

Reporting in a Microservice Architecture

Konzept für die Erstellung von Berichten in einer Microservice Architektur

Bachelorarbeit

Studiengang Informatik

Zürcher Hochschule für angewandte Wissenschaften

eingereicht bei

Beat Seeliger, Fachdozent für Software-Entwicklung

Oktober 2015

Eingereicht von
Severin Andrew Müller
Hauptstrasse 39
CH-8224 Löhningen SH
muelles5@students.zhaw.ch

Inhaltsverzeichnis

1 Einleitung	7
1.1 Ausgangslage.....	7
1.2 Ziele der Arbeit.....	8
1.3 Aufgabenstellung	8
1.3.1 Ist-Analyse	8
1.3.2 Anforderungsanalyse	8
1.3.3 Recherche	8
1.3.4 Konzept	8
1.3.5 Proof of Concept	8
1.3.6 Testing.....	8
1.3.7 Abschluss.....	8
2 Ist-Analyse	9
2.1 Einleitung	9
2.1.1 Reports in der Applikation r5.....	10
2.1.2 Reports auf der MRE	11
2.1.3 Reports auf der Web-Plattform	13
2.2 Datenquellen.....	14
2.3 Datenstrukturen.....	14
2.4 Report Templates	14
2.5 Schwächen	14
3 Anforderungsanalyse.....	15
3.1 Stakeholder	15
3.2 Visualisierung der User Stories	15
3.3 User Stories	15
3.3.1 Allgemeines.....	15
3.4.2 Automatisierte Reports.....	16
3.4.3 Report-Eigenschaften	17
3.4.4 Datensicherheit.....	18
3.4.5 Administrator Stories	18

3.5 Funktionale Anforderungen.....	19
3.5.1 Anforderungen Allgemeines	19
3.5.2 Anforderungen automatisierte Reports	20
3.5.3 Anforderungen Report Eigenschaften	23
3.6 Nicht-Funktionale Anforderungen	23
3.6.1 Performance.....	23
3.6.2 Verfügbarkeit	24
3.6.3 Look and Feel	25
3.7 Technische Anforderungen	25
3.7.1 Daten.....	25
3.7.2 Scheduler	25
3.7.3 Logging	26
4 Recherche	27
4.1 Benötigte Werkzeuge	27
4.1.1 Scheduling	27
4.1.2 Rendering	27
4.1.3 Mail-Versand	27
4.1.4 Dokumente erstellen	27
4.2 Evaluation der Werkzeuge.....	27
4.2.1 Scheduler	27
4.2.2 Rendering	30
4.2.3 Output Format	31
4.2.4 Mailer	31
5. Design	33
5.1 Benutzeroberfläche	33
5.1.1 Navigation	33
5.1.2 Header und Suche	34
5.1.3 Anzeige.....	34
5.1.4 Report ausführen	35
5.1.5 Job-Details.....	36
5.2 Benutzeroberfläche Administrator	36
5.2.1 Header, Navigation und Suche	36
5.2.2 Jobs.....	36

5.3 Datenbank Reporting	37
5.4 Scheduler	37
6 Konzept.....	38
6.1 Variante A: Multiple Data Sources in Report.....	38
6.1.1 Ablauf Variante A	39
6.1.2 Daten für Variante A	39
6.1.3 Report Parameter	41
6.2 Variante B: Dataset populated dynamically	42
6.2.1 Ablauf Variante B	43
6.2.2 Daten für Variante B	43
6.2.3 Report-Parameter	44
6.3 Variante C: Data Warehouse	45
6.3.1 Ablauf Variante C	46
6.3.2 Daten Variante C	46
6.3.3 Report Parameter Variante C.....	46
6.4 Zusammenfassung.....	46
6.5 Nutzwert-Analyse Varianten A, B und C.....	47
6.5.1 Bewertungsschema.....	47
6.5.2 Datenkomplexität	47
6.5.3 Datensicherheit.....	48
6.5.4 Report Parameter	48
6.5.5 Spezielle Daten.....	49
6.5.6 Redundanz	49
6.5.7 Performance.....	49
6.5.8 Bestehendes System	50
6.5.8 Entscheidungstabelle	50
6.6 Weitere Analyse Variante B.....	51
6.6.1 Queue.....	51
6.6.2 Datenaufbereitung.....	51
6.6.3 Sequenzdiagramm	52
6.6.4 Klassendiagramm	53
7 Proof of Concept.....	54

7.1 Bau eines Prototypen	54
7.1.1 Einleitung	54
7.1.2 Abgedeckte Anforderungen.....	54
7.1.3 Bereitstellen der Umgebung.....	54
7.1.4 Bereitstellen des Reporting-Microservice	56
7.1.5 Vorbereitung DataCollectionObject.....	57
7.1.6 Report Konfiguration	58
7.1.7 Messaging	59
7.1.8 Vorbereitung Report-Template für den Prototypen	64
7.1.9 Ausführen eines Reports.....	66
7.2 Zusammenfassung Proof of Concept.....	68
7.2.1 Anforderungen.....	68
7.2.2 Weiteres.....	68
8. Ausblick.....	69
8.1 Komplexere Daten.....	69
8.2 Weiterentwicklung System.....	69
8.2.1 Umgebung.....	69
8.2.2 UI	70
8.2.3 Datenbank.....	70
9 Testing	71
9.1 Testkonzept.....	71
9.2 Unit Tests	71
9.2.1 GivenWhenThen Prinzip	71
9.2.2 Vorbereitung des Tests	72
9.2.3 Simulieren von benötigten Daten.....	72
9.2.4 Erstellen des Unit Tests.....	73
9.3 Integrationstest.....	73
9.4 Regression Test	73
9.5 Acceptance Test	74
10 Fazit und Erkenntnisse	76
11 Anhang.....	77
11.1 Anhang A: Bilderverzeichnis	77

11.2 Anhang B: Listings.....	78
11.3 Anhang C: Tabellenverzeichnis	79
11.4 Anhang D: Abkürzungsverzeichnis	79
11.5 Anhang E: Quellen- und Literaturverzeichnis	79

1 Einleitung

1.1 Ausgangslage

Wir leben in einer Zeit, in der Information von enorm grosser Bedeutung ist. Seit das Internet seinen Siegeszug Mitte der 1990er Jahre begonnen hatte, ist die weltweite Datenmenge enorm gewachsen. Aus einer grossen Flut von Daten die nützlichen Informationen empfängergerecht und anschaulich zu repräsentieren kann sich schwierig gestalten.

Mit Berichten, sogenannten Reports, kann man Informationen rasch und einfach bereitstellen. Solche Berichte können in unterschiedlichen Formen auftreten – von einfachen Datenauszügen (-> *Dumps*) bis hin zu personalisierbaren Dashboards ist alles dabei.

Der Autor der vorliegenden Arbeit ist bei seinem Arbeitgeber unter anderem zuständig für die Erstellung und Wartung von Kundenberichten. Diese Kundenberichte sind einerseits in der Applikation, die der Arbeitgeber des Autors vertreibt, vorhanden und laufen andererseits auf einer eigenen Plattform.

Die Applikation „r5“ der MESPAS AG ist eine Komplettlösung für Reedereien, deren Schiffe und Lieferanten. Dieses Legacy System läuft seit über 10 Jahren auf einem Tomcat Server mit einer zentralen MySQL Datenbank. Fat Clients verbinden sich mit dem Zentralserver um damit zu arbeiten. Mit diesem Client hat der Kunde die Möglichkeit für fast jeden Bereich einen Report zu erstellen. In den meisten Fällen soll ein solcher Report die Information, die auf dem Screen zu sehen ist, in einer PDF-Datei ausgegeben werden. Im Einkaufsmodul (Procurement) werden zudem auch für alle benötigten Formulare wie Einkaufsanfrage, Bestellungen, etc. entsprechende Reports generiert. Dazu läuft auf demselben Server eine Open-Source Lösung von BIRT (Business Intelligence Report Tools). Auf Basis eines sogenannten Report-Templates werden die Daten aufbereitet und die Ausgabedatei generiert.

Zusätzlich zu den Reports in der Applikation r5 bietet die MESPAS AG eine entkoppelte Plattform an, die MESPAS Reporting Engine (MRE). Diese ermöglicht es den Kunden kompliziertere Reports unabhängig von der Applikation zu generieren und beinhalten meist komplexere Aggregationen als jede in der Applikation selbst. Zudem können diese MRE-Reports auch terminiert werden. Das heisst, dass ein Benutzer sich einen Report regelmässig ausführen und per E-Mail schicken lassen kann. Die Implementierung der Reports folgt dem gleichen Muster (Report Templates mit BIRT).

Seit 2014 ist eine neue, web-basierte Lösung der Applikation r5 in Form einer Microservice Architektur in Entwicklung. Jede Tätigkeit läuft in einem eigenen Microservice in einer Datenbank mit entsprechenden zugehörigen Schemas. Nun soll für diese neue Plattform ein Konzept für die Integration von Kunden-Reports entwickelt werden. Diese soll ebenfalls als Microservice implementiert werden. Mit dieser neuen Lösung soll auch die MRE abgelöst werden können. Die Herausforderung ist dabei in erster Linie die Datenaufbereitung. Da die Daten bisher in einer zentralen Datenbank verwaltet wurde mussten diese nur von dort geholt werden. Für die neue Datenbank muss ein Konzept geschaffen werden, dass diese Daten unter Berücksichtigung von Performance und Aspekten der Datensicherheit aus den verschiedenen Datenbanken geladen und für die Erstellung dieser Reports aufbereitet werden können. Zu beachten ist dabei, dass jeder Microservice bei Änderungen individuell "deploybar" ist. Deshalb soll die Definition des Reports beim zugehörigen Microservice liegen und der Reporting-Microservice muss bei solchen Änderungen die neue Definition verarbeiten können, ohne dass ein neues Deployment für den Reporting-Service nötig wird.

Der Kern der neuen Lösung ist dabei die Aufbereitung der Datenstruktur für die benötigten Reports. Das komplette System ist zu entwerfen, implementiert werden soll vorerst jedoch nur ein Prototyp im Sinne eines Proof of Concept.

1.2 Ziele der Arbeit

Das Ziel der Arbeit ist das fertige Konzept für die Integration der Reports. Dabei sollen zunächst mögliche Werkzeuge für das Rendering evaluiert werden (z.B. ob BIRT Templates weiterhin unterstützt werden können). Danach soll analysiert werden, wie mit einem Request in JSON Form alle notwendigen Daten aufbereitet und zurückgegeben werden können. Die neue Reporting-Lösung soll es zudem ermöglichen, dass die Reports terminiert und automatisiert per E-Mail versendet werden können.

Bis anhin wurden die Reports von Hand getestet. Es soll daher eine Möglichkeit für automatisierte Unit Tests geschaffen werden. Zuletzt soll anhand eines Prototyps / Proof of Concept die Lauffähigkeit dieses Konzepts dargelegt werden. Die zentrale Fragestellung ist dabei:

„Kann ein Report Daten aus mehreren Datenbanken so aufbereitet werden, dass ein Objekt mittels JSON serialisiert und die Daten vollständig sind?“

1.3 Aufgabenstellung

Aus der Ausgangslage und den Zielen der Arbeit ergibt sich folgende Aufgabenstellung:

1.3.1 Ist-Analyse

- Analyse der bisherigen Datenstrukturen
- Analyse der bisherigen Werkzeuge

1.3.2 Anforderungsanalyse

- Erarbeitung der Anforderungen an das neue Reporting-Konzept
- Anforderungen an das Datenmodell beschreiben

1.3.3 Recherche

- Evaluation der Werkzeuge durchführen
- Entscheidung für ein Tool treffen

1.3.4 Konzept

- Definition des Request-Formates
- Beschreibung der Schnittstellen
- Beschreibung der Datenaufbereitung
- Beschreibung des Test-Konzeptes

1.3.5 Proof of Concept

- Prototyp des Microservice entwickeln
- Proof of Concept anhand Beispiel-Reports beschreiben

1.3.6 Testing

- Definition von Testfällen
- Durchführung der Tests

1.3.7 Abschluss

- Fazit und Erkenntnisse

2 Ist-Analyse

Bevor mit der Erarbeitung der neuen Lösung begonnen werden kann muss zunächst analysiert werden, wie das Aktuelle System aussieht um Synergien allenfalls nutzen zu können.

2.1 Einleitung

Die bisherige Lösung steht auf drei Säulen: die Reports in der Applikation r5, die Reports auf der MRE und die Reports auf der neuen Web-Plattform. Sämtliche Reports werden mit BIRT (Business Intelligence Reporting Tools) erstellt.

