungen Diagramm

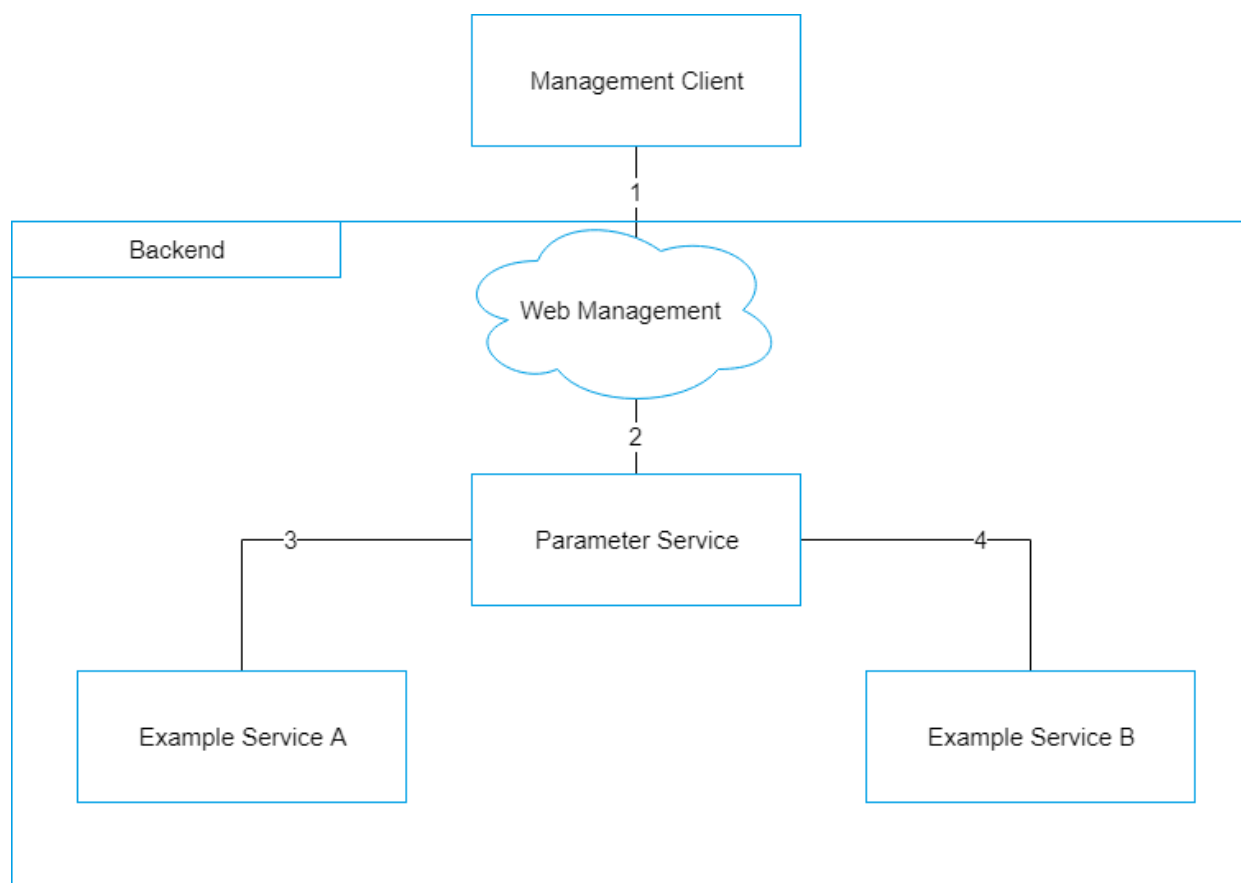


Abbildung 6 - Systemgrenzen

4.4 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:

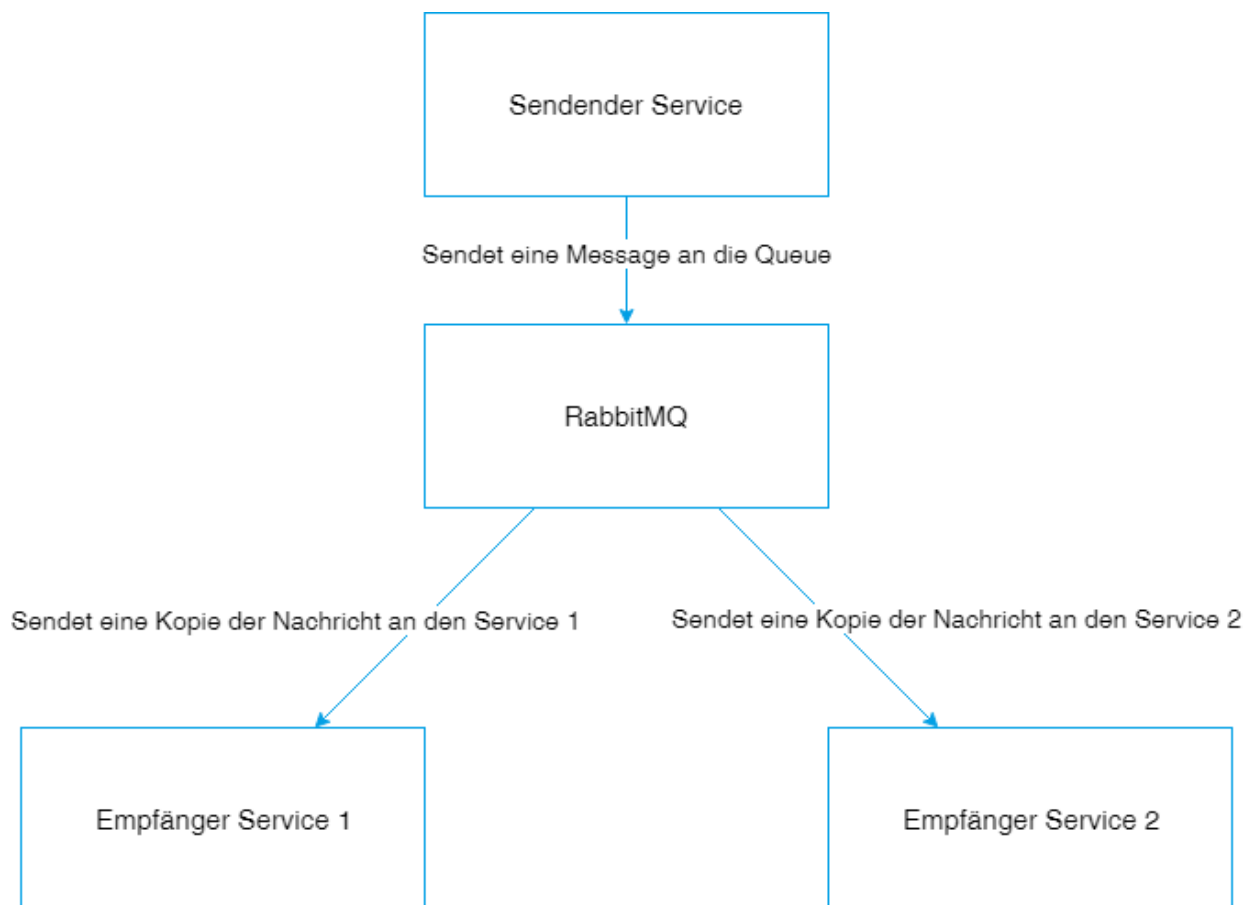


Abbildung 7 - RabbitMQ Event Kommunikation

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.

5 Tagesjournale

5.1 Tagesjournal vom 19.03.2018

Geplante Tagesziele

Nr.	Ziel	Soll	Ist	Abweichung	Abgeschlossen
1	Aufgabenbeschreibung und Allgemeines beschreiben	4	4	0	Ja
2	Umsystem und Abhängigkeiten beschreiben	1.5	1.5	0	Nein
3	Zeitplanerstellung / Taskerstellung	2	2	0	Ja

Tabelle 9 - Geplante Tagesziele Tag 1

Tätigkeit	Zeit	Aufwand	Ziel
Dokument erstellt, Aufgabenbeschreibungen erstellt	8.15 – 11.45 12.15-12.45	4	1
Umsysteme und Abhängigkeiten beschrieben	12.45 – 14.15	1.5	2
Tasks erstellt	14.30 – 15.30	1	3
Zeitplan erstellt	15.30 – 16.30	1	3
Tagesabschluss	16.30 – 17.00	0.5	-

Tabelle 10 - Tätigkeiten Tag 1

Probleme

Kriterium «eine Beispielimplementierung im Projekt Viaduc ist vorhanden und einsehbar» warf noch eine Frage auf. Siehe Hilfestellung. Des Weiteren scheint unsere Office at Work Vorlage noch einen kleinen Bug zu haben, welcher verhindert, dass Überschriften erster Stufe im Inhaltsverzeichnis erscheinen. Habe diesen Fehler unserer Expertin Melanie Müller geschickt und hoffe, ihn mit ihr diese Woche noch beheben zu können.

Tabelle 11 - Probleme Tag 1

Hilfestellung

Ich habe Matthias Hess gefragt, ob ich das Kriterium «eine Beispielimplementierung im Projekt Viaduc ist vorhanden und einsehbar» etwas komisch formuliert ist, denn es lässt vermuten, dass die IPA dann auch im Livesystem umgesetzt werden müsste. Dem ist jedoch nicht der Fall. Ich habe mir notiert, dies noch explizit mit Herr Schättin am Donnerstag an zu schauen, wenn er einen Besuch machen kommt.

Tabelle 12 - Hilfestellung Tag 1

Tagesreflexion

Ich bin überraschend gut vorangekommen. Für die zwei Probleme, denen ich heute begegnet bin, konnte ich schnell eine Lösung finden.

Tabelle 13 - Tagesreflexion Tag 1

Ich bin:

- Voraus
- **Im Plan**
- In Verzug

5.2 Tagesjournal vom 20.03.2018

Geplante Tagesziele

Nr.	Ziel	Soll	Ist	Abweichung	Abgeschlossen
1	Umsysteme und Abhängigkeiten beschreiben	0.5	0.5	0	Ja
2	Anforderungsanalyse pro Task	2	2	0	Ja
3	Vorgehensweise und Pattern-Erklärung	4	1.5	-2.5	Nein
4	Korrektur des Gegenlesens	1	0.25	- 0.75	Nein

Tabelle 14 - Geplante Tagesziele Tag 2

Tätigkeit	Zeit	Aufwand	Ziel
RabbitMQ Beschreibung der Kommunikation	8.15 – 8.45	0.5	1
Hilfe von Melanie Müller mit OAW Vorlage Inhaltsverzeichnis	8.45 – 9.00	0.25	-
Anforderungsanalyse und Umsetzungsreihenfolge definieren	9.00 – 11.00	2	2
Vorgehensweise und Pattern-Erklärung	11.00 – 12.00 12.30 – 13.00	1.5	3
Umsetzungserklärungen der einzelnen Tasks	13.00 – 16.00	3	-
Nachfrage nach der gegengelesenen Doku bei Matthias Hess, leider noch nicht dazu gekommen.	16.00 – 16.15	0.25	4
Tagesabschluss	16.15 – 16.30	0.25	-

Tabelle 15 - Tätigkeiten Tag 2

Probleme

Ich bin heute auf das Problem gestossen, dass ich während der Vorgehensweise und Pattern-Erklärung bemerkte, dass ich zuerst die Tasks definiert haben möchte, bevor ich genau die einzelnen Patterns etc. erkläre. Dies führt dazu, dass sich die Reihenfolge dieser zwei Tasks austauscht. So werde ich mich morgen vor allem mit der Pattern-Erklärung auseinandersetzen.

Tabelle 16 - Probleme Tag 2

Hilfestellung

Ich habe heute das Problem der OAW Vorlage mit Melanie Müller behoben.

Tabelle 17 - Hilfestellung Tag 2

Tagesreflexion

Der Tag verlief gut, nicht ganz wie geplant, aber ich habe durch das Schieben der Tasks keine Zeit verloren.