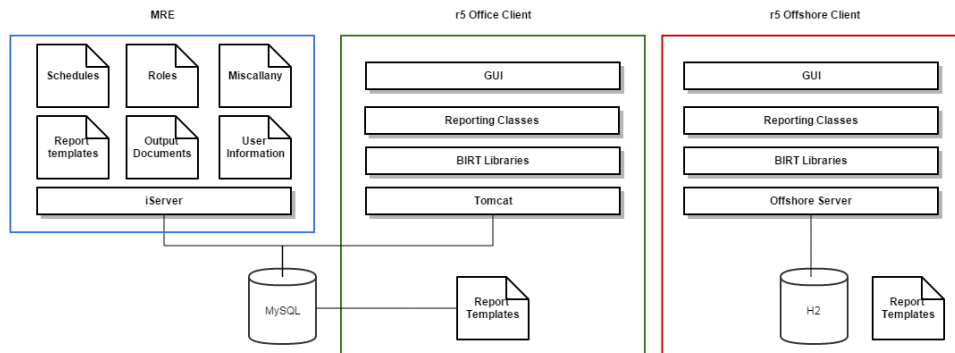


Abb. 1: Bestehende Lösung – Übersicht

Auf Abb. 1 wird eine Übersicht über die Reporting-Architektur in r5 und MRE gezeigt. Dabei fällt auf, dass die MRE und der r5 Office Client dieselbe Datenbank verwenden. Der r5 Offshore Client wird auf Schiffen installiert. Da Schiffe über keine permanente Internetverbindung verfügen, wird hier eine lokale H2 Datenbank verwendet. Die Daten werden dann jeweils über einen Synchronisationsmechanismus ausgetauscht.

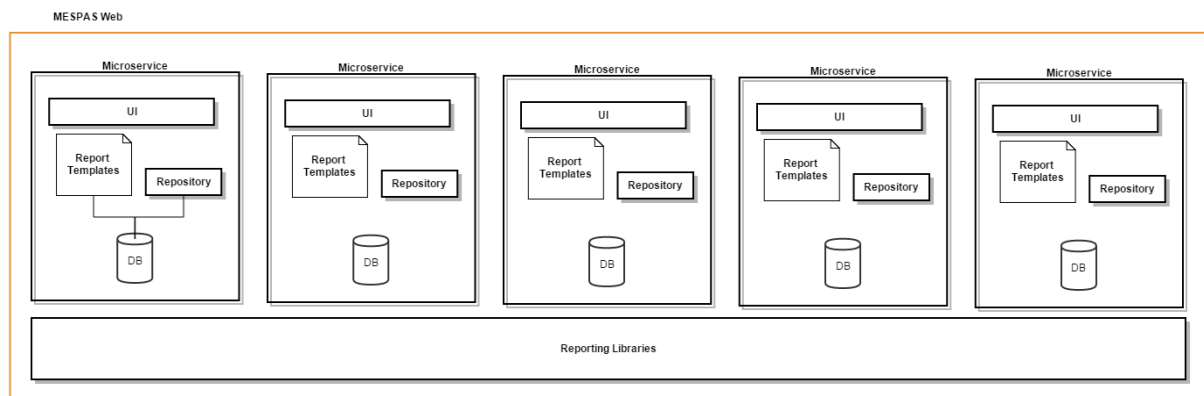


Abb. 2: Reports in MESPAS Web (vereinfacht)

Abb. 3 zeigt schematisch, wie die Erstellung eines Reports grundsätzlich abläuft.

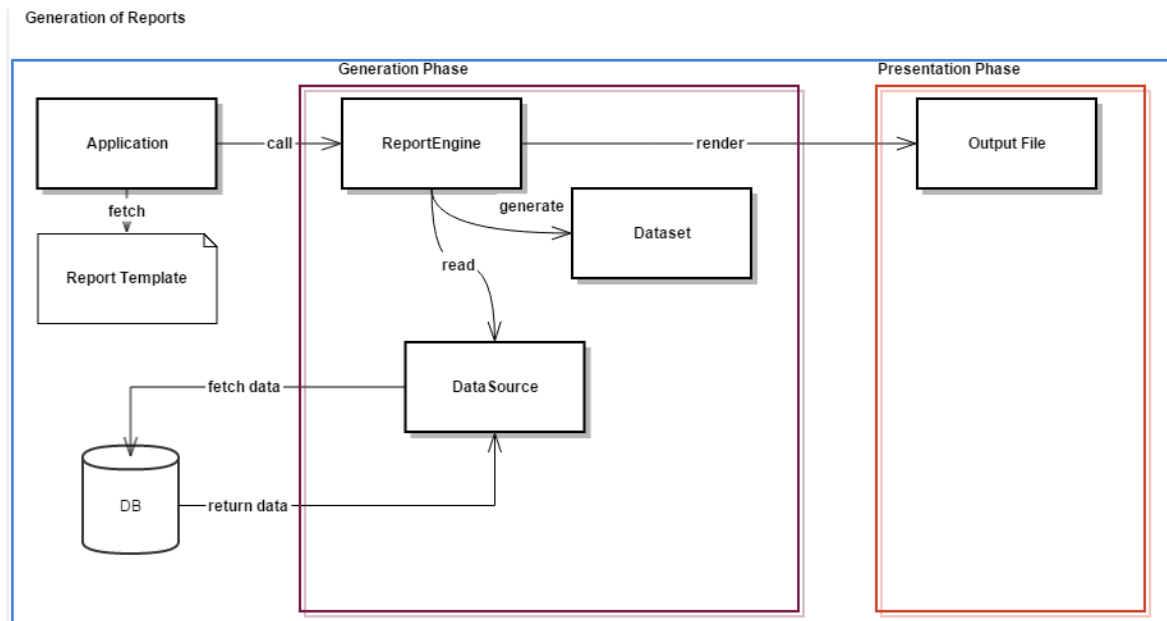


Abb. 3: Generierung von Reports

2.1.1 Reports in der Applikation r5

In der Applikation r5 befinden sich Reports die in der Regel die Informationen, die sich auf dem Screen befinden, für die selektierten Elemente abbilden sollen. Das folgende Beispiel soll dieses Prinzip näher erläutern. Im r5 gibt es den Screen „Task List“. Dieser Liste bildet ab, welche Wartungsarbeiten auf einem Schiff erledigt wurden oder zu erledigen sind:

Task List											
Vessel	Is Ar...	Doc	Offic...	Dry Do...	Task State	Last Comment Date	Date Done	YTN	Task No.	Creation Date	Target Date
T-CROWN	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	Pending	21.09.2011			V-CRO-1	09.05.2008	28.09.2008
T-PANTAU - stage	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	Not finished	09.10.2008	25.03.2008	12...	V-V2GR-1	20.02.2008	22.02.2008
	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	Finished	09.10.2012	09.10.2012		V-REF-1	21.09.2009	28.09.2009
T-MADEIRA - stage	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	Finished	12.05.2010	12.05.2010		V-MADEIRA-1	12.05.2010	13.05.2010
	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	Pending	15.04.2008			V-REF-2	04.11.2007	01.11.2008
T-PANTAU - stage	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	Postponed	18.07.2008			V-V2GR-2	25.03.2008	28.03.2008
	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	Pending				V-REF-3	22.11.2011	26.11.2011
	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	Finished	17.06.2015	17.06.2015		O-ATH-4	31.10.2007	03.11.2007
	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	Pending				V-REF-4	22.11.2011	27.11.2011
	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	Pending	23.09.2009			V-REF-4	23.09.2009	30.09.2009
T-ATHENS	<input type="checkbox"/>		<input type="checkbox"/>	<input checked="" type="checkbox"/>	Not finished		26.11.2007		V-ATH-5	26.11.2007	25.11.2007
	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	Pending	20.04.2015			V-REF-5	10.11.2011	17.11.2011
T-ATHENS	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	Not finished		20.02.2008		V-ATH-8	20.02.2008	31.03.2008
T-ATHENS	<input type="checkbox"/>		<input type="checkbox"/>	<input checked="" type="checkbox"/>	Pending				V-ATH-9	08.05.2008	16.05.2008
	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	Pending	13.05.2008			O-REF-33	01.02.2008	29.02.2008

Abb. 4: Task Liste in r5

Nun kann man die Zeilen selektieren für die man den Report erhalten möchte und den Report mittels Klick auf den Button ausführen. Für die selektierte Zeile erhält man nun folgenden Report im PDF Format:

T-MADEIRA - stage (9187485)						
2nd Engineer						
No.	Title	State	Type	Author	Creation Date	Target Date
V-MADEIRA-1	dsfasdf	Finished	Planned Maintenance	Andy Spichtig	12-May-2010	13-May-2010

Abb. 5: Beispiel Report Task Liste

In einigen Fällen gibt es auf einem Screen einen Übersichts- und einen Detailreport. Ein Übersichtsreport bildet im Wesentlichen das ab, was auf dem Screen zu sehen ist während ein Detailreport genauere Information über die ausgewählte Zeile liefert.

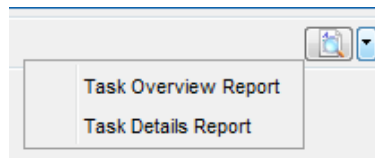


Abb. 6: Report-Button

Zusätzlich kann zwischen PDF, Excel, Word und HTML als Ausgabeformat gewählt werden.



Abb. 7: Ausgabeformat

2.1.2 Reports auf der MRE

Wie bereits erwähnt ist die MRE von der Applikation entkoppelt. Die Reports werden auf einem eigenen Server, genannt iServer betrieben. Dabei wird zwischen Standard Reports, die für alle Benutzer verfügbar sind, und kundenspezifischen Reports, die nur für Benutzer der entsprechenden Firma sichtbar sind, unterschieden.

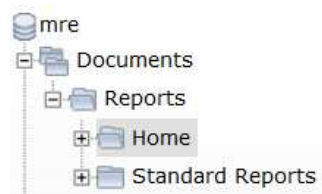


Abb. 8: Dokumentenstruktur in der MRE

MRE Reports haben eine andere Aufgabe als jene in der Applikation r5. Die MRE aggregiert und berechnet Zahlen und Fakten, die für das Zielpublikum von Interesse sind. Besonders im Einkaufsbereich, genannt „*Procurement*“, sind Übersichten über Bestellungen wie z.B. Volumen, oder Bestellungen nach Kategorie, etc. wichtig für die Reedereien. Auch im Bereich Wartung („*Planned Maintenance System*“, PMS) sind Reports die einen zusammenfassenden Charakter aufweisen von grossem Interesse.

Um einen Report auf der MRE auszuführen muss sich der Benutzer zunächst einloggen. Dann kann er über den Baum (siehe Abb. 5) zum entsprechenden Verzeichnis gelangen und den gewünschten Report ausführen. Beispiel:

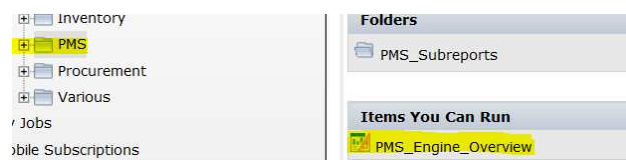


Abb. 9: Auswahl Beispiel-Report: Übersicht über die Wartungen der Motoren eines Schiffes

In diesem Beispiel erhält der Benutzer eine Übersicht über die Wartungsarbeiten die an den Motoren eines Schiffes durchgeführt wurden. Da dieser Report einen Parameter in Form einer Eingabe erwartet (in diesem Fall die Auswahl des Schiffes) wird der Benutzer danach gefragt:

Abb. 10: Auswahl eines erforderlichen Parameters für einen MRE Report

Das Resultat sieht wie folgt aus:

29-Jun-2015

Vessel Name:

Product:

BLACK PEARL - Live - 9304587

Main Engine RTA72U, Sulzer

8 RTA72U, 23920 kW @ 97

Report shows data of (last sync):

Total Running hours:

15-Jun-2015

60500 on 31-May-2015

Activity	Description (Part)	Interval	Unit	Serial Number	Last Overhaul	RH at time of Overhaul	Since Last (RH)	Wear (mm)
Inspection	Cylinder Liner w/out insulation	3,000 Hours	Unit 1	C-GER-01	03-May-2012	20,325	12,975	
			Unit 2	C-GER-02	16-Sep-2014	40,641	2,850	
			Unit 3	C-GER-05	14-Sep-2011			
			Unit 4	C-GER-12	28-Oct-2014	24,300	2,600	
			Unit 5	C-GER-06	10-Sep-2011	27,256	7,140	
			Unit 6	C-GER-19	14-Sep-2011	8,670	16,630	
			Unit 7	C-GER-11	04-Sep-2011	40,072	16,660	
			Unit 8	new liner				

Abb. 11: Resultat Beispiel-Report: Übersicht über die Wartungen der Motoren eines Schiffes

Das Resultat zeigt die Art der Aktivität, die Beschreibung der Komponente, das Wartungsintervall in Stunden, die Einheiten, sowie Daten der Wartung wie Durchführungsdatum, Laufzeit der Komponente, etc.