Tabelle 18 - Tagesreflexion Tag 2

Ich bin:

- Voraus
- **Im Plan**
- In Verzug

5.3 Tagesjournal vom 22.03.2018

Geplante Tagesziele

Nr.	Ziel	Soll	Ist	Abweichung	Abgeschlossen
1	Umsetzungserklärung der einzelnen Tasks	4	4	0	Ja
2	Expertenbesuch	1	1	0	Nein
3	Speichern der Parameter	2.5	2	-0.5	Ja
4	Erstellung der Unit Tests & des Testkonzept	1	0.5	-0.5	Nein

Tabelle 19 - Geplante Tagesziele Tag 3

Tätigkeit	Zeit	Aufwand	Ziel
Korrektur des Gegenlesens von Matthias Hess	7.30 – 8.00	0.5	-
Vorgehensweiseerklärung dokumentiert	8.00 – 9.00 11.00 – 12.30	2.5	-
Expertenbesuch	9.00 – 10.00	1	-
Umsetzungserklärung verfeinert / Negative Punkte von Herr Schättin verbessert.	10.00 -11.00	1	
Erstellung des Testkonzepts	13.30 – 14.00	0.5	4
Speichern der Parameter	14.00 – 16.00	2	3
Tagesjournal	16.00 – 16.30	0.5	-

Tabelle 20 - Tätigkeiten Tag 3

Probleme

Heute habe ich das Problem des einen Bewertungskriteriums mit Herr Schättin besprochen. Wir haben nun ein neues Kriterium an Herr Schöpflin geschickt. Dies wurde am selben Tag noch bestätigt und in der Dokumentation angepasst.

Tabelle 21 - Probleme Tag 3

Hilfestellung

Melanie Müller, unsere Office-Spezialistin hat mir gezeigt, wie man bei unseren Vorlagen ein Tabellenverzeichnis generieren kann.

Tabelle 22 - Hilfestellung Tag 3

Tagesreflexion

Da mir nicht klar war, ob der Expertenbesuch Teil der IPA-Zeit ist oder nicht, stehe ich momentan etwas im Verzug. Dies sollte ich morgen wieder einholen können.

Tabelle 23 - Tagesreflexion Tag 3

Ich bin:

- Voraus
- Im Plan
- In Verzug

5.4 Tagesjournal vom 23.03.2018

Geplante Tagesziele

Nr.	Ziel	Soll	Ist	Abweichung	Abgeschlossen
1	Speichern der Parameter	6	7	+1	Nein
2	Korrektur des Gegenlesens	1	0	-1	Nein
3	Kriterien F06 & Aufgabenstellung mit Stand vergleichen	0.5	0.5	0	Nein

Tabelle 24 - Geplante Tagesziele Tag 4

Tätigkeit	Zeit	Aufwand	Ziel
Json Serialisierung eines Parameters	8.15 – 9.30	1.25	1
API-Erweiterung, dass ein Parameter retour kommt und dies im Client anzeigen	9.30 – 11.30	2	1
API parametert, dass Lower Camel Case Json zurückgegeben wird.	11.30 – 11.45	0.25	1
Kriterien F06 & Aufgabenstellung mit Stand vergleichen	12.30 – 13.00	0.5	3
Backend Helper-Klassen schreiben für Json Serialisierung / Deserialisierung	13.00 – 16.30	3.5	1
Tagesjournal	16.30 – 17.00	0.5	-

Tabelle 25 - Tätigkeiten Tag 4

Probleme

Ein Problem, dass ich am Morgen hatte, war dass die API ein Upper Camel Case Json Objekt retour lieferte. Dies konnte ich dann nicht serialisieren. Mit der Hilfe von Benjamin Schäublin, einem Entwicklerkollegen von mir, konnte ich das Problem schnell lösen, es fehlte nur ein Attribut bei der API. Zudem bin ich lange an einem Punkt hängen geblieben, bei der das Binding einer Checkbox sich nicht updatet, wenn ich es via Code setze. Dies konnte ich mit der Angular Dokumentation von Google lösen.

Tabelle 26 - Probleme Tag 4

Hilfestellung

Google Angular Dokumentation, Benjamin Schäublin.

Tabelle 27 - Hilfestellung Tag 4

Tagesreflexion

Der Tag verlief gut. Ich bin schnell vorangekommen, leider konnte ich die Reflexion des Gegenlesens nicht machen, da ich die Dokumentation nicht wieder zurück bekommen habe heute. Sie sollte aber montags vorliegen. Das Programmieren der API gestaltet sich als schwieriger als erwartet. Der Einsatz von viel Reflection macht das ganze Arbeiten ziemlich Algorithmus-Lastig. Ich tüftle aber gerne, weshalb es mir dennoch Spass macht. Ich bin noch eine halbe Stunde in Verzug. Dies wird sich aber sicherlich im Verlaufe der nächsten Woche bessern.

Tabelle 28 - Tagesreflexion Tag 4

Ich bin:

- Voraus
- Im Plan
- In Verzug

5.5 Tagesjournal vom 26.03.2018

Geplante Tagesziele

Nr.	Ziel	Soll	Ist	Abweichung	Abgeschlossen
1	Speichern der Parameter	4.5	3.5	-1	Ja
2	Implementation des Validierungsmechanismus	3	2.5	-0.5	Nein
3	Durchführen der Tests	1	1	0	Nein

Tabelle 29 - Geplante Tagesziele Tag 5

Tätigkeit	Zeit	Aufwand	Ziel
Korrektur des Gegenlesens von Melanie Müller.	8.30 – 9.00	0.5	-
Speichern der Parameter fertiggestellt	9.00 – 11.45 12.30 – 13.15	3.5	1
Testing von Ziel 1, Fehlerbehebung der gefundenen Tests	13.15 – 14.15	1	1
Frontendseitige Validierung verbaut	14.15 – 16.45	2.5	2
Tagesabschluss	16.45 – 17.00	0.25	-

Tabelle 30 - Tätigkeiten Tag 5

Probleme

Ich hatte heute Schwierigkeiten mit RabbitMQ. Meine Service reagierten nicht. Dies konnte ich aber mit einem PC Neustart beheben.

Tabelle 31 - Probleme Tag 5

Hilfestellung

Heute habe ich keine Hilfestellung gebraucht.

Tabelle 32 - Hilfestellung Tag 5

Tagesreflexion

Ich bin sehr gut vorangekommen. Insbesondere das Speichern der Parameter fertig zu machen und die Frontendvalidierung gingen schneller als erwartet. Ich bin nun leicht dem Plan Voraus. Ich rechne damit, dass ich Morgen die Validierung früher fertig stellen kann als erwartet. Die eingesparte Zeit werde ich zum Feinschliff meiner Dokumentation einsetzen. Das Gegenlesen von Melanie Müller hat viel gebracht, da so einige Rechtschreibfehler korrigiert werden konnten.

Tabelle 33 - Tagesreflexion Tag 5

Ich bin:

- **Voraus**
- Im Plan
- In Verzug

5.6 Tagesjournal vom 27.03.2018

Geplante Tagesziele

Nr.	Ziel	Soll	Ist	Abweichung	Abgeschlossen
1	Implementation des Validierungsmechanismus	5	5.5	+0.5	Ja
2	Suche eines Parameters	1.5	1	-0.5	Nein
3	Durchführen der Tests	1	1	0	Nein

Tabelle 34 - Geplante Tagesziele Tag 6

Tätigkeit	Zeit	Aufwand	Ziel
Fertigstellung der serverseitigen Validierung.	8.30 – 12.00 12.45 – 14.45	5.5	1
Durchführen der Tests	14.45 – 15.45	1	3
Suche eines Parameters	15.45 – 16.45	1	2
Arbeitsjournal	16.45 – 17.15	0.5	-

Tabelle 35 - Tätigkeiten Tag 6

Probleme

Ich bin bei der serverseitigen Validierung lange stecken geblieben, da ich null in die API bekam. Nachdem Visual Studio abgestürzt ist ging es. Ich habe die ganze Zeit das Problem beim Inputbinding gesucht. Dadurch habe ich viel Zeit verloren.

Tabelle 36 - Probleme Tag 6

Hilfestellung

Ich habe Martin Tinner gefragt, wie ich es in der Dokumentation mit den vielen englischen Fachbegriffen regeln soll, da ich versucht habe alle zu übersetzen. Auf Rat von ihm, werde ich einfach all jene, die nicht geläufig sind ins Glossar aufnehmen.

Tabelle 37 - Hilfestellung Tag 6

Tagesreflexion

Ich habe viel Zeit verloren mit dem Problem von Visual Studio, konnte jedoch sehr schnell eine Möglichkeit fürs Suchen erarbeiten, was mich einiges an Zeit aufholen liess.

Tabelle 38 - Tagesreflexion Tag 6

Ich bin:

- Voraus
- **Im Plan**
- In Verzug

5.7 Tagesjournal vom 29.03.2018

Geplante Tagesziele

Nr.	Ziel	Soll	Ist	Abweichung	Abgeschlossen
1	Suche eines Parameters	6.5	7.5	+1	Ja
2	Testen	1	0	-1	Nein
3	Kriterien F06 & Aufgabenstellung mit Stand vergleichen	0.5	0.5	0	Nein

Tabelle 39 - Geplante Tagesziele Tag 7

Tätigkeit	Zeit	Aufwand	Ziel
Suche eines Parameters fertigstellen	7.00 – 9.00	2	1
Dokumentation der Realisierung Verfeinert, Codedokumentation Backend fertig und Frontend angefangen.	9.00 – 12.30 13.00 – 15.00	5.5	1
Kriterien	15.00 – 15.30	0.5	3
Tagesjournal	15.30 – 16.00	0.5	-

Tabelle 40 - Tätigkeiten Tag 7

Probleme

Keine Probleme

Tabelle 41 - Probleme Tag 7

Hilfestellung

Keine Hilfestellungen

Tabelle 42 - Hilfestellung Tag 7

Tagesreflexion

Ich bin gut vorangekommen. Die Codedokumentation sieht bereits sehr gut aus.

Tabelle 43 - Tagesreflexion Tag 7

Ich bin:

- Voraus
- **Im Plan**
- In Verzug

5.8 Tagesjournal vom 03.04.2018

Geplante Tagesziele

Nr.	Ziel	Soll	Ist	Abweichung	Abgeschlossen
1	Erstellung der Unittests	3	4	+1	Ja
2	Korrektur des Gegenlesens	1	2	+1	Ja
3	Getting Started Doku	2	0	-2	Nein
4	Durchführen der Tests	1	1	0	Ja

Tabelle 44 - Geplante Tagesziele Tag 8

Tätigkeit	Zeit	Aufwand	Ziel
Korrektur des Gegenlesens meiner Schwester Shana Kessler	8.15 – 10.15	2	2
Erstellung der Unittests	10.15 -11.45 12.30 – 15.00	4	1
Durchführen der Tests	15.00 – 16.00	1	4
Tagesjournal	16.00 – 16.30	0.5	-

Tabelle 45 - Tätigkeiten Tag 8

Probleme

Heute hatte ich keine Probleme.

Tabelle 46 - Probleme Tag 8

Hilfestellung

Keine Hilfestellungen.

Tabelle 47 - Hilfestellung Tag 8

Tagesreflexion

Ich konnte den ganzen Tag durch gut arbeiten, die Unittests haben jedoch länger gebraucht als erwartet. Ich nehme nicht an, dass ich noch Zusatzfunktionalität umsetzen kann im Rahmen der IPA.

Tabelle 48 - Tagesreflexion Tag 8

Ich bin:

- Voraus
- **Im Plan**
- In Verzug

5.9 Tagesjournal vom 05.04.2018

Geplante Tagesziele

Nr.	Ziel	Soll	Ist	Abweichung	Abgeschlossen
1	Korrektur der gefundenen Fehler	2	2	0	Ja
2	Konzept und Umsetzungsvergleich	4	5.5	+ 1.5	Ja

Tabelle 49 - Geplante Tagesziele Tag 9

Tätigkeit	Zeit	Aufwand	Ziel
Getting Started Dokumentation	7.00 – 9.00	2	-
Korrektur der gefundenen Fehler	9.00 – 11.00	2	1
Konzept und Umsetzungsvergleich	11.00 – 12.00 13.00 – 15.30	3.5	2
Tagesjournal	15.30 – 16.00	0.5	-

Tabelle 50 - Tätigkeiten Tag 9

Probleme

Heute ist aufgefallen, dass ein Klemmheft etwas wenig für ca. 100 Seiten ist. Deshalb habe ich den Experten Patrick Schättin gefragt, ob ein Ordner für die Abgabe auch in Ordnung wäre

Tabelle 51 - Probleme Tag 9

Hilfestellung

Keine Hilfestellung

Tabelle 52 - Hilfestellung Tag 9

Tagesreflexion

Der Tag verlief gut. Ich konnte viel an meiner Dokumentation arbeiten und einige Fehler noch beheben.

Tabelle 53 - Tagesreflexion Tag 9

Ich bin:

- Voraus
- **Im Plan**
- In Verzug

5.10 Tagesjournal vom 06.05.2018

Geplante Tagesziele

Nr.	Ziel	Soll	Ist	Abweichung	Abgeschlossen
1	Korrektur des Gegenlesens	1	1	0	Ja
2	Kriterien F06 & Aufgabenstellung mit Stand vergleichen	0.5	1	+0.5	Ja
3	Expertenbesuch	0.5	0	0.5	Ja
4	Reflexion	1	1	0	Ja
5	Zeitpuffer	4	4.5	+ 0.5	Ja

Tabelle 54 - Geplante Tagesziele Tag 10

Tätigkeit	Zeit	Aufwand	Ziel
Korrekturen des Gegenlesens von Melanie Müller	8.00 - 9.00	1	1
Kriterien F06 & Aufgabenstellung mit Stand vergleichen	9.00 – 10.00	1	2
Zeitpuffer verwendet für einfügen des Zeitplans und dessen Beschreibung	10.00- 12.00	2	5
Reflexion	13.00 – 14.00	1	4
Zeitpuffer für Reflexion	14.00 – 14.30	0.5	4
Tagesjournal	14.30 – 15.00	0.5	-
Zeitpuffer für Drucken und IPA Versand	15.00 – 17.00	2	-

Tabelle 55 - Tätigkeiten Tag 10

Probleme

Keine Probleme

Tabelle 56 - Probleme Tag 10

Hilfestellung

Keine Hilfestellungen

Tabelle 57 - Hilfestellung Tag 10

Tagesreflexion

Rückblickend auf die Ganze IPA hat mir das Projekt viel Spass gemacht. Das komplett selbständige Arbeiten war zwar herausfordernd aber auch schön. Ich bin zum Glück mit allem fertig geworden, weshalb ich nun mit gutem Gewissen drucken und anschliessend abgeben kann.

Tabelle 58 - Tagesreflexion Tag 10

Ich bin:

- Voraus
- **Im Plan**
- In Verzug

5.11 Allgemeines zum Tagesjournal

Das Tagesjournal wurde entweder in ausgedruckt oder elektronisch Martin Tinner gezeigt. Er hat diese jeweils am Ende des Tages gesehen.

Datum	Kandidat	Verantwortlicher
-------	----------	------------------

Tabelle 59 - Tagesjournale gesehen

6 Taskerstellung nach den Anforderungen

Die Einzeltasks sind eine Abbildung der Use-Cases und der Kriterien. Sie formulieren spezifische Umsetzungspunkte. Die Use-Cases der Anwendung sehen wie folgt aus:

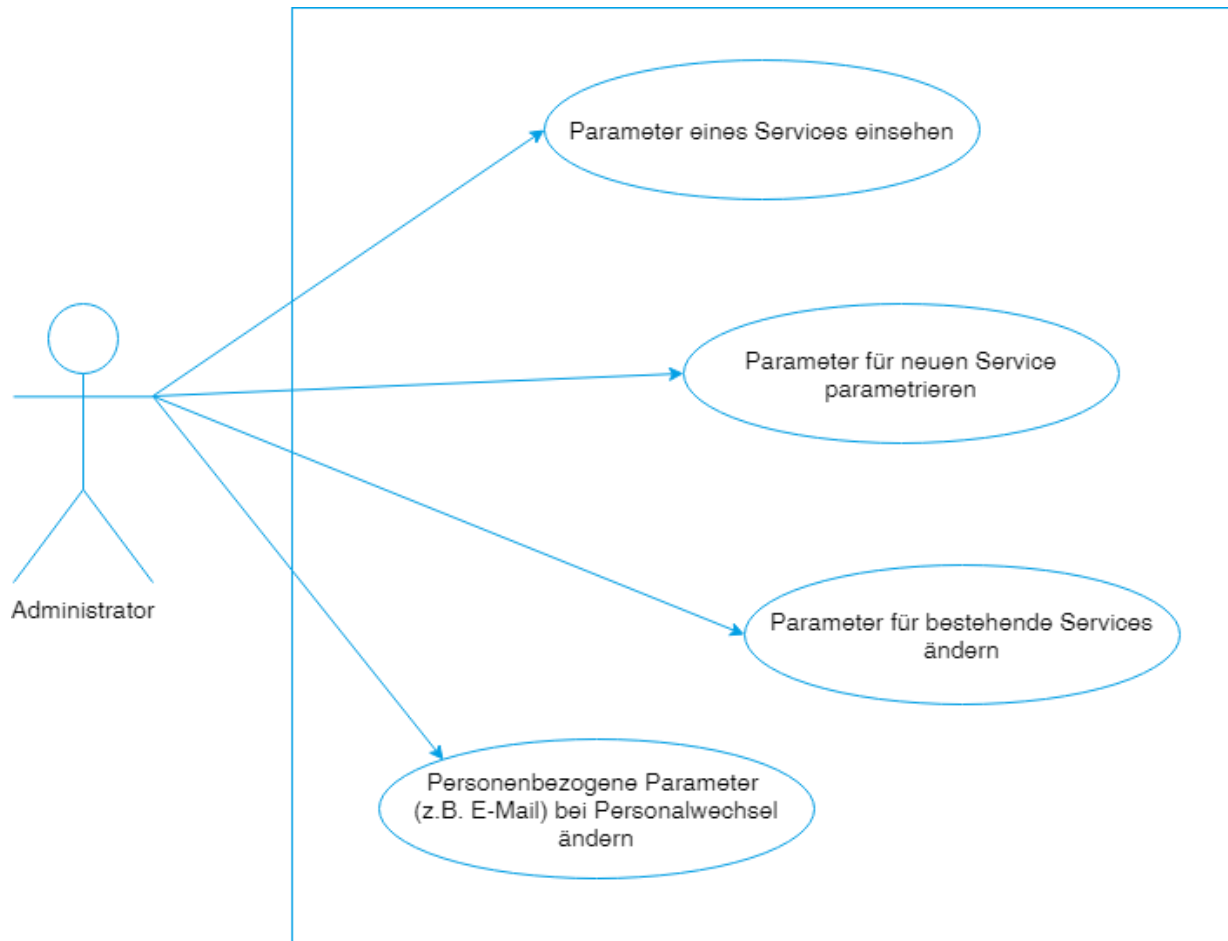


Abbildung 8 - Use-Case Diagramm

Das Use-Case Diagramm stellt den Akteur, in diesem Fall den Systemadministrator, und die Anwendungsfälle dar. Da diese Applikation für einen Administrator gedacht ist, steht die Funktionalität im Vordergrund. Die Use-Cases sind nach den bisherigen Anforderungen im Viaduc aufgestellt. Die Parameter sind technischer (z.B. ein DB-Connection-String), fachlicher (z.B. verschiedene Dauern und Daten für bestimmte Vorgänge) oder menschlicher (z.B. eine E-Mailadresse oder Personenverweise in Texten) Natur.

6.1 Einzeltasks für die Dokumentation

6.1.1 Getting Started Dokumentation

Zu erfüllende 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 60 - 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 61 - Getting Started Dokumentation Anforderungen

Geschätzter Aufwand: 2h

6.2 Einzeltasks für die Realisierung

6.2.1 Anzeige & Speichern der Parameter

Zu erfüllende 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

Die Einbindung der Parameterverwaltung soll mit der Angabe des Parametertyps und einem Minimum von weiterem Code mögl. sein.

Tabelle 62 - 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 63 - Anzeige & Speichern der Parameter Anforderungen

Geschätzter Aufwand: 12h

6.2.2 Implementation des Validierungsmechanismus

Zu erfüllende 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 64 - 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 65 - Implementation des Validierungsmechanismus Anforderungen

Geschätzter Aufwand: 8h

6.2.3 Suchen eines Parameters

Zu erfüllende Ziele

Das Auffinden des gewünschten Parameters wird vom System unterstützt

Tabelle 66 - 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 67 - Suchen eines Parameters Anforderungen

Geschätzter Aufwand: 8h

6.2.4 Erstellung der Unit Tests und des Testkonzepts

Anforderung

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

Tabelle 68 - Erstellung der Unit Tests und des Testkonzepts Anforderungen

Gesamter geschätzter Aufwand: 4h

6.3 Einzeltaskerstellung der Kontrolltasks

6.3.1 Durchführen der Tests

Zu erfüllende 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

Es werden sprechende Namen für Klassen / Methoden verwendet

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

Tabelle 69 - 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 70 - 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

7 Zeitplan & Burn down

Der Zeitplan ist nach Tagen respektive Halbtagen gegliedert. AM ist der Vormittag PM der Nachmittag.

7.1 Zeitplan

Phase	Kurzbeschreibung	Zeit- schät- zung	Montag 19.03.2018				Dienstag 20.03.2018				Donnerstag 22.03.2018				Freitag 23.03.2018				Montag 26.03.2018			
			AM		PM		AM		PM		AM		PM		AM		PM		AM		PM	
			Soll	Ist	Soll	Ist	Soll	Ist	Soll	Ist	Soll	Ist	Soll	Ist	Soll	Ist	Soll	Ist	Soll	Ist	Soll	Ist
I	Aufgabenstellung und Allgemeines beschreiben	4	4	3.5	0.5																	
P	Umsysteme und Abhängigkeiten beschreiben	2			1.5	1.5	0.5	0.5														
	Zeitplanerstellung / Taskerstellung	2			2	2																
E	Anforderungsanalyse pro Task	2					2	2														
	Vorgehensweise und Paternerklärung	4					1	1	3	0.5	2.5											
	Umsetzungserklärungen der einzelnen Tasks	4							3		4	1										
R	Speichern der Parameter	13											2.5	2	3	3.5	3	3.5	4.5	3.5	3	2.5
	Implementation des Validierungsmechanismus	8																				
	Suche eines Parameters	8																				
	Erstellung der Unit Tests & des Testkonzept	4											1	0.5								
	Zeitpuffer / Zusatzfunktionen	4																				
K	Durchführen der Tests	4																			1	1
	Korrektur des Gegenlesens	4					1		0.5		0.5				1		0.5	0.5	0.5			
	Kriterien F06 & Aufgabenstellung mit Stand vergleichen	1.5																				
	Expertenbesuch	1.5									1	1										
	Korrektur der gefundenen Fehler	2																				
A	Tagesjournal & Tages-Versionserstellung	5			0.5	0.5			0.5	0.5			0.5	0.5			0.5	0.5			0.5	0.5
	Reflexion	1																				
	Getting Started	2																				
	Konzept und Umsetzungsvergleich	4																				
Total		80	4	3.5	4	4.5	4.5	3.5	3.5	4.5	5	5	4	3	4	3.5	4	4.5	4.5	4	4.5	4
Burn down		80	76	77	72	72	68	69	64	64	59	59	55	56	51	53	47	48	43	44	38	40