Ein wichtiger Unterschied zwischen der MRE und den Reports im r5 ist, dass die MRE fähig ist, die Ausführung von Reports zu terminieren. Viele Benutzer wünschen sich in regelmässigen Abständen den gleichen Report mit den jeweils aktuellen Daten und sie möchten diesen auch automatisch erhalten. Der Benutzer kann diese Terminierung selbst vornehmen. Bei jedem Report findet sich ein Symbol in Form einer Uhr:



Abb. 12: Symbol zur Terminierung von MRE Reports

Wählt man dieses Symbol aus gelangt man zum Fenster zur Terminierung von Reports. Darin kann man die benötigten Informationen hinterlegen:

Abb. 13: Terminierung von Reports (Einstellungen)

Im Bereich **Schedule** hinterlegt man die allgemeinen Informationen wie Jobname und die Ausführungsdaten. Dabei kann man zwischen drei Optionen auswählen:

- Einmalige, sofortige Ausführung des Reports („Right now“)

- Einmalige Ausführung des Reports an einem bestimmten Datum („Once“)
- Wiederkehrende Ausführung im angegeben Intervall („Recurring“)

Im Bereich **Parameter** kann, falls erforderlich, der entsprechende Parameter der für den Report angegeben wurde hinterlegt werden.

Im Bereich **Save As** werden die Information hinterlegt, die für den Report nach der Ausführung benötigt werden. Dabei kann angegeben wo das Output-Dokument abgelegt werden und ob der Report per E-Mail verschickt werden soll.

Abb. 14: Terminierung von Reports - Optionen für den Output

Die entsprechende E-Mail Adresse ist im Profil des Benutzers hinterlegt.

2.1.3 Reports auf der Web-Plattform

Wie in Kapitel 1.1 erwähnt, ist die Web-Plattform seit 2014 in Entwicklung. Im August 2015 wurde die erste Version produktiv geschaltet. Es handelt sich dabei um einen Client für die Lieferanten von Ersatzteilen und Materialien. Für diesen Client wurde bereits eine Reporting-Lösung implementiert. Ein grosser Unterschied zu den Reports in r5 ist die Auswahl der Elemente. In r5 können die Elemente die im Report erscheinen sollen ausgewählt werden. Auf der Web-Plattform (MW) werden Reports nur für alle angezeigten Elemente ausgeführt. Beispiel:

No.	Buyer	Due date	Status
1	Univan Ship Management Ltd.		Sent
2	Univan Ship Management Ltd.		Sent

Abb. 15: Übersicht der Offerten

In Abb. 12 ist die Übersicht der Offerten, die der Benutzer erstellt hat, zu sehen. Möchte der Benutzer nun eine Übersicht dieser Offerten in einem Report aufbereitet haben braucht er nun nur den Button „PDF Report“ zu bestätigen und er hält sofort den gewünschten Report:

Offer Overview

van Ship Management Ltd. (Hong Kong)

Offer Number	Offer Date	Brand - Type	Installation
RFQ Number	RFQ Date	Label	Person
		Serial No.	Telephone
		Builder	
	04-Aug-2015	Tanabe - H-74	NAVE GALACTIC - STAGE (91919191
3-123-E/02	04-Aug-2015	80368	
		Shanghai Jaingnan Changxing Shipyard Ltd.	
	04-Aug-2015	Tanabe - H-74	NAVE GALACTIC - STAGE (91919191
3-123-E/02	04-Aug-2015	80368	
		Shanghai Jaingnan Changxing Shipyard Ltd.	
	04-Aug-2015	Shinko - CVA 450	NAVE GALACTIC - STAGE (91919191

Abb. 16: Beispiel-Report Web-Plattform: Übersicht Offerten

2.2 Datenquellen

Wie in Abb. 3. zu sehen ist, wird nach dem Laden des Report Templates die Datenquelle gelesen. Die Datenquelle ist ein Element in BIRT, das festlegt, auf welche Art und Weise die Daten aufbereitet werden. Dabei unterscheidet man grob zwischen zwei Kategorien: Scripted Data Sources und Datenbanken. Die Scripted Data Sources verwendet man dann, wenn man die Daten nicht mittels SQL, sondern mittels Java aufbereiten möchte.

2.3 Datenstrukturen

Mit Datenquellen alleine können noch keine Resultate abgebildet werden. Dazu benötigt man sogenannte Datasets. Ein Dataset ist im Grunde nichts anderes als ein relationales Modell einer Untermenge von Daten. Diese Daten können im Report verwendet werden. Um ein Dataset zu erzeugen kann man eine SQL Abfrage schreiben (für Datasets, die als Basis eine Datenbank haben) oder diese programmatisch abfragen.

2.4 Report Templates

BIRT Reports werden als Template mittels dem BIRT Report Designer erstellt. Die BIRT Runtime ist grundsätzlich abwärtskompatibel jedoch kann eine Runtime keinen Report ausführen der mit einem neueren BIRT Report Designer erstellt wurde. Nachfolgend ist eine Übersicht der Runtime und der dazu benötigten Ausgabe des BIRT Report Designers:

Plattform	Runtime Version	BIRT Designer Version
r5	2.5.2	2.5.1
MRE	2.6.2	2.6.2, Proprietär
MESPAS Web	3.2.23	3.2.23

Tabelle 1: Verwendete BIRT Version

2.5 Schwächen

Das aktuelle System funktioniert in allen Belangen fehlerfrei. Da jedoch in der neuen Lösung keine einzelne, zentrale Datenbank zur Verfügung muss die Verarbeitung neu entworfen werden. Zudem ist die proprietäre Lösung für die MRE sehr kostspielig, daher ist eine eigene Lösung nur schon aus wirtschaftlichen Überlegungen begrüssenswert.

Story Nr.	US-02	Priorität: mittel
Titel	Getrennte Bereiche für Standard- und Kundenspezifische Reports	
Beschreibung	Als Benutzer möchte ich einen Bereich haben in dem meine kundenspezifischen Reports abgelegt sind und von den Standard-Reports getrennt sind.	
Bemerkungen	Ein Kundenspezifischer Report ist ein Report, der speziell für Benutzer dieses Kunden erstellt wurde. Standard-Reports hingegen können von allen Benutzern verwendet werden.	
Akzeptanzkriterien	Benutzer A kann die kundenspezifischen Reports von Benutzer B nicht sehen. Benutzer B kann die kundenspezifischen Reports von Benutzer A nicht sehen. Benutzer A und B können alle Standard-Reports sehen.	

Story Nr.	US-03	Priorität: mittel
Titel	Report-Kategorien	
Beschreibung	Als Benutzer möchte ich, dass Reports nach Kategorien gruppiert werden.	
Bemerkungen	Die Applikation kennt verschiedene Module und die Reports sollen in Kategorien, die sich an diesen Modulen orientieren, eingeteilt werden.	
Akzeptanzkriterien	Die Reports sind korrekt kategorisiert. Reports die keiner bestimmten Kategorie zugeordnet werden können sind in einer allgemeinen Kategorie eingeteilt.	

Story Nr.	US-04	Priorität: hoch
Titel	Report suchen	
Beschreibung	Als Benutzer möchte ich nach einem Report suchen können.	
Bemerkungen		
Akzeptanzkriterien	Die Suche liefert alle Reports zurück die den Suchkriterien entsprechen. Die Suche liefert keine falschen Resultate zurück.	

3.4.2 Automatisierte Reports

Story Nr.	US-05	Priorität: hoch
Titel	Terminierung von Reports	
Beschreibung	Als Benutzer möchte ich einen Report terminieren können	
Bemerkungen	Ein Benutzer soll diesen Schedule selbst setzen können.	
Akzeptanzkriterien	Ein terminierter Report wird im korrekten Intervall ausgeführt Ein terminiert ausgeführter Report enthält die korrekten Daten.	

Story Nr.	US-06	Priorität: hoch
Titel	Mail-Versand	
Beschreibung	Als Benutzer möchte ich, dass ein Report automatisch per Mail an mich versendet werden kann.	
Bemerkungen	Abhängigkeit von Story S3	
Akzeptanzkriterien	Der Report kommt an die im Benutzerprofil hinterlegte E-Mail Adresse	

	an. Der korrekte Report wurde ausgeführt. Der Report enthält die korrekten Daten.
--	---

Story Nr.	US-07	Priorität: hoch
Titel	Löschen von Schedules	
Beschreibung	Als Benutzer möchte ich die Ausführung von terminierten Reports stoppen können.	
Bemerkungen		
Akzeptanzkriterien	Ein terminierter Report ist nach der Lösung nicht mehr sichtbar.	

Story Nr.	US-08	Priorität: niedrig
Titel	Status von terminierten Reports	
Beschreibung	Als Benutzer möchte ich, dass ich eine Übersicht über meine terminierten Reports mit deren Status einsehen kann.	
Bemerkungen	Die Übersicht soll alle terminierten Reports anzeigen, unabhängig von dessen Status	
Akzeptanzkriterien	Der Reporting-Service enthält einen Bereich mit einer Status-Übersicht Die Statusmeldungen liefern Informationen über den Job.	

Story Nr.	US-09	Priorität: mittel
Titel	Erneutes Ausführen fehlgeschlagener Jobs	
Beschreibung	Als Benutzer möchte ich, dass, wenn ein Ausführungsjob fehlschlägt ein Administrator (Entwickler) diesen manuell erneut ausführen kann.	
Bemerkungen		
Akzeptanzkriterien	Ein fehlgeschlagener Ausführungsjob kann durch den Benutzer mit einem Klick erneut ausgeführt werden.	

3.4.3 Report-Eigenschaften

Story Nr.	US-10	Priorität: mittel
Titel	Formate	
Beschreibung	Als Benutzer möchte ich zwischen verschiedenen Output-Formaten wählen können.	
Bemerkungen		
Akzeptanzkriterien	Bei der Ausführung eines Report kann der Benutzer zwischen folgenden Formaten als Output auswählen: PDF, Excel	

Story Nr.	US-11	Priorität: hoch
Titel	Report-Parameter	
Beschreibung	Als Benutzer möchte ich, dass Reports Parameter entgegen nehmen können	
Bemerkungen		
Akzeptanzkriterien	Parameter wird korrekt verarbeitet. Parameter ist vom korrekten Typ	

	Report liefert korrekte Daten aufgrund des Parameters zurück
--	--

3.4.4 Datensicherheit

Story Nr.	US-12	Priorität: hoch
Titel	Mandantentrennung	
Beschreibung	Als Benutzer möchte ich, dass meine Daten nicht für Benutzer anderer Firmen sichtbar sind.	
Bemerkungen		
Akzeptanzkriterien	Der Reporting-Service stellt sicher, dass die Mandantentrennung gewährleistet ist.	

3.4.5 Administrator Stories

Story Nr.	US-13	Priorität: mittel
Titel	Administrator – Report-Rollen	
Beschreibung	Als Administrator möchte ich den Reports die benötigten Rollen zuweisen können.	
Bemerkungen	Die ganze Benutzerverwaltung ist bereits vorhanden und somit nicht Teil dieser Lösung.	
Akzeptanzkriterien	Der Report kann von der entsprechenden Rolle ausgeführt werden Der Report kann nicht von unberechtigten Rollen ausgeführt werden.	

Story Nr.	US-14	Priorität: mittel
Titel	Administrator – Reports hochladen	
Beschreibung	Als Administrator möchte ich einen Report auf das System laden können.	
Bemerkungen		
Akzeptanzkriterien	Der Administrator kann einen Report hochladen Ein Benutzer hat keine Optionen einen Report zu verändern oder hochzuladen	

Story Nr.	US-15	Priorität: mittel
Titel	Administrator – Erneutes Versenden fehlgeschlagener Reports	
Beschreibung	Als Administrator möchte ich ein Report-Template auf das System laden können.	
Bemerkungen	Die gleiche Anforderung existiert für Benutzer in Story US-07	
Akzeptanzkriterien	Ein fehlgeschlagener Ausführungsjob kann durch einen Administrator mit einem Klick erneut ausgeführt werden.	

3.5 Funktionale Anforderungen

Zu den im vorherigen Kapitel erarbeiteten Stories müssen nun die Anforderungen entwickelt werden.

3.5.1 Anforderungen Allgemeines

Anforderung Nr.	R-001	Priorität: mittel
Story	US-01	
Titel	Report ausführen	
Beschreibung	Ein Report soll durch einen Benutzer ausgeführt werden können. Da ein Report unter Umständen Daten aus verschiedenen Datenbanken beziehen ist es erforderlich, dass der Report mehrere Datenbanken ansteuern kann.	
Bemerkungen		
Akzeptanzkriterien	Der Report enthält die korrekten Daten Das Layout des Reports ist korrekt.	