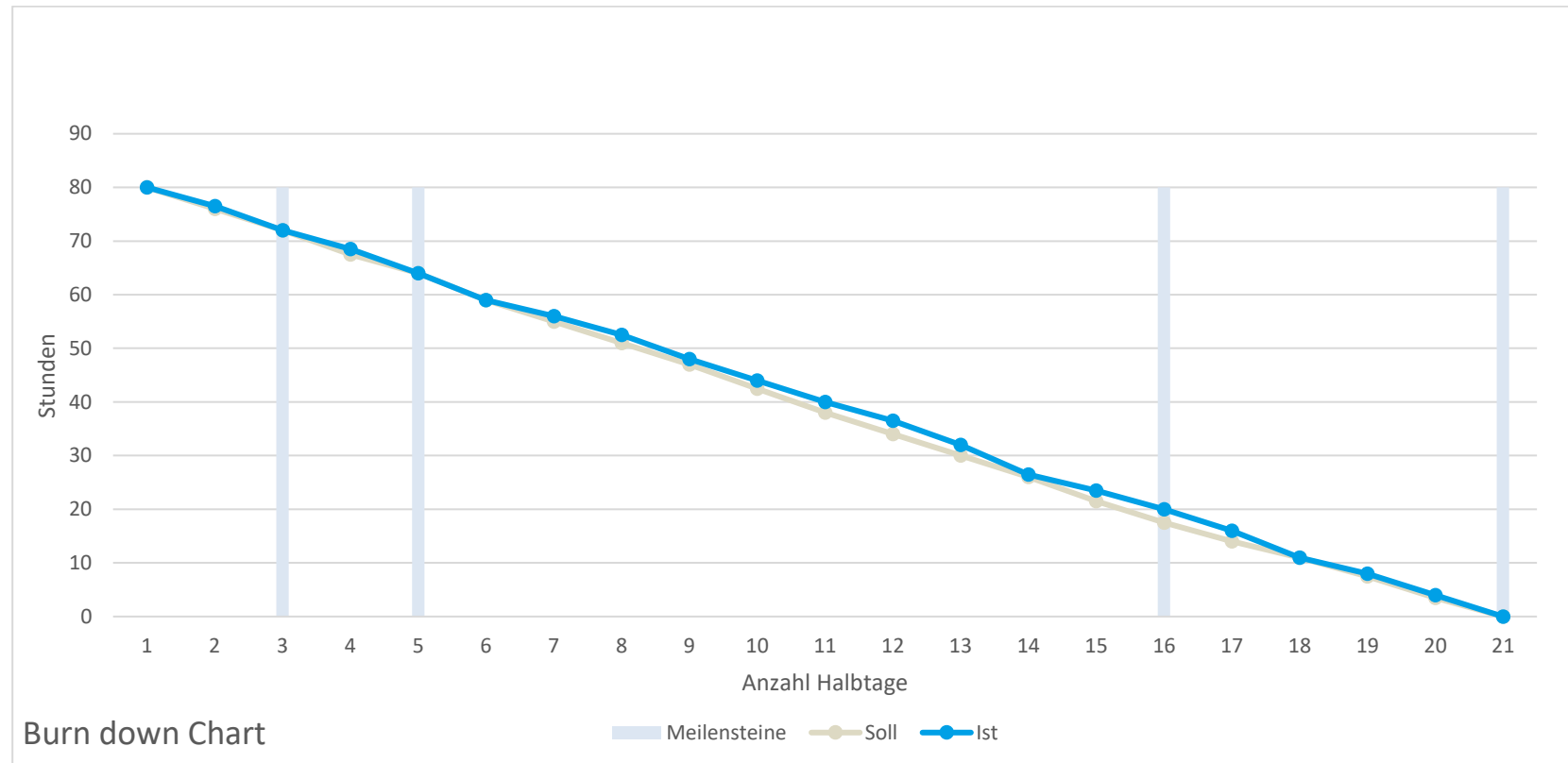
Phase	Kurzbeschreibung	Zeit- schät- zung	Dienstag 27.03.2018				Donnerstag 29.03.2018				Dienstag 03.04.2018				Donnerstag 05.04.2018				Freitag 06.04.2018			
			AM		PM		AM		PM		AM		PM		AM		PM		AM		PM	
			Soll	Ist	Soll	Ist	Soll	Ist	Soll	Ist	Soll	Ist	Soll	Ist	Soll	Ist	Soll	Ist	Soll	Ist	Soll	Ist
I	Aufgabenstellung und Allgemeines beschreiben	4																				
P	Umsysteme und Abhängigkeiten beschreiben	2																				
	Zeitplanerstellung / Taskerstellung	2																				
E	Anforderungsanalyse pro Task	2																				
	Vorgehensweise und Paternerklärung	4																				
	Umsetzungserklärungen der einzelnen Tasks	4																				
R	Speichern der Parameter	13																				
	Implementation des Validierungsmechanismus	8	4	3.5	1	2																
	Suche eines Parameters	8			1.5	1	4	5.5	2.5	2												
	Erstellung der Unit Tests & des Testkonzept	4									3	1.5	2.5									
	Zeitpuffer / Zusatzfunktionen	4																	2	2	2	2.5
K	Durchführen der Tests	4			1	1			1				1	1								
	Korrektur des Gegenlesens	4									1	2							1	1		
	Kriterien F06 & Aufgabenstellung mit Stand vergleichen	1.5							0.5	0.5									0.5	1		
	Expertenbesuch	1.5																	0.5			
	Korrektur der gefundenen Fehler	2													2							
A	Tagesjournal & Tages-Versionserstellung	5			0.5	0.5			0.5	0.5			0.5	0.5			0.5	0.5			0.5	0.5
	Reflexion	1																			1	1
	Getting Started	2											2									
	Konzept und Umsetzungsvergleich	4													1	2.5	3	3				
Total		80	4	3.5	4	4.5	4	5.5	4.5	3	4	3.5	3.5	4	3	4.5	3.5	3.5	4	4	3.5	4
Burn down		80	34	37	30	32	26	27	22	24	18	20	14	16	11	12	7.5	8	3.5	4	0	0

Erklärungen zum Zeitplan

Der Morgen und Nachmittag zusammen ergeben jeweils einen Tag. Der Morgen ist jeweils weiss und der Nachmittag grau. Die Tasks sind auf eine halbe Stunde genau geschätzt. Die Einteilung der Zeitachse ist jedoch auf Halbtage (vier Stunden) genau. Die Reihenfolge ist möglichst so gewählt, dass man bis und mit Realisierungsphase von oben nach unten sich durcharbeiten kann. Die Korrektur- und Auswertungsphase passiert fortlaufend, weshalb diese Tasks so verteilt sind, wie sie fällig sind. Die Meilensteine sind jeweils nach der Planungs-, Entscheidungs- und Realisationsphase und bei Projektende nach der Auswertung. Die Kontrollphase ist über das ganze Projekt verteilt, weshalb sie erst mit Abgabe wirklich zu Ende ist.

7.2 Burn down

Das Burn down Chart zeigt grafisch auf, wie viele Stunden für welchen Schritt benötigt wurden.



8 Entscheide der einzelnen Tasks

8.1 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 71 - Umsetzungsreihenfolge

8.2 Erstellung des Testkonzepts

8.2.1 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.

8.2.2 Mögliche Lösungen

8.2.2.1 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 72 - Lösungsmatrix Unittests

8.2.2.2 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 auszuführen	15 / 25
Einfach zu schreiben / definieren	10 / 25
Zeitaufwand des Tests	25 / 25
Zeitaufwand des Schreibens der Test	0 / 25
Total	50 / 100

Tabelle 73 - Lösungsmatrix Integration Tests / e2e Tests

8.2.2.3 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 sind, da ein Computer nicht alles testen kann (z.B. sieht die Webseite optisch gut aus).

Kriterium	Punktzahl
Einfach auszuführen	25 / 25
Einfach zu schreiben / definieren	25 / 25
Zeitaufwand des Tests	5 / 25
Zeitaufwand des Schreibens der Test	25 / 25
Total	80 / 100

Tabelle 74 - Lösungsmatrix User-Testing

8.2.3 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.

8.3 Anzeige & Speichern der Parameter

8.3.1 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.

8.3.2 Mögliche Lösungen Speichern

8.3.2.1 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 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 auszulesen	25 / 25
Einfach zu schreiben	25 / 25
Einfach erweiterbar	5 / 25
Fix definierte Form	0 / 25
Total	55 / 100

Tabelle 75 - Lösungsmatrix Speichern der Parameter als Parametertyp im Json

8.3.2.2 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 auszulesen	10 / 25
Einfach zu schreiben	15 / 25
Einfach erweiterbar	25 / 25
Fix definierte Form	25 / 25
Total	75 / 100

Tabelle 76 - Lösungsmatrix Speichern der Parameter als generischer Typ im Json

8.3.3 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.

8.3.4 Mögliche Lösung Anzeige

8.3.4.1 Ähnlich wie in Firefox / Waterfox die about:config

Search:

Preference Name	Status	Type	Value
browser.bookmarks.autoExportHTML	default	boolean	false
browser.bookmarks.editDialog.firstEditField	default	string	namePicker
browser.bookmarks.max_backups	default	integer	15
browser.bookmarks.showMobileBookmarks	modified	boolean	false
browser.bookmarks.showRecentlyBookmarked	default	boolean	true
browser.places.smartBookmarksVersion	modified	integer	8
browser.tabs.loadBookmarksInBackground	default	boolean	false
browser.uiCustomization.state	modified	string	{"placements":{"PanelUI-contents":["edit-contr...
browser.urlbar.suggest.bookmark	default	boolean	true
extensions.webextensions.themes.icons.buttons	default	string	back,forward,reload,stop,bookmark_star,bookmark...
places.frecency.bookmarkVisitBonus	default	integer	75
places.frecency.unvisitedBookmarkBonus	default	integer	140
services.sync.engine.bookmarks	default	boolean	true
services.sync.engine.bookmarks.validation.inter...	default	integer	86400
services.sync.engine.bookmarks.validation.max...	default	integer	1000
services.sync.engine.bookmarks.validation.perc...	default	integer	10
services.sync.log.logger.engine.bookmarks	default	string	Debug
services.sync.prefs.sync.browser.urlbar.suggest....	default	boolean	true

Abbildung 9 - 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.

8.3.4.2 Ähnlich wie die Chrome Settings

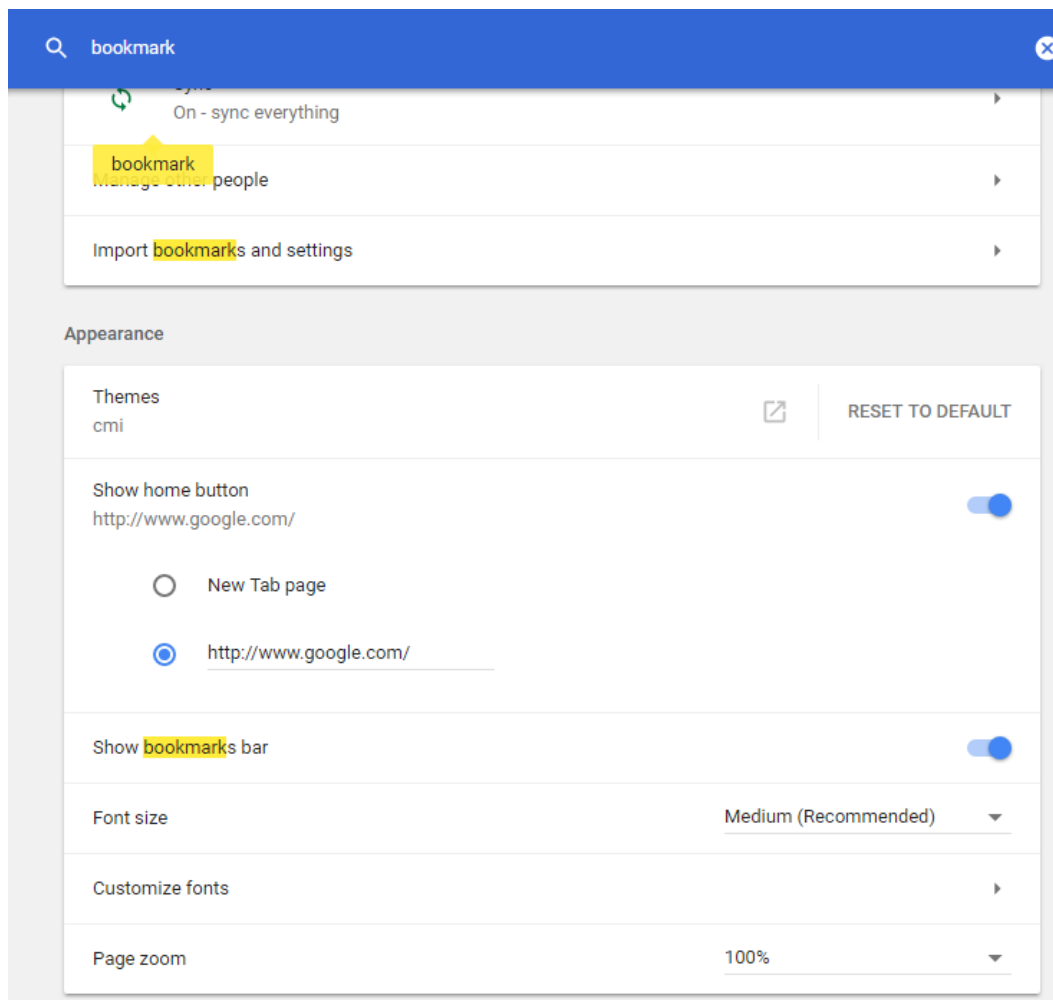


Abbildung 10 - Chrome Einstellungen

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.