Anforderung Nr.	R-002	Priorität: mittel
Story	US-02	
Titel	Kundenspezifische Reports	
Beschreibung	Die Story US-02 verlangt, dass die Benutzer über einen Bereich verfügen in dem die KSR abgelegt sind. Diese Reports dürfen nur für die Benutzer der jeweiligen Organisation sichtbar sein. Dies ist über eine Rolle, die als Datenbankfeld realisiert wird, zu steuern.	
Bemerkungen		
Akzeptanzkriterien	Ein Report muss eine Rolle haben. Es ist nicht möglich einem Report keine Rolle zuzuweisen. Benutzer A kann die kundenspezifischen Reports von Benutzer B nicht sehen. Benutzer B kann die kundenspezifischen Reports von Benutzer A nicht sehen. Benutzer A und B können alle Standard-Reports sehen.	

Anforderung Nr.	R-003	Priorität: mittel
Story	US-03	
Titel	Report-Kategorien	
Beschreibung	Wie in der Story US-03 beschrieben, werden Report in Kategorien eingeteilt. Daraus ergibt sich folgende Anforderung: ein Report muss beim Hochladen einer bestimmten Kategorie zugeordnet werden müssen. Dies ist über ein Datenbankfeld zu steuern.	
Bemerkungen		
Akzeptanzkriterien	Ein Report muss eine Kategorie haben Es ist nicht möglich einem Report keine Kategorie zuzuweisen. Der Report wird im korrekten Bereich angezeigt Der Report ist nicht in anderen Kategorien sichtbar.	

Anforderung Nr.	R-004	Priorität: mittel
Story	US-10	
Titel	Report-Format	
Beschreibung	Der Benutzer soll bei der Ausführung eines Reports das Format auswählen können. Zur Auswahl stehen sollen PDF und Excel.	
Bemerkungen		
Akzeptanzkriterien	Das gespeicherte Report-Dokument hat die Korrekte Datei-Endung Das gespeicherte Dokument lässt sich im entsprechenden Programm fehlerfrei öffnen. Das Dokument enthält die korrekten Daten.	

3.5.2 Anforderungen automatisierte Reports

Anforderung Nr.	R-005	Priorität: mittel
Story	US-05	
Titel	Terminierung von Reports – einmalige Ausführung	
Beschreibung	Der Benutzer soll die Möglichkeit haben einen Report zu schedulen. Das heisst, er kann den Zeitpunkt der Ausführung des ausgewählten Reports selber bestimmen.	
Bemerkungen		
Akzeptanzkriterien	Der korrekte Report wird ausgeführt Der Report wird zur korrekten Zeit ausgeführt	

Anforderung Nr.	R-006	Priorität: mittel
Story	US-04	
Titel	Terminierung von Reports – regelmässige Ausführung	
Beschreibung	Der Benutzer soll die Möglichkeit haben einen bestimmten Report regelmässig ausführen zu lassen. Dabei soll er folgendes bestimmen können: - Zeitpunkt der erstmaligen Ausführung - Intervall der Ausführung - Zeitpunkt der letztmaligen Ausführung, oder - Anzahl der Ausführungen (frei wählbar)	
Bemerkungen	Verwandt mit Anforderung R-004	
Akzeptanzkriterien	Der korrekte Report wird ausgeführt Der Report wird zur korrekten Zeit erstmalig ausgeführt Der Report wird zur korrekten Zeit letztmalig ausgeführt Die Ausführung erfolgt in den korrekten Abständen	

Anforderung Nr.	R-007	Priorität: mittel
Story	US-05, US-10	
Titel	Terminierung von Reports – Format	
Beschreibung	Der Benutzer soll die Möglichkeit haben, beim terminieren eines Reports das Format des Enddokumentes auszuwählen. Folgende Formate müssen zwingend unterstützt werden: PDF und XLS(X) (Excel)	
Bemerkungen	Verwandt mit Anforderung R-004	

Akzeptanzkriterien	Der korrekte Report wird ausgeführt Der Report wird zur korrekten Zeit erstmalig ausgeführt Der Report wird zur korrekten Zeit letztmalig ausgeführt Die Ausführung erfolgt in den korrekten Abständen Der Report wird im korrekten Format ausgeführt
---------------------------	---

Anforderung Nr.	R-008	Priorität: mittel
Story	US-07	
Titel	Terminierung von Reports – Ausführung stoppen	
Beschreibung	<p>Der Benutzer muss die Möglichkeit haben die Ausführung eines terminierten Reports zu suspendieren oder zu stoppen. Um die Ausführung zu suspendieren soll jeder terminierte Report mit einer Checkbox versehen werden die auf einen aktiven Status hinweist. Durch Deselektion dieser Checkbox wird die Ausführung suspendiert.</p> <p>Um die Ausführung eines terminierten Reports zu stoppen muss der Benutzer die Möglichkeit haben die Ausführung gänzlich zu stoppen. Das Löschen dieses Schedule ist zu bestätigen.</p>	
Bemerkungen	Verwandt mit Anforderung R-004	
Akzeptanzkriterien	Ein suspendierter Report wird nicht mehr ausgeführt, bis dieser wieder aktiviert wird Ein Report, dessen Ausführung gelöscht wurde, wird nicht mehr ausgeführt Ein Report, dessen Ausführung gelöscht wurde, ist nicht mehr als Schedule sichtbar	

Anforderung Nr.	R-009	Priorität: mittel
Story	US-06	
Titel	Terminierung von Reports – Mail Versand	
Beschreibung	<p>Der Benutzer muss einen Report per Mail versenden können. Dieses Senden soll sich dabei auf automatisierte Reports beschränken, d.h. Report die Manuell ausgeführt werden, werden nicht versendet.</p> <p>Es ist daher ein Feld zur Verfügung zu stellen, dass eine E-Mail Adresse beinhalten kann, an welche der ausgeführte Report zu versenden ist.</p>	
Bemerkungen	Verwandt mit Anforderung R-004	
Akzeptanzkriterien		

Anforderung Nr.	R-010	Priorität: mittel
Story	US-06	
Titel	Terminierung von Reports – Mail Versand / Auswahl	
Beschreibung	Das Versenden von Reports per E-Mail (siehe Anforderung R-007) soll optional sein, d.h. der Benutzer soll beim Terminieren eines Reports auswählen können, ob der Report auch per E-Mail versendet werden können soll. Der Benutzer soll dies mit einer Checkbox auswählen können.	

Bemerkungen	Blockiert von Anforderung R-008
Akzeptanzkriterien	Der korrekte Report wird ausgeführt Ein Schedule mit aktiver E-Mail Checkbox eine E-Mail mit dem Report aus. Ein Schedule ohne aktive Checkbox löst eine E-Mail aus.

Anforderung Nr.	R-011	Priorität: mittel
Story	US-07	
Titel	Terminierung von Reports – Mail Versand / Format	
Beschreibung	Wie in Anforderung R-005 beschrieben, soll der Benutzer beim terminieren eines Reports ein Format auswählen können. Dieses Format muss auch beim Versand der E-Mail berücksichtigt werden.	
Bemerkungen	Blockiert von Anforderung R-008	
Akzeptanzkriterien	Der korrekte Report wird ausgeführt Ein Schedule mit aktiver E-Mail Checkbox eine E-Mail mit dem Report aus. Ein Schedule ohne aktiver Checkbox löst eine E-Mail aus. Die E-Mail enthält den Report im korrekten Format.	

Anforderung Nr.	R-012	Priorität: mittel
Story	US-08	
Titel	Terminierung von Reports – Status	
Beschreibung	Der Benutzer soll den Status jedes terminieren Reports einsehen können. Daher ist eine Seite einzurichten auf der Benutzer tabellarisch eine Übersicht über seine ausgeführten Reports einsehen kann. Die Übersicht soll folgendes beinhalten: - Datum und Zeit - Name des Ausführungsauftrags - Der Benutzer der den Schedule eingestellt hat - Status (erfolgreich oder fehlgeschlagen)	
Bemerkungen	Blockiert von Anforderung R-008	
Akzeptanzkriterien	Die Liste enthält die korrekten Ausführungen Die Ausführungen in der Liste haben die korrekten Status	

Anforderung Nr.	R-013	Priorität: mittel
Story	US-08, US-09	
Titel	Terminierung von Reports – Erneutes Versenden bei Fehlschlag	
Beschreibung	Der Benutzer soll den Status jedes terminieren Reports einsehen können. Daher ist eine Seite einzurichten auf der Benutzer tabellarisch eine Übersicht über seine ausgeführten Reports einsehen kann. Die Übersicht soll folgendes beinhalten: - Datum und Zeit - Name des Ausführungsauftrags - Status (erfolgreich oder fehlgeschlagen)	

Bemerkungen	Blockiert von Anforderung R-008
Akzeptanzkriterien	Die Liste enthält die korrekten Ausführungen Die Ausführungen in der Liste haben die korrekten Status

3.5.3 Anforderungen Report Eigenschaften

Anforderung Nr.	R-014	Priorität: mittel
Story	US-11	
Titel	Report-Parameter - Allgemein	
Beschreibung	Ein Report muss unter Umständen parametrisiert werden können. Dabei ist sicherzustellen, dass das Format korrekt verarbeitet wird und mit dem Parameter die korrekten Daten geliefert werden.	
Bemerkungen		
Akzeptanzkriterien	Das Format des Parameters ist korrekt Die Daten sind korrekt.	

Anforderung Nr.	R-015	Priorität: mittel
Story	US-11	
Titel	Report-Parameter – Optionale Parameter	
Beschreibung	Ein Report muss optionale Parameter unterstützen, d.h. dem Benutzer ist es freigestellt eine Auswahl zu treffen.	
Bemerkungen		
Akzeptanzkriterien	Das Format des Parameters ist korrekt Die Daten sind korrekt.	

3.6 Nicht-Funktionale Anforderungen

Neben den funktionalen Anforderungen gibt es einige nicht-funktionale Anforderungen die die neue Lösung erfüllen soll. Diese sind wie folgt definiert:

3.6.1 Performance

Die Performance muss zwischen zwei unterschiedlichen Aspekten unterschieden werden: die Performance der Reports und die der Plattform selbst. Unter der Performance der Plattform versteht man die Geschwindigkeit mit der:

- einzelne Seiten geladen werden
- nach Reports gesucht werden kann

Daraus ergeben sich folgende Anforderungen:

Anforderung Nr.	R-016	Priorität: mittel
Titel	Performance - Report	
Beschreibung	Reports sollen grundsätzlich performant sein. Bei Reports, die direkt ausgeführt d.h. nicht als Schedule werden, möchte der Benutzer nicht lange auf das Ergebnis warten müssen. Bei terminierten Reports ist die Akzeptanz höher.	

	Da mit der neuen Lösung die Daten von verschiedenen Datenbanken geholt werden müssen kann es eine leichte Verschlechterung der Performance in dieser Hinsicht geben.
Bemerkungen	
Akzeptanzkriterien	Ein Report, der direkt ausgeführt wird benötigt für die Ausführung nicht länger als 60 Sekunden Ein terminierter Report benötigt für die Ausführung nicht länger als 180 Sekunden. Die Performance der Reports darf generell nicht mehr als 10% schlechter als vorher sein.

Anforderung Nr.	R-017	Priorität: mittel
Titel	Performance – Suche	
Beschreibung	Der Benutzer möchte, dass Suchresultate rasch erscheinen.	
Bemerkungen		
Akzeptanzkriterien	Die Suche nach einem Report dauert nicht länger als 15 Sekunden.	

Anforderung Nr.	R-018	Priorität: mittel
Titel	Performance – Plattform	
Beschreibung	Der Benutzer möchte, dass die die Funktionalität der Plattform eine gute Performance aufweist und er nicht lange auf Reaktionen warten muss.	
Bemerkungen		
Akzeptanzkriterien	Das Reporting darf sich nicht langsamer verhalten als der Rest der Plattform. Entsprechende Performance Tests inklusive Vergleich mit anderen Modulen sind daher durchzuführen.	

3.6.2 Verfügbarkeit

Anforderung Nr.	R-019	Priorität: mittel
Titel	Verfügbarkeit	
Beschreibung	Der Benutzer möchte, dass die Reporting-Plattform grundsätzlich verfügbar ist. Dies ist aber nur der Fall, wenn die ganze Plattform verfügbar ist.	
Bemerkungen	Für die Verfügbarkeit ist Operations zuständig. Jedoch kann sichergestellt werden, dass beim Systemstart auch das Reporting gestartet wird.	
Akzeptanzkriterien	Wenn die ganze Plattform verfügbar ist, sind auch die Reports verfügbar.	

3.6.3 Look and Feel

Anforderung Nr.	R-020	Priorität: mittel
Titel	Look and Feel	
Beschreibung	Die MESPAS AG verfügt über interne Richtlinien bezüglich Corporate Design (CD). Der Benutzer möchte, dass die neue Lösung optisch zum Gesamtbild passt.	
Bemerkungen		
Akzeptanzkriterien	Das Look and Feel der neuen Lösung hält die CD Richtlinien ein.	

3.7 Technische Anforderungen

Da die neue Plattform über einen komplett anderen Aufbau als die bestehende MRE verfügt gibt es einige technische Anforderungen, die erfasst werden müssen.

3.7.1 Daten

Anforderung Nr.	R-021	Priorität: mittel
Titel	Daten – Datenbank	
Beschreibung	Ein Report muss in der Lage sein, sich auf die benötigten Datenbanken zu verbinden und Daten holen zu können.	
Bemerkungen		
Akzeptanzkriterien	Ein Report der zwei oder mehr Datenbanken benötigt verbindet sich zu den korrekten Datenbanken. Ein Report verbindet sich nicht auf nicht benötigte Datenbanken.	

Anforderung Nr.	R-022	Priorität: mittel
Titel	Daten – Datenobjekte	
Beschreibung	Um Daten in einem Report abbilden zu können wird ein sogenanntes Dataset benötigt. Ein Element in einem Report kann immer nur ein Dataset gleichzeitig verwenden (z.B. eine Tabelle kann nur Daten aus einem Dataset abbilden). Daher ist sicherzustellen, dass die Daten in sinnvoller Art und Weise zusammengestellt werden.	
Bemerkungen		
Akzeptanzkriterien	Das Dataset enthält die korrekten Daten die abgebildet werden müssen.	

3.7.2 Scheduler

Anforderung Nr.	R-023	Priorität: mittel
Titel	Scheduler	
Beschreibung	Da Reports terminiert werden können sollen, benötigt die Lösung einen Scheduler. Dieser soll regelmässig prüfen, ob es einen Report gibt, der ausgeführt werden soll und wenn eine Intervallsgrenze erreicht ist, den entsprechenden Report oder die gewünschte Aktion ausführen.	

Bemerkungen	Zunächst sind bestehende Lösungen zu prüfen
Akzeptanzkriterien	Der Scheduler führt die Aufgaben in den korrekten Abständen aus.

3.7.3 Logging

Anforderung Nr.	R-024	Priorität: mittel
Titel	Logging - Fehler	
Beschreibung	In der MRE findet in der heutigen Form kein echtes Logging statt. Wenn ein Fehler auftritt erhält der Benutzer zwar eine Fehlermeldung aber ein Stacktrace wird nicht geloggt. In der neuen Lösung ist daher darauf zu achten, dass allfällige Exceptions korrekt geloggt werden.	
Bemerkungen		
Akzeptanzkriterien	Bei Fehlern wird der korrekte Stacktrace geloggt	

Anforderung Nr.	R-025	Priorität: mittel
Titel	Logging - Ausführung	
Beschreibung	In r5 wird die Ausführungszeit der Report in das Serverlog geschrieben, in der MRE jedoch nicht. Da die Ausführungszeit die Wichtigste Kennzahl für die Performance ist, ist die Ausführungszeit jedes Reports in der neuen Lösung zu loggen.	
Bemerkungen		
Akzeptanzkriterien	Für jeden Report werden zwei Log-Einträge mit folgendem Inhalt geschrieben: Start: Timestamp, Benutzer-Id, Report-Id Ende. Timestamp, Benutzer-Id Report-Id, Ausführungszeit	

4 Recherche

Nachdem die Anforderungen erfasst wurden müssen nun die benötigten Komponenten ermittelt werden. Zunächst wird festgestellt, welche Werkzeuge für die Umsetzung benötigt werden. Gibt für einen Einsatzbereich mehrere ähnlich gute Lösungen, ist eine Nutzwertanalyse durchzuführen.

4.1 Benötigte Werkzeuge

4.1.1 Scheduling

Eines der wichtigsten Features der neuen Lösung ist die Fähigkeit, Reports zu terminieren (User Story US-05) Um diese Reports entsprechend ausführen zu können wird ein Scheduler benötigt der konfigurierbar ist und die Aufgaben zuverlässig ausführt.

4.1.2 Rendering

Da die Reports auf Basis eines Templates erstellt werden sollen wird ein Tool benötigt, das in der Lage ist, BIRT Templates zu verarbeiten.

4.1.3 Mail-Versand

Die neue Lösung soll Reports auch per Mail versenden können (User Story US-06). Daher wird ein Werkzeug benötigt, das es erlaubt, programmatisch E-Mail mit Attachment zu versenden.

4.1.4 Dokumente erstellen

Die User Story US-10 verlangt, dass Reports in unterschiedlichen Dateiformaten erstellt werden können. Daher wird ein Tool benötigt, das beim Rendering entsprechend Output-Formate unterstützt.

4.2 Evaluation der Werkzeuge

4.2.1 Scheduler

In MESPAS r5 wurde bisher Quartz von Apache für das Scheduling verwendet. Da in MESPAS Web jedoch Spring verwendet wird soll auch der Spring Scheduler geprüft werden.

4.2.1.1 Spring

Das Spring Framework bietet Schnittstellen für die asynchrone Ausführung und terminieren von Tasks an^[1]. Über ein XML File können die Grundeinstellungen konfiguriert werden und über die API, die vom Spring Framework bereitgestellt wird, können die Schedules direkt programmatisch eingestellt werden.

4.2.1.2 Quartz

Quartz ist ein Scheduling Tool von Terracota das als Open-Source Software unter der Apache 2.0 Lizenz erhältlich ist. Im Gegensatz zu Spring müssen die Intervalle als Cronjob in einem XML File eingestellt werden.

4.2.1.3 Nutzwert-Analyse

Da auf den ersten Blick beide Lösung bereits im Einsatz sind und ihre Stärken haben ist hier eine Nutzwert-Analyse zur Entscheidung vonnöten. Dabei sind verschiedene Kriterien zu berücksichtigen die als Entscheidungsgrundlage dienen sollen. Folgende Kriterien werden festgelegt:

- Abdeckung der benötigten Funktionen
- Einfachheit der Integration
- Produktreife
- Komplexität
- Community

Um die Kriterien und Gesamtheit zu bewerten um letztendlich eine Entscheidung für ein Werkzeug treffen zu können wird eine Nutzwertanalyse (NWA) durchgeführt. Es handelt sich dabei um ein relativ altes Verfahren, das seine Ursprünge in der volkswirtschaftlichen „Utility Analysis“ hat^[2]

Zunächst werden die Kriterien für die Varianten etwas genauer betrachtet. Dazu wird untersucht, in wie weit die Variante für die Lösung geeignet ist. Dazu werden Punkte von 1 bis 5 gegeben und gewichtet. Das Punkteschema entspricht einer linearen Skala die die Erfüllung der geforderten Funktionalität widerspiegelt:

- 0 Punkte: Funktionalität überhaupt nicht erfüllt
- 1 Punkt: Funktionalität nur minimal erfüllt
- 2 Punkte: Funktionalität weist gravierende Lücken auf
- 3 Punkte: Funktionalität weist akzeptable Lücken auf
- 4 Punkte: Funktionalität weitgehend abgedeckt
- 5 Punkte: Funktionalität komplett erfüllt.

Am Schluss werden die gewichteten Punkte addiert und ermöglicht so eine Entscheidung für eine Variante. Dies ergibt folgende Formel zur Berechnung des Nutzwertes:

$$N_k = \sum_{i=1}^r n_{ki} w_i \quad \text{mit} \quad \sum_{i=1}^r w_i = 1$$

Erklärung:

N_k : Gesamtnutzwert

N_{ki} : Nutzwert Kriterium

W_i : Gewicht

4.2.1.3.1 Abdeckung der Funktionalität

Beide Varianten bringen funktional alles mit was für den Einsatz benötigt wird. Für beide Lösungen sind die Schedules individuell konfigurierbar und funktionieren sehr zuverlässig. Bewertung: je 5 Punkte.

4.2.1.3.2 Integration in das Projekt

Beide Lösungen sind relativ simpel in das Projekt integrierbar. Da Quartz nicht zum Spring Framework gehört muss die Library erst heruntergeladen und über den Klassenpfad in das Projekt eingebunden werden. Bewertung: 3 Punkte.

Da die Microservice das Spring Framework verwendet muss der Scheduler nicht zusätzlich eingebunden werden, er kann also sofort verwendet werden. Bewertung: 4 Punkte.

4.2.1.3.2 Produktreife

Die erste Version von Quartz war bereits 2002 verfügbar^[3]. Seither wurden Weiterentwicklungen durchgeführt. Jedoch schneidet Quartz in punkto Fehlerbehandlung ungenügend ab. Bewertung: 3 Punkte.

Spring ist seit 2004 im Einsatz und ein sehr umfangreiches Framework und wird ebenfalls ständig weiterentwickelt. Die Fehlerbehandlung ist einfach und intuitiv.

Bewertung: 5 Punkte.

4.2.1.3.3 Komplexität

Ein wichtiges Kriterium ist auch die Komplexität. Für ein Werkzeug ist es vorteilhaft wenn es in- nert nützlicher Frist verwendet werden kann und nicht erst wochenlange Einführungskurse besucht werden müssen. Quartz ist einiges komplexer in der Verwendung als der Spring Scheduler, da das Setup von Quartz einiges aufwändiger ist und Quartz wie im Kapitel vorher erwähnt in Sachen Fehlerbehandlung sehr unübersichtlich ist. Bewertung: 2 Punkte.

Spring ist umfassend dokumentiert und intuitiv in der Anwendung. Auch komplexere Schedules können ohne grossen Aufwand eingestellt werden. Bewertung: 4 Punkte.

4.2.1.3.4 Community

Wenn ein externes Werkzeug verwendet wird, möchte man im Fall von Problemen schnell Hilfe bekommen. Gerade in der Open-Source Welt ist die Community ein wichtiger Grundpfeiler für Fehlerbehebung und Hilfestellung bei Problemen. Daher wird die Community der beiden Varianten etwas näher betrachtet.

Quartz ist ein weit verbreitetes Tool daher gibt es auch viele Foren, Tutorials, etc. an die man sich bei Problemen wenden kann. Bewertung: 4 Punkte

Spring als grosses Framework verfügt über eine riesige Community wo man zu fast jedem Problem rasch eine Lösung findet. Das Forum war bisher eine zentrale Anlaufstelle, dieses wurde jedoch geschlossen da der Support nun über Stackoverflow stattfindet. Stackoverflow bietet so alle Vorteile die das Forum hatte. Bewertung: 5 Punkte.

4.2.1.3.5 Entscheidungstabelle

Die vergebenen Punkte für die Kriterien ergeben folgende Matrix:

Kriterium	Gewichtung	Bewertung A (Quartz)	Bewertung B (Spring)	Gewichteter Teilnutzen A	Gewichteter Teilnutzen B
Abdeckung der Funktionalität	50%	5	5	2.5	2.5
Integration in das Projekt	20%	3	4	0.6	0.8
Produktreife	5%	3	5	0.15	0.25
Komplexität	15%	2	4	0.3	0.6
Community	10%	4	5	0.4	0.5
	100%				
				Nutzwert A	Nutzwert B
				3.95	4.65

Tabelle 2: Nutzwert-Analyse Scheduler

Die Summe der einzelnen Gewichte hat ergeben, dass Spring für dieses Projekt geeigneter und daher verwendet werden wird.

4.2.2 Rendering

Da die bisherigen Report Templates wiederverwendet werden sollen bietet es sich an, weiterhin BIRT als Rendering-Werkzeug zu verwenden. BIRT liefert eine einfach zu integrierende API mit die für die Verarbeitung der Report Templates notwendig ist.

4.2.2.1 Einführung

BIRT stammt aus der Feder von Actuate, welche die initiale Version der Eclipse Foundation nach ihrer Aufnahme zur Verfügung gestellt hatte^[4]. Eclipse hat die das Projekt weiterentwickelt und BIRT ist als Open-Source Projekt verfügbar.