8.3.4.3 Ähnlich wie in Visual Studio

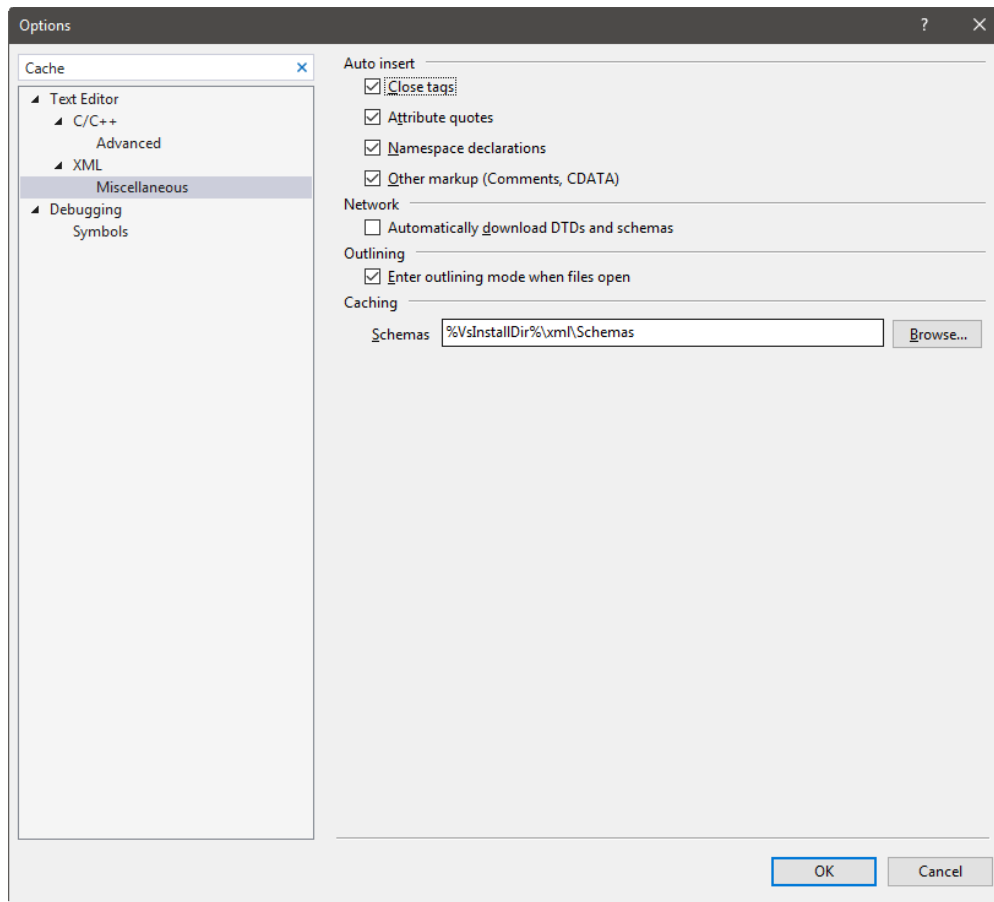


Abbildung 11 - Visual Studio Einstellungen

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 Checkboxes. 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.

8.3.5 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:

The mockup shows a web browser window with the address bar displaying 'https://www.ipa.ch'. Below the address bar is a search bar with the placeholder text 'Suchen nach...' and a magnifying glass icon. To the right of the search bar is a 'Validieren' button. Below the search bar are two parameter groups. The first group is titled 'Parameter Gruppe' and contains three rows: 'Parameter 1', 'Parameter 2', and 'Parameter 3'. Each row has a corresponding 'Parameter 1 Wert', 'Parameter 2 Wert', and 'Parameter 3 Wert' column. The second group is titled 'Parameter Gruppe 2' and contains two rows: 'Parameter 4' and 'Parameter 5'. Each row has a corresponding 'Meine Änderungen' and 'Parameter 5 Wert' column. A 'Speichern' button is located to the right of the 'Parameter 5 Wert' column.

Abbildung 12 - 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.

8.4 Implementation des Validierungsmechanismus

8.4.1 Anforderung

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

8.4.2 Mögliche Lösungen

8.4.2.1 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 77 - Lösungsmatrix nur serverseitig im Service den Parameter selbst testen lassen

8.4.2.2 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 78 - Lösungsmatrix Regular Expression im Parameter

8.4.3 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).

8.5 Suche eines Parameters

8.5.1 Anforderung

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

8.5.2 Mögliche Lösungen

8.5.2.1 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 79 - Lösungsmatrix Volltextsuche über alles

8.5.2.2 Suche auf die Parameternamen

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

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 80 - Lösungsmatrix Suche auf die Parameternamen

8.5.2.3 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 81 - Lösungsmatrix Suche auf Parameternamen und Parameterwert

8.5.3 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.

8.6 Erstellung der Unittests

8.6.1 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.

8.6.2 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.

8.7 Durchführen der Tests

8.7.1 Anforderung

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

8.7.2 Umsetzungsbeschreibung

Das Kapitel Testing wird geschrieben als Testprotokoll.

8.8 Getting Started Dokumentation

8.8.1 Anforderung

Die Anforderungen an diese Dokumentation sind aus der Aufgabenstellung her genau definiert. Es gilt eine «Step by Step» Anleitung zu schreiben. Diese soll einen konzeptionellen Überblick verschaffen und mindestens 2 Screenshots des GUIs und 2 Beispiele für die Validierung enthalten.

8.8.2 Umsetzungsbeschreibung

Das Getting Started ist sinnvollerweise in Word geschrieben. Für diese Projektarbeit wird die Getting Started in dieser Dokumentation eingefügt.

9 Realisierung

Das Fehlerhandling während der gesamten Realisierung ist nach dem Fail-Fast Pattern aufgebaut. So soll ein unerwarteter Fehler definitiv zum Absturz führen. Allgemein wurde vor allem Null Reference Exceptions abgefangen. Speichern ist nur nach der Validierung möglich. Falls ein falscher Wert z.B. über XSS übergeben wird, so passiert am Client nichts und der Parameter wird nicht gespeichert, da die Validierung fehlschlägt.

9.1 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 wurden eingehalten
Es können keine falschen Parameter gespeichert werden

Tabelle 82 - Testfälle

Die Testfälle müssen jeweils ausgeführt werden. Ist die Funktionalität hinter einem Testfall noch nicht vorhanden, so wird dies im Testprotokoll festgehalten. Die Testfälle sind Use-Case Tests der Applikation. Tritt während des Testens ein Fehler auf, so ist dieser gleich zu beheben.

9.2 Anzeige & Speichern der Parameter

9.2.1 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 der 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.

Die Gruppierung nach Service ist weggelassen, da es eine Zusatzanforderung ist und die Zeit dafür nicht ausreicht.

Name	Default	Value	Buttons
CMI.Manager.ExampleServiceA.Date	12.02.2017	3.7.15	Speichern Abbrechen
CMI.Manager.ExampleServiceA.EmailAdress		example@mymail.com	
CMI.Manager.ExampleServiceA.ServiceOn	false	<input checked="" type="checkbox"/>	
CMI.Manager.ExampleServiceA.NumberOfIterations		25	
CMI.Manager.ExampleServiceB.EndeDatum	12.02.2017	12.02.2015	
CMI.Manager.ExampleServiceB.EmailAdress		myothermail@hotmail.com	
CMI.Manager.ExampleServiceB.FehlerVerstecken	true	<input checked="" type="checkbox"/>	
CMI.Manager.ExampleServiceB.Sekunden		60	

Abbildung 13 - Screenshot umgesetzte Lösung Anzeige & Speichern der Parameter

9.2.2 Backend

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

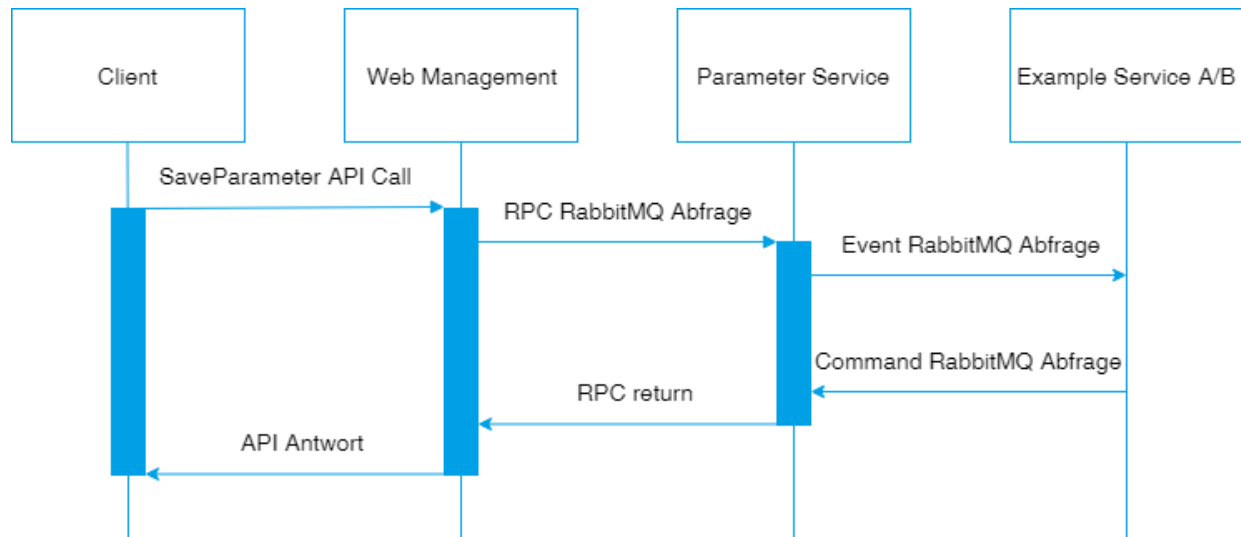


Abbildung 14 - Sequenzdiagramm Servicekommunikation beim «Save»

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.

9.2.3 Technische Umsetzung Backend

Wird ein Parameter geladen, wird die API aufgerufen. Die API holt sich anschliessend via RabbitMQ alle Parameter. Dies geschieht, mit einem selbstgebaute 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.

9.2.4 Erstellte und Bearbeitete Klassen

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.cs
 - SaveParameterEventResponseConsumer
 - SaveParameterRequestConsumer
- CMI.Web.Management
 - Controllers
 - ParameterController

Frontend

- parameter
 - HTML: ParameterComponent
 - Less: ParameterComponent
 - Klasse: ParameterComponent
- parameterList
 - HTML:ParameterListComponent
 - Less: ParameterListComponent
 - Klasse: ParameterListComponent
- Klasse: parameterEntity
- services
 - Klasse: HttpService
 - Klasse ParameterService

9.3 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.

Name	Standardwert	Wert
CMI.Manager.ExampleServiceA.Date	12.02.2017	<input type="text" value="12.07.2017"/>
CMI.Manager.ExampleServiceA.EmailAddress		<input type="text" value="example@mymail.com"/>
CMI.Manager.ExampleServiceA.ServiceOn	false	<input checked="" type="checkbox"/>
CMI.Manager.ExampleServiceA.NumberOfIterations		<input type="text" value="25"/>
CMI.Manager.ExampleServiceB.EndeDatum	12.02.2017	<input type="text" value="12asdasda.02.2015"/>
CMI.Manager.ExampleServiceB.EmailAddress		<input type="text" value="myothermail@hotmail.com"/>
CMI.Manager.ExampleServiceB.FehlerVerstecken	true	<input checked="" type="checkbox"/>
CMI.Manager.ExampleServiceB.Sekunden		<input type="text" value="60"/>

Abbildung 15 – Screenshot umgesetzte Lösung Implementation des Validierungsmechanismus

9.3.1 Erstellte und bearbeitete Klassen

Backend

- CMI.Contract.Parameter
 - ParameterHelper.cs

Frontend

- parameter
 - HTML: ParameterComponent
 - Less: ParameterComponent
 - Klasse: ParameterComponent
- parameterList
 - HTML:ParameterListComponent
 - Less: ParameterListComponent
 - Klasse: ParameterListComponent

9.4 Suche eines Parameters

Die Suche funktioniert rein clientseitig, da kein Paging eingesetzt wird. Somit erzeugt die Suche kein Traffic. Die Suche geht auf den Parameternamen und den Parameterwert. Sind Treffer im Namen gefunden, wird diese Stelle gelb hervorgehoben. Wird ein Treffer im Wert gefunden, wird das ganze Inputfeld hervorgehoben. Die Suche ist nicht Case-Sensitiv.

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Management-Client des Bundesarchivs

Anmelden

Sie sind nicht angemeldet ⓘ

Zentralisierte Parameterverwaltung

mail

Name	Standardwert	Wert
CMI.Manager.ExampleServiceB.EmailAddress		mymail@someemail.ch
CMI.Manager.ExampleServiceA.EmailAddress		

Schweizerisches Bundesarchiv BAR

Abbildung 16 - Screenshot umgesetzte Lösung Suche eines Parameters

9.4.1 Erstellte und bearbeitete Klassen

Frontend

- parameter
 - HTML: ParameterComponent
 - Less: ParameterComponent
 - Klasse: ParameterComponent
- parameterList
 - HTML:ParameterListComponent
 - Less: ParameterListComponent
 - Klasse: ParameterListComponent
- highlight
 - HTML: HighlightComponent
 - Less: HighlightComponent
 - Klasse: HighlightComponent

9.5 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. Während des Schreibens der Unittests ist mir ebenfalls noch ein kleiner Fehler in der Validierung aufgefallen, diese konnte noch eine Nullreference Exception werfen. Dieser Fehler wurde noch kurzerhand gefixt.

9.5.1 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_correctly	Die Serialisierung des Namens funktioniert.
Can_be_serialized_saved_and_can_then_be_deserialized_and_got	Das Speichern und Auslesen eines validen Parameters funktioniert.

Tabelle 83 - Aussagen ParameterSerializer Tests

9.5.2 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.

Tabelle 84 - Aussage ParameterValidation Tests

9.5.3 Erstellte und bearbeitete Klassen

Backend

- CMI.Contract.Parameter.Tests
 - TestSetting
 - ParameterValidationTests
- CMI.Contract.Parameter
 - ParameterHelper

10 Getting Started mit der Parameterverwaltung

10.1 Einen eigenen Parameter schreiben

Um einen eigenen Parameter einem Service hinzuzufügen gibt es 2 Möglichkeiten.

10.1.1 Ein Setting existiert und nur ein Parameter soll hinzugefügt werden

Im <MyService>Setting.cs im Service ist eine Klasse definiert, die ISetting implementiert.

- 1) Füge in diesem Setting den Parameter als eine Property hinzu.
- 2) Füge der Property die benötigten Attribute hinzu (Siehe Kapitel: «Unterstützte Attributtypen»).
- 3) Der Parameter kann nun parametrisiert werden.

10.1.2 Kein Setting existiert

- 1) Erstelle eine neue C# Klasse <MyService>Setting.cs im Service. Lass die Klasse ISettings implementieren
- 2) Das Interface befindet sich im CMI.Parameter.Contract, dieses muss noch im Service referenziert werden.
- 3) Füge in diesem Setting den Parameter als eine Property hinzu.
- 4) Füge der Property die benötigten Attribute hinzu (Siehe Kapitel: «Unterstützte Attributtypen»).
- 5) Geh zu der Start-Methode im <MyService>. Hier müssen noch die <MyService> durch den Servicennamen ersetzt werden. Die Console.Out.WriteLineAsync können durch andere Logmethoden ersetzt werden. Ändere das Skript demzufolge ab und füge in der Start-Methode folgende zwei RabbitMQ Endpunkte ein:

```
BusConfigurator.ConfigureBus((cfg, host) =>
{
    cfg.ReceiveEndpoint(host, "GetAllParameters<MyService>", ep =>
    {
        ep.Handler<GetParameterEvent>(context =>
        {

            ParameterBusHelper.SubscribeGetEvent<<MyService>Setting>(ParameterBus);
            return Console.Out.WriteLineAsync("Get Parameters");
        });
    });
    cfg.ReceiveEndpoint(host, "SaveParameters<MyService>", ep =>
    {
        ep.Handler<SaveParameterEvent>(context =>
        {

            ParameterBusHelper.SubscribeSaveEvent<<MyService>Setting>(ParameterBus,
context.Message.Parameter);
            return Console.Out.WriteLineAsync("Saved Parameter");
        });
    });
});
```

10.2 Einen Parameter/ein Setting auslesen

Ein Parameter respektive ein Setting kann wie folgt ausgelesen werden.

```
var setting = ParameterHelper.GetSetting(new <MySetting>Setting());  
var param = setting.<MyParameter>;
```

10.3 Beispielsparameter

Hier sind vier Beispiele für häufige Parameter.

```
public class ExampleSetting : ISetting  
{  
    [Default("false")]  
    public bool ExampleBool;  
  
    [Default("max.muster@supermail.ch")]  
    [Validation(@"([a-zA-Z0-9_]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+)")]  
    public string ExampleEMail;  
  
    [Validation(@"((0[1-9]|[12][0-9]|3[01])[- /.](0[1-9]|1[012])[-  
/.]?\\d\\d\\d\\d)")]  
    public string ExampleDate;  
  
    [Mandatory]  
    [Description("You can here describe your number!")]  
    public int ExampleNumber;  
}
```

10.3.1 Unterstützte Parametertypen


Es können String, Boolean, Int, Long, Float und Double verwendet werden. Es sind keine anderen Typen unterstützt.

10.3.2 Unterstützte Attributtypen

Mandatory (Muss Feld, wenn gesetzt), Default (Standardwert Hinweis für den Benutzer), Validation (Regex für die Validierung), Description (Beschreibung des Parameters für den Hilfetext)

10.4 Resultat:

Alle Parameter werden angezeigt



Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Management-Client des Bundesarchivs

Anmelden

Sie sind nicht angemeldet ⓘ


Zentralisierte Parameterverwaltung

Name	Standardwert	Wert
CMI.Manager.ExampleService.ExampleBool	false	<input type="checkbox"/>
CMI.Manager.ExampleService.ExampleEmail	max.muster@supermail.ch	<input type="text"/>
CMI.Manager.ExampleService.ExampleDate		<input type="text"/>
CMI.Manager.ExampleService.ExampleNumber		<input type="text" value="0"/>

Schweizerisches Bundesarchiv BAR

Abbildung 17 - Screenshot Getting Started

Die Validierung verhindert das Speichern eines falschen Parameters



Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Management-Client des Bundesarchivs

Anmelden

Sie sind nicht angemeldet ⓘ

Zentralisierte Parameterverwaltung

Name	Standardwert	Wert
CMI.Manager.ExampleService.ExampleBool	false	<input type="checkbox"/>
CMI.Manager.ExampleService.ExampleEmail	max.muster@supermail.ch	<input type="text"/>
CMI.Manager.ExampleService.ExampleDate		<input type="text"/>
CMI.Manager.ExampleService.ExampleNumber		<input type="text"/>

Schweizerisches Bundesarchiv BAR

Abbildung 18 - Screenshot Getting Started Validierung

11 Testing

11.1 Testdurchgang nach Task Anzeige & Speichern der Parameter

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.

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 wurden 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 85 - Resultate Testdurchgang 1

11.2 Testdurchgang nach Task Implementation des Validierungsmechanismus

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

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 wurden eingehalten	Funktioniert
Es können keine falschen Parameter gespeichert werden	Funktioniert

Tabelle 86 - Resultate Testdurchgang 2

11.3 Testdurchgang nach Task Suche eines Parameters

Es wurden in diesem Testdurchgang keine Fehler gefunden.

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 wurden eingehalten	Funktioniert
Es können keine falschen Parameter gespeichert werden	Funktioniert

Tabelle 87 - Resultate Testdurchgang 3

11.4 Testdurchgang nach Task Erstellung der Unittests

Es wurden in diesem Testdurchgang keine Fehler gefunden.

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 wurden eingehalten	Funktioniert
Es können keine falschen Parameter gespeichert werden	Funktioniert

Tabelle 88 - Resultate Testdurchgang 4

12 Code Dokumentation Backend

12.1 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.

12.1.1 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.

12.1.1.1 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();
}
```

12.1.1.2 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);
}
```

12.1.1.3 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;
    }
}
```

12.1.1.4 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;
}
```

12.1.1.5 Methode: InitialSaveParameter

Diese Methode initialisiert einen Parameter, der noch nicht vorhanden ist, indem ein leereres Setting eines `ISetting`typs 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);
}
```

12.1.1.6 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;
}
```

12.1.1.7 Methode: GetSettingPath

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

```
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);
}
```

12.1.1.8 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;
}
```

12.1.1.9 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";
}
```

12.1.2 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.

12.1.2.1 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) });
}
```


12.1.2.2 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) });
    }
}
```

12.1.3 Klasse: Parameter

Diese Klasse zeigt, wie ein Parameter auszusehen hat.

12.1.3.1 Property: Name

Der Name des Parameters.

```
public string Name { get; set; }
```

12.1.3.2 Property: Type

Der Typ des Parameters.

```
public string Type { get; set; }
```

12.1.3.3 Property: Description

Ein Hilfetext des Parameters.

```
public string Description { get; set; }
```

12.1.3.4 Property: Value

Der parametrisierte Wert des Parameters.

```
public string Value { get; set; }
```

12.1.3.5 Property: Default

Die Standardeinstellung des Parameters.

```
public string Default { get; set; }
```

12.1.3.6 Property: RegexValidation

Der Validierungs-String des Parameters.

```
public string RegexValidation { get; set; }
```

12.1.3.7 Property: Mandatory

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

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

12.1.4 Interface: **ISetting**

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

12.1.5 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.

12.1.6 Ordner: Attributes

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

12.1.6.1 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;
    }
}
```

12.1.6.2 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;
    }
}
```

12.1.6.3 Klasse: MandatoryAttribute

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

```
[AttributeUsage(AttributeTargets.Field)]
public class MandatoryAttribute : Attribute
{
}
```

12.1.6.4 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;
    }
}
```

12.1.7 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.

12.1.7.1 Klasse: GetParameterEvent

```
public class GetParameterEvent
{
}
```

12.1.7.2 Klasse: GetParameterEventResponse

```
public class GetParameterEventResponse
{
    public Parameter[] Parameters { get; set; }
}
```

12.1.7.3 Klasse: GetParameterRequest

```
public class GetParameterRequest
{
}
```

12.1.7.4 Klasse: GetParameterResponse

```
public class GetParameterResponse
{
    public Parameter[] Parameters { get; set; }
}
```

12.1.8 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.

12.1.8.1 Klasse: SaveParameterEvent

```
public class SaveParameterEvent
{
    public Parameter Parameter { get; set; }

    public SaveParameterEvent(Parameter parameter)
    {
        Parameter = parameter;
    }
}
```

12.1.8.2 Klasse: SaveParameterEventResponse

```
public class SaveParameterEventResponse
{
    public bool Success { get; set; } = false;
}
```

12.1.8.3 Klasse: SaveParameterRequest

```
public class SaveParameterRequest
{
    public Parameter Parameter { get; set; }

    public SaveParameterRequest(Parameter parameter)
    {
        Parameter = parameter;
    }
}
```

12.1.8.4 Klasse: SaveParameterResponse

```
public class SaveParameterResponse
{
    public bool Success { get; set; } = false;
}
```

12.2 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.

12.3 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.

12.4 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.

12.5 Assembly: CMI.Manager.ExampleServiceA

12.5.1 Klasse: ExampleServiceA

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

12.5.1.1 Property: ParameterBus

```
private IBusControl ParameterBus { get; set; }
```

12.5.1.2 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();
}
```

12.5.1.3 Methode: Stop

```
public void Stop()
{
}
```

12.5.2 Klasse: ExampleSettingA

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

12.5.2.1 Property: Date

```
[Mandatory]
[Default("12.02.2017")]
public string Date;
```

12.5.2.2 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;
```

12.5.2.3 Property: ServiceOn

```
[Default("false")]
public bool ServiceOn;
NumberOfIterations
[Validation(@"[0-9]|[0-9][0-9]")]
[Mandatory]
public int NumberOfIterations;
```


12.6 Assembly: CMI.Manager.ExampleServiceB

12.6.1 Klasse: ExampleServiceB

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

12.6.1.1 Property: ParameterBus

Businstanz für die RabbitMQ Kommunikation.

```
private IBusControl ParameterBus { get; set; }
```

12.6.1.2 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();
}
```

12.6.1.3 Methode: Stop

```
public void Stop()  
{  
}
```

12.6.2 Klasse: ExampleSettingB

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

12.6.2.1 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;
```

12.6.2.2 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;
```

12.6.2.3 Property: FehlerVerstecken

```
[Default("true")]  
public bool FehlerVerstecken;
```

12.6.2.4 Property: Sekunden

```
[Validation(@"[0-9]|[0-5][0-9")]  
[Mandatory]  
public int Sekunden;
```

12.7 Assembly: CMI.Manager.Parameter

12.7.1 Klasse: GetParameterEventResponseConsumer

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

12.7.1.1 Methode: Consume

```
public async Task Consume(ConsumeContext<GetParameterEventResponse> context)
{
    ParameterRequestResponseHelper.Parameters.AddRange(context.Message.Parameters);
    await Console.Out.WriteLineAsync("Parameter Recived");
}
```

12.7.2 Klasse: GetParameterRequestConsumer

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

12.7.2.1 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");
}
```

12.7.3 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.

12.7.3.1 Property: Parameters

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

12.7.3.2 Property: SavedSuccessfully

```
public static bool? SavedSuccessfully { get; set; }
```

12.7.4 Klasse: ParameterService

Diese Klasse regelt das Startverhalten vom Parameterservice

12.7.4.1 Property: ParameterBus

Businstanz für die RabbitMQ Kommunikation.