4.2.2.2 Verwendung

Um BIRT innerhalb der Applikation zu verwenden muss die Library ins Projekt eingebunden werden. Dann können die benötigten Klassen direkt verwendet werden.

```
11 import org.eclipse.birt.report.engine.api.EngineConfig;  
12 import org.eclipse.birt.report.engine.api.ReportEngine;  
13 import ch.srgen.lib.logging.Logger;
```

Abb. 18: Import der BIRT Packages

Um einen Report innerhalb einer Java Applikation auszuführen muss ein „*IReportRunnable*“ Objekt für das Report Template angelegt werden. Dazu wird eine Instanz der Klasse *ReportEngine* benötigt. Über diese kann das Template dann als Objekt geöffnet werden:

```
IReportRunnable design;  
is = new FileInputStream(reportTemplate);  
design = engine.openReportDesign(is);
```

Listing 1: Instanzieren eines *IReportRunnable* Objektes

Danach muss für die Ausführung ein BIRT Task erstellt werden:

```
final IRunAndRenderTask task = engine.createRunAndRenderTask(design);
```

Listing 2: Erstellen eines BIRT Tasks

Nun ist der Task bereit zur Ausführung. Nun können mit der Klasse *IRenderOption* noch weitere Optionen hinzugefügt werden. Dies kann nützlich sein, wenn man zum Beispiel Verbindungsinformationen an die Reporting-Engine weitergeben möchte.

Dasselbe gilt für Report Parameter. Diese können hier direkt weitergegeben werden, um so beispielsweise Informationen über die Umgebung oder den aktuellen Benutzer standardmässig weiterzugeben.

```
// parameters is a Map that is passed to the method generateReport()  
task.setParameterValues(parameters);
```

Listing 3: Übergabe von Parametern an einen Report

Nun ist das Task-Objekt fertig vorbereitet und kann mit der Methode *run()* ausgeführt werden. Für den Fall, dass bei der Ausführung Fehler aufgetreten sind, können diese mit der Methode

`getErrors()` in eine Liste geschrieben werden. Zuletzt muss der Task mit `close()` geschlossen werden. Zurückgegeben wird dabei ein File, welches dann geöffnet werden kann.

4.2.3 Output Format

Die neue Lösung soll zunächst zwei Output-Formate unterstützen: PDF und Excel. Daher wird ein Tool benötigt welches diese Formate ermöglicht. Da BIRT bereits als Renderer feststeht, es nahe-liegend die BIRT Library auch hier zu verwenden, da das OutputFormat lediglich als Option gesetzt

```
IRenderOption options = new RenderOption();  
options.setOutputFormat("pdf");
```

Listing 4: Output-Format eines Reports festlegen

Somit werden die Anforderungen der Stories US-01, US-10 und US-11 erfüllt.

4.2.4 Mailer

Die Story US-06 verlangt, dass ein Report per E-Mail versendet werden können soll. Dazu muss die Applikation einen Mailer verwenden der in der Lage ist, Attachments zu versenden. Da in der Applikation bereits JavaMail verwendet wird, ist eine weitere Untersuchung müssig, da JavaMail über sämtliche Funktionen, die benötigt werden, verfügt.

4.2.4.1 Einführung in JavaMail

Die JavaMail API bietet ein plattform- und protokollunabhängiges Framework an um E-Mails und Nachrichtenapplikationen zu erstellen^[5]. Um die API zu verwenden muss die JavaMail Bibliothek im Klassenpfad eingebunden werden.

Die Verwendung von JavaMail ist relativ trivial. Im folgenden Abschnitt wird kurz erläutert wie die Library verwendet werden kann.

4.2.4.2 Verwendung von JavaMail

Zunächst muss eine Klasse erstellt werden, die das Interface „Mail“ implementiert. Die im Interface definierten Methoden müssen ebenso implementiert werden und es ist empfehlenswert, die notwendigen Parameter als Instanzvariablen zu deklarieren und diese im Konstruktor zu initialisieren:

```
public class MyMail implements Mail {  
  
    /* mail settings */  
  
    private String host;  
    private String user;  
    private String pass;  
    private int port;
```

Listing 5: Implementierung einer Mail-Klasse

Wie in diesem Listing zu sehen ist wird ein zunächst User, Host, Passwort und ein Port benötigt. Dies sind die Parameter für den Mail-Server über den die E-Mail verschickt werden soll.

Bevor die Mail versendet werden kann muss jedoch eine Session aufgebaut werden. Dazu werden die Verbindungsdaten festgelegt und die Standard-Instanz der Klasse Session mit allen benötigten Daten aufgerufen:

```
@Override
public void createSession() throws NoSuchProviderException {
    session = Session.getDefaultInstance(properties,
        new javax.mail.Authenticator() {
            protected PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(user,pass);
            }
        });
}
```

Listing 6: Erstellen einer Session für den Mail-Versand

Um die Mail vorzubereiten wird zunächst ein Message Objekt erstellt, das die eigentliche Mail repräsentiert. Es benötigt Daten wie Empfängeradresse, CC/BCC Feld, einen Betreff und einen Textkörper (-> body). Soll ein Anhang mitgesendet werden wird zudem ein Objekt für den Anhang benötigt. Dafür ist die Klasse MimeBodyPart.

```
// neues Mail-Objekt
message = new MimeMessage(session);
// Absenderadresse festlegen
message.setFrom(new InternetAddress(user));
// Empfängeradresse
message.setRecipients(Message.RecipientType.TO, InternetAddress.parse(email));
// Blindkopie
message.setRecipients(Message.RecipientType.BCC, InternetAddress.parse(BCC));
// Betreff
message.setSubject(subject);
// Mail-Text
message.setText(body);
```

Listing 7: Erstellen eines Mail-Objektes

Wird ein Anhang benötigt muss das Mail in zwei Teile zerlegt werden. Dazu kann ein Objekt der Klasse MimeBodyPart erstellt und der Mail-Text und das Attachment darin gesetzt werden:

```
MimeBodyPart messagePart = new MimeBodyPart();
messagePart.setText(body);
MimeBodyPart attachmentPart = new MimeBodyPart();
FileDataSource fds = new FileDataSource(file) {
    @Override
    public String getContentType() {
        return Mail.MIME_PDF;
    }
};
attachmentPart.setDataHandler(new DataHandler(fds));
attachmentPart.setFileName(fds.getName());
Multipart multipart = new MimeMultipart();
multipart.addBodyPart(messagePart);
multipart.addBodyPart(attachmentPart);
message.setContent(multipart);
```

Listing 8: Erstellen eines Mail-Objektes mit Attachment

Mit der Methode setContent() wird das Mail mit dem Anhang zum Mail-Objekt hinzugefügt und mit Transport.send() kann es daraufhin versendet werden.

Mit diesen wenigen Funktionen kann also eine E-Mail vorbereitet, ein Anhang hinzugefügt und dann versendet und die User Story US-06 somit erfüllt werden.

5. Design

Grundsätzlich wird das Ziel verfolgt, einen Report mit wenigen Klicks ausführen zu können. Die Daten müssen korrekt sein und das Look-And-Feel den internen Richtlinien entsprechen.

Unabhängig von der exakten Art und Weise wie ein Report ausgeführt wird gibt es einige Gemeinsamkeiten die für jedwede Variante gleich ist. Diese Gemeinsamkeiten sollen in diesem Kapitel beschrieben werden.

5.1 Benutzeroberfläche

Unabhängig von der Integration wird die Benutzeroberfläche im Einklang mit den internen Richtlinien gestaltet werden. Die Hauptseiten in MW haben jeweils einen ähnlichen Aufbau, der aus 4 Komponenten besteht:

- Navigation
- Header
- Suche
- Anzeige

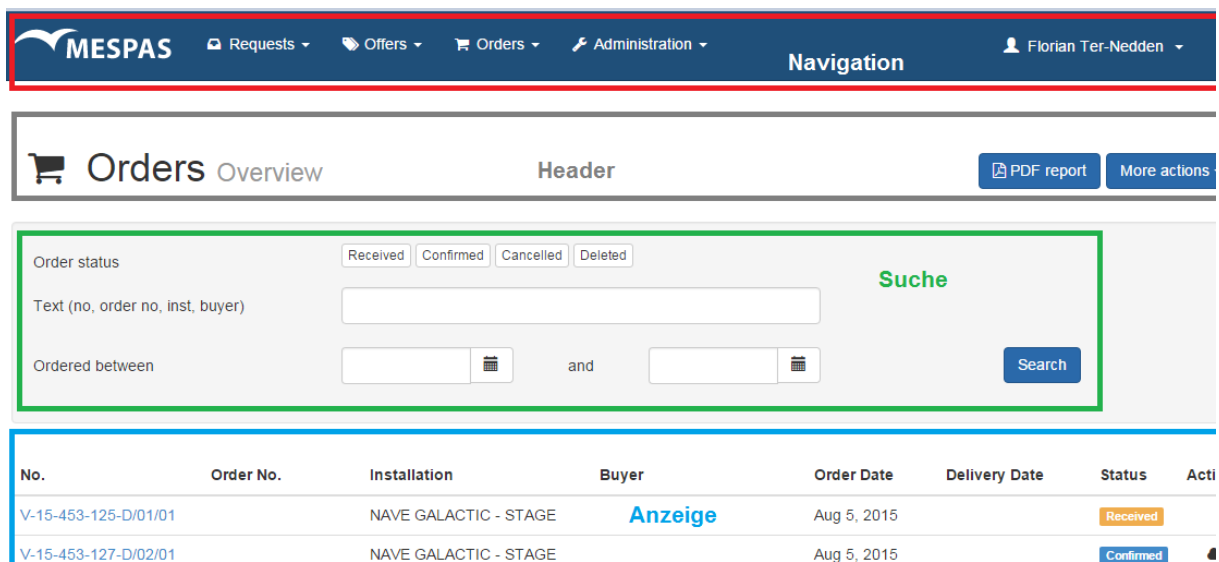


Abb. 19: Aufbau Hauptseiten

Die neue Reporting-Lösung soll daher passend eingefügt werden.

5.1.1 Navigation

Damit ein Benutzer die neue Lösung benutzen kann muss er über die notwendige Rolle verfügen. Die Implementierung dieser Rollen wurde bereits ausserhalb dieser Arbeit vorgenommen. Wenn ein Benutzer die benötigte Rolle hat ist ein Bereich Reporting in der Navigationsleiste verfügbar:

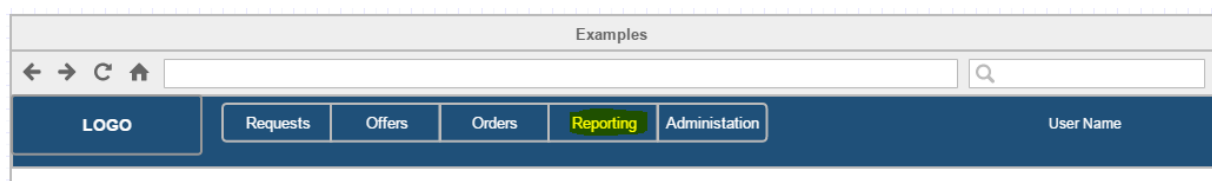


Abb. 20: Mock-Up neue Navigationsleiste

5.1.2 Header und Suche

In der neuen Lösung soll der Benutzer nach einem Report suchen können. Dabei kann er folgende Suchparameter auswählen:

- Report-Kategorie (Report Type) als Auswahlbutton
- Freitext als Textfeld
- Kundenspezifische und / oder Standard-Reports als Checkbox
- Upload-Datum als Datumfelder (von -> bis)

Im Bereich Report Kategorie sollen folgende Buttons verfügbar sein:

- Procurement (Einkauf)
- PMS (Planned Maintenance System)
- Inventory (Inventar)
- IDM (Installation Data Management -> Information und Einstellungen für ein Schiff)
- DMS (Document Management System -> Dokument wie z.B. Handbücher, Zertifikate, etc.)
- Various (Verschiedenes -> alles was keiner Kategorie zugeordnet werden kann).