```
public static IBusControl ParameterBus { get; set; }
```

12.7.4.2 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();
}
```

12.7.4.3 Methode: Stop

```
public void Stop()
{
    ParameterBus.Stop();
}
```

12.7.5 Klasse: SaveParameterEventResponseConsumer

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

12.7.5.1 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 occurred while
saving!");
    }
}
```

12.7.6 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.

12.7.6.1 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");
}
```

12.8 Assembly: CMI.Web.Management

12.8.1 Ordner: Controllers

12.8.1.1 Klasse: ParameterController

Diese Klasse enthält die API Controllers.

12.8.1.1.1 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);
}
```

12.8.1.1.2 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);
}
```


12.8.2 Ordner: Helpers

12.8.2.1 Klasse: BusHelper

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

12.8.2.1.1 Property: ParameterBus

Businstanz für die RabbitMQ Kommunikation.

```
public static IBusControl ParameterBus { get; }
```

12.8.2.1.2 Klasse: BusHelper

```
static BusHelper()  
{  
    ParameterBus = BusConfigurator.ConfigureBus((cfg, host) =>  
    {  
  
    });  
    ParameterBus.Start();  
}
```

12.9 Assembly: CMI.Contract.Parameter.Tests

12.9.1 Klasse: TestSetting

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

12.9.1.1 Property: TestFlag

```
[Default("false")]  
[Description("Some Test")]  
public bool TestFlag;
```

12.9.1.1.1 Property: TestMailAdress

```
[Default("max.muster@supermail.ch")]  
[Mandatory]  
[Validation(@"([a-zA-Z0-9_]+@[a-zA-Z0-9]+\.[a-zA-Z0-9-]+)")]  
public string TestMailAdress;
```

12.9.1.2 Property: TestDate

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

12.9.1.3 Property: TestNumber

```
[Description("Testwert")]
public int TestNumber;
```

12.9.1.4 Property: TestUnknownType

```
public char TestUnknownType;
```

12.9.1.5 Property: TestMandatory

```
[Mandatory]
public string TestMandatory;
```

12.9.2 ParameterSerializerTests

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

12.9.2.1 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);
}
```

12.9.2.2 Methode: Name_can_be_serialized_correctly

```
[TestMethod]
public void Name_can_be_serialized_correctly()
{
    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.TestMailAddress"));
    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"));
}
```

12.9.2.3 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("TestMailAddress") && 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"));
}
```

12.9.2.4 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("TestMailAddress") && p.Default ==
    "max.muster@supermail.ch"));
    Assert.AreEqual(pl.Count(p => p.Default != null), 2);
    Assert.AreEqual(pl.Count(p => p.Default == null), 4);
}
```

12.9.2.5 Methode: Value_can_be_serialized_correctly

```
[TestMethod]
public void Value_can_be_serialized_correctly()
{
    var testSetting = new TestSetting
    {
        TestDate = "03.04.2018",
        TestFlag = true,
        TestMailAddress = "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("TestMailAddress") && 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()));
}
```

12.9.2.6 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);
}
```

12.9.2.7 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("TestMailAddress") &&
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);
}
```

12.9.2.8 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);
}
```

12.9.2.9 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);
}
```

12.9.2.10 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,
        TestMailAddress = "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.TestMailAddress,
testSetting.TestMailAddress);
        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);
        }
    }
}
```

12.9.3 Klasse: ParameterValidationTests

Diese Klasse testet die Parametervalidation.

12.9.3.1 Methode: ParameterList_Should_be_valid

```
[TestMethod]
public void ParameterList_Should_be_valid()
{
    var testSetting = new TestSetting
    {
        TestDate = "03.04.2018",
        TestFlag = true,
        TestMailAddress = "testmail@mail.ch",
        TestNumber = 20,
        TestMandatory = "Test"
    };
    var parameterList =
ParameterHelper.GetParameterListFromSetting(testSetting);
    Assert.IsTrue(ParameterHelper.ValidateParameter(parameterList));
}
```

12.9.3.2 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"))));
}
```


12.9.3.3 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"))));
}
```

12.9.3.4 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"))));
}
```

12.9.3.5 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"))));
}
```

12.9.3.6 Methode: Empty_mandatory_value_with_regex_defined_should_not_be_valide

```
[TestMethod]
public void Empty_mandatory_value_with_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("TestMailAddress"))));
}
```

12.9.3.7 Methode: None_empty_mandatory_conform_value_with_regex_defined_is_valide

```
[TestMethod]
public void None_empty_mandatory_conform_value_with_regex_defined_is_valide()
{
    var testSetting = new TestSetting();
    testSetting.TestMailAddress = "max.muster@supermail.ch";
    var parameterList =
ParameterHelper.GetParameterListFromSetting(testSetting);
    Assert.IsTrue(ParameterHelper.ValidateParameter(parameterList.First(p =>
p.Name.EndsWith("TestMailAddress"))));
}
```

12.9.3.8 Methode: None_empty_unconform_mandatory_value_with_regex_defined_is_not_valide

```
[TestMethod]
public void
None_empty_unconform_mandatory_value_with_regex_defined_is_not_valide()
{
    var testSetting = new TestSetting();
    testSetting.TestMailAddress = "This is not a mail address!";
    var parameterList =
ParameterHelper.GetParameterListFromSetting(testSetting);
    Assert.IsFalse(ParameterHelper.ValidateParameter(parameterList.First(p =>
p.Name.EndsWith("TestMailAddress"))));
}
```

13 Code Dokumentation Frontend

13.1 Ordner: services

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

13.1.1 Klasse: `HttpService`

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

13.1.2 Klasse: `ParameterService`

Dieser Service ist für die API Aufrufe der Parameterverwaltung zuständig.

13.1.2.1 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();  
}
```

13.1.2.2 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();  
}
```

13.1.2.3 Methode: _createUrl

Diese Methode stellt die URL der Webseite zur Verfügung. Diese Methode gibt es so auch im Projekt Viaduc. Sie wurde nur leicht abgeändert, um die URL des IPA Systems zurückzugeben.

```
private _createUrl(): 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;  
}
```

13.2 Ordner: highlight

13.2.1 Klasse: HighlightComponent

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

13.2.1.1 Property: highlight

Der Text, welcher hervorgehoben werden soll.

```
@Input()  
public highlight: string;
```

13.2.1.2 Property: text

Der Text, der angezeigt werden soll.

```
@Input()  
public text: string;
```

13.2.1.3 Methode: constructor

```
public constructor() {}
```

13.2.1.4 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;
}
```

13.2.2 HTML: HighlightComponent

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

</span>
```

13.3 Ordner: parameterManager

13.3.1 Klasse: Parameter

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

13.3.1.1 Property: name

```
name: string;
```

13.3.1.2 Property: value

```
value: string;
```

13.3.1.3 Property: type

```
type: string;
```

13.3.1.4 Property: mandatory

```
mandatory: boolean;
```

13.3.1.5 Property: description

```
description: string;
```

13.3.1.6 Property: regexValidation

```
regexValidation: string;
```

13.3.1.7 Property: default

```
default: string;
```

13.4 Ordner: parameter

13.4.1 Klasse: ParameterComponent

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

13.4.1.1 Property: parameter

Diese Property ist das Parameterobjekt.

```
@Input()  
public parameter: Parameter;
```

13.4.1.2 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>();
```

13.4.1.3 Property: searchString

Dies ist der Suchtext vom Suchfeld.

```
@Input()  
public searchString: string;
```

13.4.1.4 Property: active

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

```
public active: boolean = false;
```

13.4.1.5 Property: value

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

```
public value: string;
```

13.4.1.6 Property: checked

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

```
public checked: boolean;
```

13.4.1.7 Property: validationError

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

```
public validationError: boolean;
```

13.4.1.8 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();
```

13.4.1.9 Methode: constructor

```
constructor (private _paramService: ParameterService) {  
}
```


13.4.1.10 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(() => {
        if (this.value) {
            this.validationError = !this._validateString(this.value);
        } else {
            this.validationError = !this._isValid();
        }
    });
    this.value = this.parameter.value;
    this.checked = this.parameter.value === 'True';
}
```

13.4.1.11 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;
    }
}
```

13.4.1.12 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;
}
```

13.4.1.13 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);
    }
}
```

13.4.1.14 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;
}
```

13.4.1.15 Methode: _isValid

Diese Methode validiert den Parameter.

```
private _isValid(): boolean {  
    return this._validateString(this.parameter.value);  
}
```

13.4.1.16 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;  
    }  
}
```

13.4.1.17 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';  
}
```

13.4.1.18 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';  
}
```

13.4.2 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>
```

13.4.3 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;  
  }  
}
```

13.5 Ordner: ParameterList

13.5.1 Klasse: ParameterListComponent

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

13.5.1.1 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;
```

13.5.1.2 Property: filteredParameters

Diese Property enthält alle momentan angezeigten Parameter.

```
public filteredParameters: Parameter[] = [];
```

13.5.1.3 Property: _allParameters

Diese Property enthält alle vom Server geladenen Parameter.

```
private _allParameters: Parameter[] = [];
```

13.5.1.4 Property: validationEvent

Dies ist der Validierungsevent.

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

13.5.1.5 Property: searchString

Dies ist das Inputbindig des Suchfelds.

```
public searchString: string = '';
```

13.5.1.6 Property: searchedStringUpToDate

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

```
public searchedStringUpToDate: boolean;
```

13.5.1.7 Methode: constructor

Im Konstruktor werden alle Parameter geladen.

```
constructor(private _params: ParameterService) {  
    this.getAllParameters();  
}
```

13.5.1.8 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;  
    });  
}
```

13.5.1.9 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;  
    }  
}
```


13.5.1.10 Methode: emitValidationEvent

Dies ist der Eventauslöser für die Validierung.

```
public emitValidationEvent() {  
    this.validationEvent.emit();  
}
```

13.5.1.11 Methode: searchParam

Diese Methode filtert die Parameter nach der Eingabe im Suchfeld. Name und Value des Parameters werden 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;  
    }  
}
```

13.5.2 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>
```

13.5.3 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%;  
    }  
}
```

14 Schlusswort

14.1 Ausblick auf nach der Arbeit

In der Woche nach Abschluss der Arbeit, muss ich das Ganze noch ins Livesystem einbinden. Während diesem Schritt könnte man noch die Gruppierung nach Services einbauen, die zwar geplant war aber als Kann-Anforderung keinen Platz mehr fand. Etwas anderes, was man noch einbauen könnte, wäre eine «Speichern erfolgreich» Meldung. Die IPA könnte auch als Grundlage für eine zentrale Verwaltungsmöglichkeit für das CMIAXIOMA und CMISTAR sein. Dies sind unsere Geschäfts- und Protokollverwaltungslösungen respektive unsere Archivlösungen. Wir haben vor diese von dem einen grossen Service, den wir momentan noch haben, in mehrere kleine aufzuteilen. Da können die gesammelten Erfahrungen dieser IPA sicherlich wieder zum Einsatz kommen.

14.2 Reflexion

Ich habe die IPA mit grosser Nervosität angefangen. Diese Nervosität konnte ich aber schnell ablegen und mich voll und ganz auf die Arbeit konzentrieren.

Das Dokumentieren ging besser als erwartet. Ich rechnete damit, dass ich grosse Mühe haben werde mit der Dokumentation, denn ich war bisher in Schulprojekten eher gut im Programmieren und eher weniger gut im Dokumentieren. Durch das IPERKA Vorgehensmodell konnte ich mich während der Planung- und Entscheidungsphasen wirklich auf das Planen und Entscheiden konzentrieren. Ich habe mich an einigen Orten während dieser Phase zurückerinnern können an das erste und zweite Lehrjahr, in denen ich die verschiedensten Diagramme kennengelernt habe. Dies half mir sehr einen Anhaltspunkt für das Ganze zu finden. In der Lehre konnte ich oftmals konzeptionell Probleme analysieren. Ich denke, dass mir dies enorm half um die grosse Aufgabe gut zu analysieren und zu zerlegen.

Wegen des Programmierens habe ich überhaupt diese Lehre angefangen. Deshalb verwundert es wohl nicht gross, dass mir das Programmieren am meisten Spass gemacht hat. Ich glaube mir ist der generische Ansatz gut gelungen in der Umsetzung. Ich konnte hier auf das Wissen zurückgreifen, welches ich während knapp vier Jahren bei der CM Informatik AG und der Greenshare AG aufgebaut habe. Das Programmieren der Reflection in der ParameterHelper Klasse habe ich sehr spannend gefunden. Die Zeit ist während der Realisierungsphase gefühlt extrem schnell vorbeigegangen.

Das Korrigieren und Auswerten zum Schluss zog sich noch ein wenig. Durch das Gegenlesen sind jeweils einige Rechtschreibfehler zum Vorschein gekommen.

Ich denke diese Arbeit widerspiegelt gut, was ich in den letzten Jahren gelernt habe. Während dem Schreiben dieser Arbeit konnte ich viel Gelerntes nochmals vertiefen. Ich konnte viel neues Wissen über die Funktionsweise von RabbitMQ, Masstransit und vielen anderen Aspekten vom Viaduc Projekt gewinnen. Ich bin da hauptsächlich im Frontend unterwegs, wo ich selbst meine Stärke sehe. Es war jedoch ein sehr spannender Einblick ins Backend und die Problemstellungen, die sich da stellen. Ich gebe die Arbeit mit verschiedene Emotionen ab. Ich bin froh, dass diese Prüfung endlich vorbei ist und ich bald das EFZ Zeugnis in den Händen halten darf, schaue aber auch zurück und sehe all die schönen Erfahrungen, die ich während der Lehre gemacht habe. Ich freue mich aber auf den noch vor mir stehenden Berufsalltag und alles was dieser mit sich bringen mag.

14.3 Danksagung

Besonderen Dank gilt Martin Tinner, welcher mich mit vollem Engagement durch die IPA begleitet hat. Melanie Müller, für das Gegenlesen der Dokumentation und der Bereitschaft mir bei jedem Wordproblem zu helfen. Matthias Hess und Benjamin Schäublin, welche mir die letzten Jahre viel über das Programmieren beibrachten und ich während der IPA bei Fragen angehen durfte. Zuletzt Doris Fux, sie hat mir durch die Lehre hindurch gezeigt, dass der Informatiker-Beruf nicht nur programmieren ist, sondern auch noch viel Interesse für die Materie hinter dem Programm braucht.

15 Verzeichnisse

15.1 Glossar

Begriff	Erklärung
Angular	Clientframework für «Single Page Applikationen».
API	Serverseitige Funktion.
Backend	Serverteil des Programms, ist für die Datenverarbeitung im Hintergrund verantwortlich
Bootstrap	CSS Framework
Branch	Einen Stand des Source Codes, der unabhängig vom restlichen Entwicklungsstand ist zur Implementierung eines Features ohne Einfluss von aussen.
Camel-Case	Eine Möglichkeit etwas zu benennen. Dieser Satz steht in Camel Case
CMIAxioma	Geschäfts- und Protokollverwaltungslösung der CM Informatik AG.
CMiSTAR	Archivlösung der CM Informatik AG.
Commit	Änderungen ins Git übernehmen.
Contributer	Englisch der Beiträger. Im Kontext von Git / Github ein Entwickler
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.
E2E Tests	End-to-End Tests, womit das Frontend getestet werden kann.
Flag	Ein Ja/Nein Wert
Framework	Eine Programmierhilfe mit gewisser Funktionalität, sodass das Grundgerüst nicht selbst geschrieben werden muss.
Frontend	Benutzeroberfläche und alles, was dafür berechnet werden muss. Interagiert mit dem Backend
Git / Github	Git ist ein Versionsverwaltungssystem und Github ist ein Cloud-Speicherort dafür.
GUI	Grafical User Interface, die Benutzerüberfläche
Inputbindig	Das binden einer Variable an ein Inputfeld
Lower-Camel-Case	Gleich wie Camel-Case nur kleiner Buchstabe am Anfang
MassTransit	C#.Net Anbindung für RabbitMQ
Mergen	Zusammenführen von 2 Branchen
Message Queue	Ein Stapel von Nachrichten von einem Service an den anderen.
Mouseover	GUI Event beim drüberfahren der Maus über das Objekt.
officeatwork	Tool für die Vorlagenverwaltung
OnFocus	GUI Event beim Anklicken / Anspringen mit Tab eines Inputelements.
Overload	Gleiche Methode mit verschiedenen Übergabewerten

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.
Prefix	Ein voranstehender Text.
Queue	Englisch für «Reihe». Eine Kommunikation-Queues im Kontext dieser Arbeit ist ein «Stapel» voller Anweisungen von RabbitMQ.
RabbitMQ	Message Queue Tool zur Kommunikation der Microservices
Regex	Abkürzung für Regular Expression, ist ein Muster um etwas in einem Text zu finden.
Repository	Repository ist ein Github Projekt
Scrum	Agile Projektmethode
Snake Case	Dieser_satz_ist_in_snake_case_geschrieben
Traffic	Datenverkehr zwischen Frontend und Backend
Visual Studio	Programmierungsumgebung für C# in diesem Projekt
Webstorm	Programmierungsumgebung für TypeScript, HTML und Less in diesem Projekt

Tabelle 89 - Glossar

15.2 Tabellenverzeichnis

Tabelle 1 – Dokumentablage	15
Tabelle 2 - Source Code Ordnerstruktur	16
Tabelle 3 - Konventionen Dokumentation	18
Tabelle 4 - Code-Konventionen C#	18
Tabelle 5 - Code Konventionen TypeScript.....	18
Tabelle 6 - Layout-Konventionen	19
Tabelle 7 - Unittests-Konventionen	19
Tabelle 8 - Systemgrenze Erklärungen Diagramm.....	21
Tabelle 9 - Geplante Tagesziele Tag 1	23
Tabelle 10 - Tätigkeiten Tag 1.....	23
Tabelle 11 - Probleme Tag 1.....	23
Tabelle 12 - Hilfestellung Tag 1.....	23
Tabelle 13 - Tagesreflexion Tag 1.....	23
Tabelle 14 - Geplante Tagesziele Tag 2	24
Tabelle 15 - Tätigkeiten Tag 2.....	24
Tabelle 16 - Probleme Tag 2.....	24
Tabelle 17 - Hilfestellung Tag 2.....	24
Tabelle 18 - Tagesreflexion Tag 2.....	24
Tabelle 19 - Geplante Tagesziele Tag 3	25
Tabelle 20 - Tätigkeiten Tag 3.....	25
Tabelle 21 - Probleme Tag 3.....	25
Tabelle 22 - Hilfestellung Tag 3.....	25
Tabelle 23 - Tagesreflexion Tag 3.....	25
Tabelle 24 - Geplante Tagesziele Tag 4	26
Tabelle 25 - Tätigkeiten Tag 4.....	26
Tabelle 26 - Probleme Tag 4.....	26
Tabelle 27 - Hilfestellung Tag 4.....	26
Tabelle 28 - Tagesreflexion Tag 4.....	26
Tabelle 29 - Geplante Tagesziele Tag 5	27
Tabelle 30 - Tätigkeiten Tag 5.....	27
Tabelle 31 - Probleme Tag 5.....	27
Tabelle 32 - Hilfestellung Tag 5.....	27
Tabelle 33 - Tagesreflexion Tag 5.....	27
Tabelle 34 - Geplante Tagesziele Tag 6	28
Tabelle 35 - Tätigkeiten Tag 6.....	28
Tabelle 36 - Probleme Tag 6.....	28
Tabelle 37 - Hilfestellung Tag 6.....	28
Tabelle 38 - Tagesreflexion Tag 6.....	28
Tabelle 39 - Geplante Tagesziele Tag 7	29
Tabelle 40 - Tätigkeiten Tag 7.....	29
Tabelle 41 - Probleme Tag 7.....	29
Tabelle 42 - Hilfestellung Tag 7.....	29
Tabelle 43 - Tagesreflexion Tag 7.....	29
Tabelle 44 - Geplante Tagesziele Tag 8	30
Tabelle 45 - Tätigkeiten Tag 8.....	30
Tabelle 46 - Probleme Tag 8.....	30
Tabelle 47 - Hilfestellung Tag 8.....	30
Tabelle 48 - Tagesreflexion Tag 8.....	30
Tabelle 49 - Geplante Tagesziele Tag 9	31

Tabelle 50 - Tätigkeiten Tag 9.....	31
Tabelle 51 - Probleme Tag 9.....	31
Tabelle 52 - Hilfestellung Tag 9.....	31
Tabelle 53 - Tagesreflexion Tag 9.....	31
Tabelle 54 - Geplante Tagesziele Tag 10	32
Tabelle 55 - Tätigkeiten Tag 10.....	32
Tabelle 56 - Probleme Tag 10.....	32
Tabelle 57 - Hilfestellung Tag 10.....	32
Tabelle 58 - Tagesreflexion Tag 10.....	32
Tabelle 59 - Tagesjournale gesehen.....	33
Tabelle 60 - Getting Started Dokumentation Ziele	35
Tabelle 61 - Getting Started Dokumentation Anforderungen.....	35
Tabelle 62 - Anzeige & Speichern der Parameter Ziele	35
Tabelle 63 - Anzeige & Speichern der Parameter Anforderungen	35
Tabelle 64 - Implementation des Validierungsmechanismus Ziele.....	36
Tabelle 65 - Implementation des Validierungsmechanismus Anforderungen.....	36
Tabelle 66 - Suchen eines Parameters Ziele	36
Tabelle 67 - Suchen eines Parameters Anforderungen.....	36
Tabelle 68 - Erstellung der Unit Tests und des Testkonzepts Anforderungen.....	36
Tabelle 69 - Durchführen der Tests Ziele	37
Tabelle 70 - Durchführen der Tests Anforderungen	37
Tabelle 71 - Umsetzungsreihenfolge.....	41
Tabelle 72 - Lösungsmatrix Unittests	41
Tabelle 73 - Lösungsmatrix Integration Tests / e2e Tests.....	42
Tabelle 74 - Lösungsmatrix User-Testing.....	42
Tabelle 75 - Lösungsmatrix Speichern der Parameter als Parametertyp im Json.....	43
Tabelle 76 - Lösungsmatrix Speichern der Parameter als generischer Typ im Json.....	43
Tabelle 77 - Lösungsmatrix nur serverseitig im Service den Parameter selbst testen lassen	48
Tabelle 78 - Lösungsmatrix Regular Expression im Parameter	48
Tabelle 79 - Lösungsmatrix Volltextsuche über alles	49
Tabelle 80 - Lösungsmatrix Suche auf die Parameternamen.....	49
Tabelle 81 - Lösungsmatrix Suche auf Parameternamen und Parameterwert	50
Tabelle 82 - Testfälle.....	52
Tabelle 83 - Aussagen ParameterSerializer Tests	58
Tabelle 84 - Aussage ParameterValidation Tests	59
Tabelle 85 - Resultate Testdurchgang 1	63
Tabelle 86 - Resultate Testdurchgang 2	64
Tabelle 87 - Resultate Testdurchgang 3	65
Tabelle 88 - Resultate Testdurchgang 4	65
Tabelle 89 - Glossar	118
Tabelle 90 – Quellenverzeichnis	122

15.3 Bildverzeichnis

Abbildung 1 - Struktur der Dokumentablage	15
Abbildung 2 - Frontend Ordnerstruktur	16
Abbildung 3 - Backend Ordnerstruktur	16
Abbildung 4 - IPERKA	19
Abbildung 5 - Umsysteme	20
Abbildung 6 - Systemgrenzen	21
Abbildung 7 - RabbitMQ Event Kommunikation	22
Abbildung 8 - Use-Case Diagramm	34
Abbildung 9 - Firefox/Waterfox Einstellungen	44
Abbildung 10 - Chrome Einstellungen	45
Abbildung 11 - Visual Studio Einstellungen	46
Abbildung 12 - Mockup	47
Abbildung 13 - Screenshot umgesetzte Lösung Anzeige & Speichern der Parameter	53
Abbildung 14 - Sequenzdiagramm Servicekommunikation beim «Save»	54
Abbildung 15 – Screenshot umgesetzte Lösung Implementation des Validierungsmechanismus	56
Abbildung 16 - Screenshot umgesetzte Lösung Suche eines Parameters	57
Abbildung 17 - Screenshot Getting Started	62
Abbildung 18 - Screenshot Getting Started Validierung	62

15.4 Quellenverzeichnis

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

Tabelle 90 – Quellenverzeichnis