Daraus ergibt sich folgendes Mock-Up:

Abb. 21: Mock-Up neuer Header und Suche

5.1.3 Anzeige

In diesem Bereich werden alle Reports aufgelistet, die der Benutzer ausführen kann, das heisst seine kundenspezifischen sowie die Standard-Reports. Dabei sollen die Standard und die kundenspezifischen Bereich jeweils in einem eigenen Bereich angezeigt und die folgenden Informationen in den Übersichten dargestellt werden:

- Report Name
- Report Beschreibung
- Upload-Datum
- Kategorie
- Aktionen (Ausführen und Terminieren)

Daraus ergeben sich für die Anzeige folgende Mock-Ups:

My Reports

Standard Reports

Jobs

My Reports

Report Name	Description	Upload Date	Type	Actions
Supplier Overview	Supplier Overview / all customer companies available	2015-05-05 15:15:15	Procurement	Run Schedule
Stock Value	Inventory list with value of items	2015-04-04 14:14:14	Inventory	Run Schedule
Jobs_Overdue_Year_Fleet	Overview over the OD within the months of the year	2015-06-06 12:13:39	PMS	Run Schedule
Power_Characteristics	Engine Power characteristics	2013-01-02 16:19:54	PMS	Run Schedule
Document Overview	List of documents with expiration date on vessel	2014-11-06 19:24:09	DMS	Run Schedule
Case Status Report	Report displaying the current status of a procurement case	2009-08-22 14:42:22	Procurement	Run Schedule

Abb. 22: Mock-Up Anzeige „My Reports“

My Reports

Standard Reports

Jobs

Standard Reports

Report Name	Description	Upload Date	Type	Actions
Supplier Overview	Supplier Overview / all customer companies available	2015-05-05 15:15:15	Procurement	Run Schedule
Stock Value	Inventory list with value of items	2015-04-04 14:14:14	Inventory	Run Schedule
Jobs_Overdue_Year_Fleet	Overview over the OD within the months of the year	2015-06-06 12:13:39	PMS	Run Schedule
Power_Characteristics	Engine Power characteristics	2013-01-02 16:19:54	PMS	Run Schedule
Document Overview	List of documents with expiration date on vessel	2014-11-06 19:24:09	DMS	Run Schedule
Case Status Report	Report displaying the current status of a procurement case	2009-08-22 14:42:22	Procurement	Run Schedule

Abb. 23: Mock-Up Anzeige "Standard Reports"

Zudem benötigt der Benutzer eine Übersicht über seine terminierten Reports. In dieser Übersicht sollen die wichtigsten Informationen über den Status des Jobs, letzte Ausführung, und die Möglichkeit, die Details eines Jobs einzusehen:

My Reports		Standard Reports		Jobs			
Job Name	Report	Last Execution	Last Result	Actions			
Weekly Suppliers List	Supplier Overview	2015-05-05 15:15:15	Succeeded	Resend	Details	Details	
Dayil Order Report	Order Overview	2015-04-04 14:14:14	Failed	Resend	Details	Delete	
Weekly Running Hours Report	Running Hours and Maintenance Report	2015-06-06 12:13:39	Succeeded	Resend	Details	Delete	
Monthly Stock Info	Stock Overview	2013-01-02 16:19:54	Succeeded	Resend	Details	Delete	

Abb. 24: Mock-Up Jobs

5.1.4 Report ausführen

Da einige Reports über Parameter verfügen, die der Benutzer angeben muss, muss eine Möglichkeit geschaffen werden, diese einzugeben. Daraus ergibt sich folgendes Design:

Please fill out the fields

Parameter 1

Parameter 2

Parameter 2 ☐ Param Button1 ☐ Param Button 2

Abb. 25: Eingabe Report Parameter

5.1.5 Job-Details

In der Detailansicht eines Jobs soll der Benutzer die wichtigsten Informationen über einen Job einsehen und editieren können. Für das Anlegen und Editieren eines Jobs soll das gleiche Interface verwendet werden. Neben den allgemeinen Informationen über den Job müssen allfällige Parameter die der Report benötigt übergeben werden können. Zudem stehen dem Benutzer hier zwei Optionen zur Verfügung: Einen Job einmal („Once“) oder mehrmals („Recurring“) zu einer bestimmten Zeit ausführen zu lassen. Zusätzlich muss der Benutzer eine E-Mail Adresse hinterlegen können an die der Report im gewünschten Format (Excel oder PDF) versendet wird. Aus diesen Anforderungen ergibt sich das folgende Mock-Up:

Create Job for Report "sample Report"

Job Name

Schedule ☐ Once ☐ Recurring

at

☐ Starting / /

☐ Ending / /

☐ Send by e-mail to following address:

Report Format

Parameters

Parameter 1

Parameter 1

Parameter 1 ☐ Param Button1 ☐ Param Button 2

Abb. 26: Mock-Up Job Details

Wenn die Details eines existierenden Jobs aufgerufen werden sind die Felder mit den dazugehörigen Informationen entsprechend abgefüllt.

5.2 Benutzeroberfläche Administrator

5.2.1 Header, Navigation und Suche

Dieser Bereich der Benutzeroberfläche soll identisch mit der Oberfläche für den Benutzer. Die Rubrik „My Reports“ trifft auch für den Administrator gleichermassen zu, da die MESPAS AG auch über eigene Reports verfügt. Hier werden entsprechend diese Reports gelistet.

5.2.2 Jobs

Der Administrator muss nicht nur seine eigenen, sondern alle Jobs sehen können. Deshalb werden auf dem Display für die Jobs zusätzliche Felder angezeigt mit denen er nach bestimmten Kriterien filtern kann und zusätzlich auch der Firmenname des entsprechenden Jobs angezeigt wird.

Das Mock-Up für diesen Bereich sieht so aus:

My Reports

Standard Reports

Jobs

Filter

Company

☐ Show failed jobs

☐ Show succeeded jobs

☐ Show only jobs from the last

14 days

Company	Job Name	Report	Last Execution	Last Result	Actions
Company A	Weekly Suppliers List	Supplier Overview	2015-05-05 15:15:15	Succeeded	Resend Details Delete
Company B	Dayil Order Report	Order Overview	2015-04-04 14:14:14	Failed	Resend Details Delete
Company B	Weekly Running Hours Report	Running Hours and Maintenance Report	2015-06-06 12:13:39	Succeeded	Resend Details Delete
Company A	Monthly Stock Info	Stock Overview	2013-01-02 16:19:54	Succeeded	Resend Details Delete

Abb. 27: Mock-Up Jobs für Administratoren

5.3 Datenbank Reporting

Unabhängig von der Implementierung benötigt das Reporting eine Datenbank in der Information über Reports, Jobs, Status, etc. gespeichert sind. Das Schema wird ebenfalls für jede Implementierung verwendet werden können.

Folgende Tabellen werden für die neue Reporting-Lösung benötigt:

- **JOBS:** Enthält Informationen über Jobs, die ausgeführt werden sollen
- **SCHEDULE:** Enthält Informationen über Zeiten, zu denen ein Job ausgeführt werden soll.
- **REPORT:** Das Report Template, das von der Datenbank geladen werden soll
- **REPORT_PARAMS:** Allfällige Parameter die für die Ausführung benötigt werden.
- **REPORT_CATEGORY:** Wird für die Kategorisierung der Reports benötigt.
- **REPORT_TYPE:** Definiert den Typ eines Reports (Modul-Zugehörigkeit)

Mit diesen Tabellen können die Anforderungen in den User Stories US-02, US-03, US-04, US-05 und US-08 erfüllt werden.

5.4 Scheduler

Die Recherche hat ergeben, dass die Verwendung des Spring Schedulers die beste Lösung ist, da das ganze Projekt Spring verwendet. Unabhängig von der Lösung muss dieser Scheduler verwendet werden, da ein Mechanismus benötigt wird, der regelmässig prüft, ob ein Job ansteht, der ausgeführt werden muss.

6 Konzept

In diesem Kapitel sollen drei Lösungsvarianten für das Kernproblem – die Aufbereitung der Daten – untersucht, und dann mittels einer Nutzwertanalyse eine Entscheidung für eine dieser Varianten getroffen werden.

6.1 Variante A: Multiple Data Sources in Report

Die Variante A sieht vor, dass in jedem Report Template die benötigten Datenbanken mittels separaten Datenquellen direkt angesteuert werden. Das heisst, dass die Daten direkt in den jeweiligen Report Templates definiert werden. Folgendes Schema soll geprüft werden:

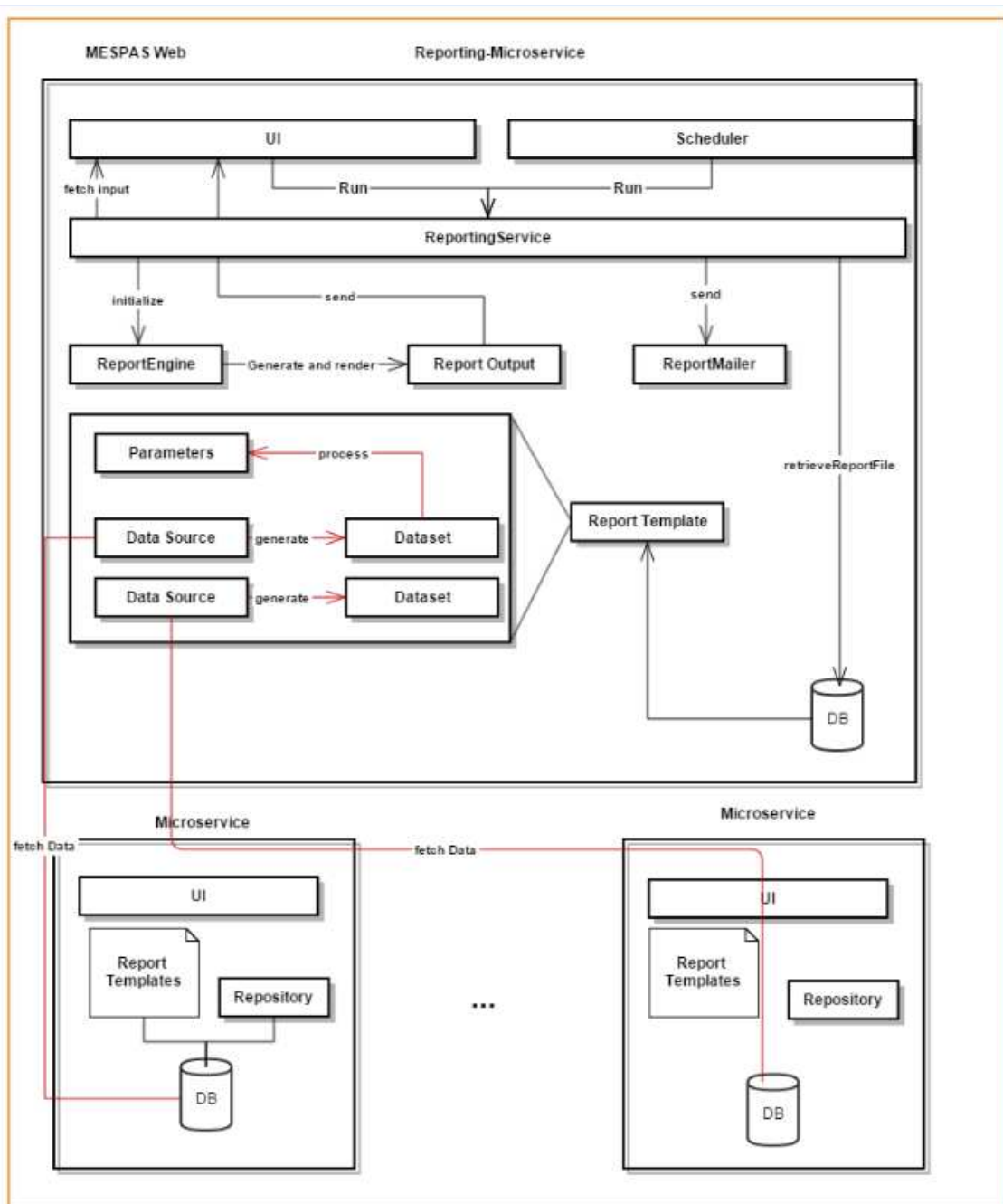


Abb. 28: Übersicht Variante A

6.1.1 Ablauf Variante A

Wie einleitend beschrieben werden die Daten in dieser Variante direkt im Report Template definiert. Wird ein Report manuell oder vom Scheduler ausgeführt, wird der Reporting-Service aufgerufen. Dieser lädt das Report Template von der Datenbank und initialisiert die ReportingEngine (-> BIRT Klasse). Argumente zur Eingrenzung der Daten werden als Report-Parameter an den Reporting-Service übergeben damit die diese im Dataset berücksichtigt werden können. Sind die Daten vorbereitet erstellt die Reporting-Engine unter Berücksichtigung des gewünschten Formats (PDF/Excel) das Output-File und liefert diese entweder zurück an das UI oder bereitet den Mailer vor, damit eine E-Mail versendet werden kann.

In den folgenden Unterkapiteln werden die einzelnen Schritte genauer erläutert.

6.1.2 Daten für Variante A

BIRT erlaubt es, mehrere Datenquellen für einen Report zu definieren. Im Designer Tool können unter „Data Source „ verschiedene Quellen angegeben werden. Dabei müssen die Typen der Datenquelle nicht denselben Typ (SQL, Scripts, o.ä.). aufweisen sondern es dürfen unterschiedliche solche verwendet werden.

Die Daten können dabei auf unterschiedliche Arten aufbereitet werden. Sollen die Daten mit SQL generiert werden, wird zunächst eine JDBC Data Source definiert. Dazu ist der entsprechende Treiber notwendig der als Bibliothek eingebunden wird. Alle notwendigen Parameter können direkt festgelegt werden^[6]:

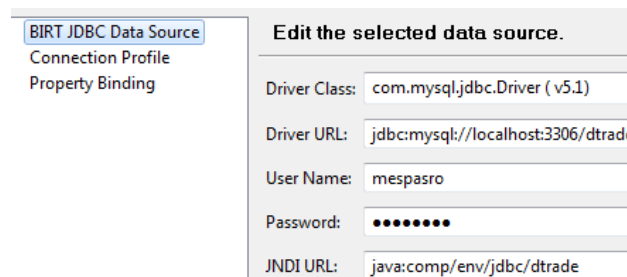


Abb. 29: SQL Datenquelle im BIRT Designer

Sobald die Datenquelle erstellt wurde wird das Dataset erstellt. In dieses Dataset wird die SQL Query die zur Aufbereitung der Daten dienen soll geschrieben:

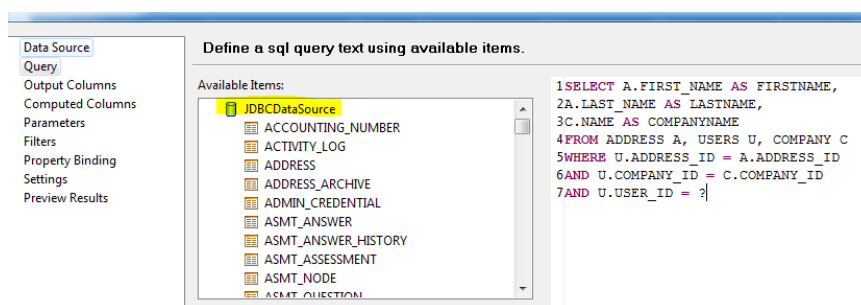


Abb. 30: SQL Dataset

In einer SQL Query werden die Spaltendefinitionen (Spaltenname und –Typ) automatisch erkannt und gesetzt. Somit ist das Dataset fertig erstellt und kann für das Layout verwendet werden.

Sollen die Daten über ein Script bereitgestellt werden muss eine sogenannte „Scripted Data Source“ bereitgestellt werden. Diese Quelle dient lediglich als Platzhalter für ein Dataset, es beinhaltet keine

Funktionen da die Verantwortlich für die Daten nicht bei der Datenquelle liegt. Im Dataset müssen dann die Spalten und der jeweilige Datentyp angegeben werden. Das Dataset verfügt also nicht über eine Query die automatisch aufgerufen wird, sondern die Daten müssen über ein Script in das Dataset geladen werden.

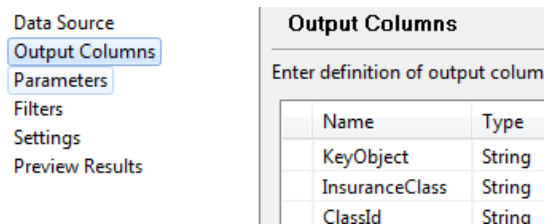


Abb. 31: Ausschnitt aus einem Scripted Dataset

Wie erwähnt müssen die Daten in einem Script aufbereitet werden. BIRT bietet die Möglichkeit, Java Packages direkt in ein Script einzubinden, sodass auf Java Klassen zugegriffen werden. Mittels einer Scriptsprache (Rhino) die Java sehr ähnlich ist können die Daten programmatisch vorbereitet werden.

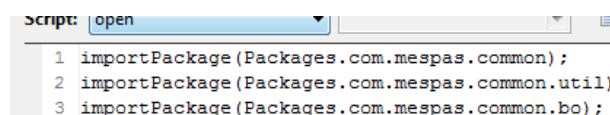


Abb. 32: Import von Java Packages in BIRT

Dabei erfolgt das Scripting einem von BIRT festgelegten Schema, da ein Script verschiedene Phasen durchläuft^[7]:

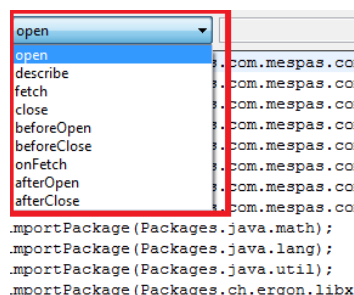


Abb. 33: Phasen eines BIRT Scripts

6.1.2.1 Initialisierung von Scripts

Zunächst wird ein Script initialisiert („open“). Darin werden üblicherweise die benötigten Packages importiert. Zudem sollte hier festgelegt werden, um welche Daten es überhaupt geht. Soll der Report beispielsweise alle Benutzer einer Firma anzeigen soll diese Liste hier geladen werden. Über einen Parameter kann die ID der Firma übergeben und so im Script verwendet werden.

Werden mehrere Einträge erwartet muss hier ein Zähler initialisiert werden:

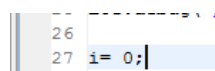


Abb. 34: Initialisieren eines Zählers

6.1.2.2 Abfüllen der Daten

In der Phase „*fetch*“ geschieht die eigentliche Aufbereitung der Daten. Man muss sich das Script Fenster als eine Art offene, programmatische Methode vorstellen. Es muss also immer etwas zurückgegeben werden. Im Fall von *fetch* ist das *true* oder *false*. Das folgende Listing zeigt, wie eine solche Datenaufbereitung funktionieren könnte und syntaktisch korrekt ist:

```
1 if (i < userList.size()) { // haben wir in "open" abgefüllt.  
2     user = userList.get(i);  
3     row["loginName"] = user.getLoginName();  
4     row["firstName"] = user.getFirstName();  
5     row["firstName"] = user.getLastName();  
6     i++;  
7     return true;  
8 }  
9 return false;
```

Listing 9: Daten-Aufbereitung mittels Script in BIRT

In diesem Beispiel werden Benutzerdaten in das Dataset geladen. Sobald das Script durchgelaufen ist, stehen die Daten wie in einem SQL Dataset zur Verfügung und können im Layout entsprechend eingefügt werden.

6.1.3 Report Parameter

In der Variante A müssen Argumente für die Daten über Report Parameter mitgegeben werden. Das heisst, dass beim Aufrufen des Reporting-Service die benötigten Argumente mitgegeben werden. Die API von BIRT bietet entsprechende Schnittstellen an. Im Report Template müssen diese Parameter dann ebenfalls eingebunden werden. Im Tool von BIRT kann dies mit wenigen Klicks bewerkstelligt werden:

Abb. 35: Report-Parameter erstellen

Wenn die Box „Is Required“ aktiv ist wird von der BIRT API eine Exception geworfen wenn kein gültiger Wert für diesen Parameter an die Reporting-Engine übergeben wird.

6.2 Variante B: Dataset populated dynamically

In der Variante B werden die Daten nicht im Report Template definiert und geladen sondern direkt im Quellcode. Dazu wird ein Objekt an eine Queue übergeben, welches die benötigten Datenbanken einzeln durchläuft und die Daten auf dem Weg lädt bis letztendlich sämtliche Daten in einem Objekt enthalten sind.

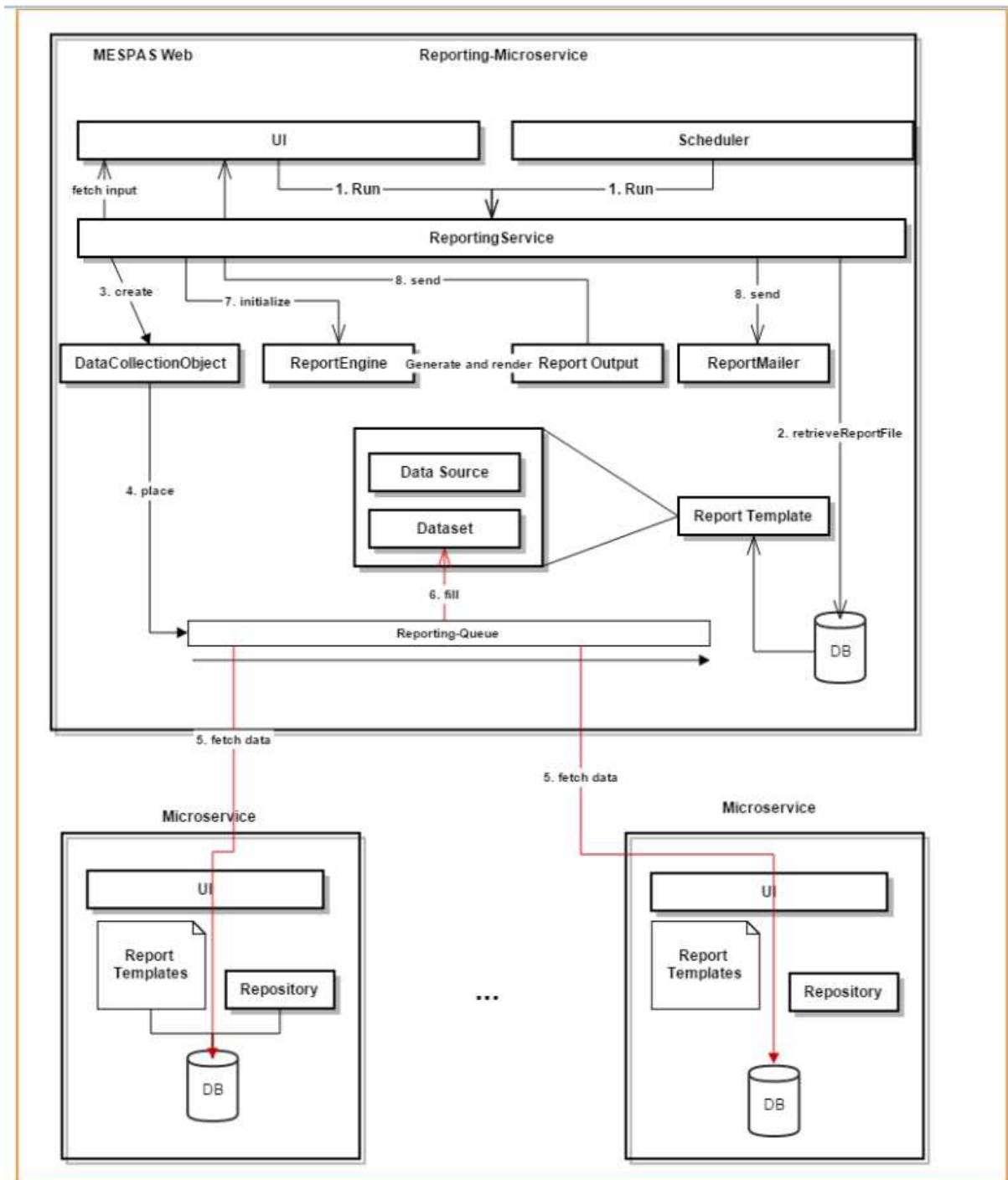


Abb. 36: Übersicht Variante B

6.2.1 Ablauf Variante B

Ein Request, der Informationen darüber enthält, welche Datenbanken und welche Daten davon benötigt werden wird an den Reporting-Service geschickt. Dieser erstellt ein Datenkollektorobjekt das in eine Queue gestellt wird. Diese Queue schickt das Objekt an die im Request enthaltene Datenbank und bestückt das Objekt mit den Daten. Sobald alle Datenbanken abgearbeitet wurden wird ein „Plain old Java Object“ (POJO) Objekt in das Report Template gestellt. So können die Daten dann im Report verwendet werden.

6.2.2 Daten für Variante B

In dieser Variante wird im Report Template nur eine Data Source und ein Dataset benötigt, da die Daten, die benötigt werden, in einem DataCollectionObject (DCO) definiert werden. Das DCO wird in eine Queue gestellt, welche die benötigten Datenbanken ansteuert und anhand der Information die im DCO vorhanden sind, die Daten lädt und in den Resultatteil des DCO schreibt. Aus diesem DCO wird dann ein POJO generiert, das alle benötigten Daten im Report Template abbildet und dort verwendet werden können.

6.2.2.1 DataCollectionObject

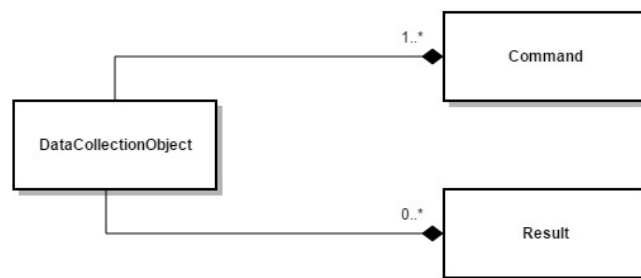


Abb. 37: DataCollectionObject

Das DataCollectionObject besteht aus zwei Hauptkomponenten: dem Befehl (command) und dem Resultat (result). Der Befehl ist eine Liste von (Unter-)Befehlen die abgearbeitet werden. Ein solcher Befehl benötigt folgende Informationen:

- service: der Microservice in dem die Information zu suchen ist
- key: die ID mit welcher die Information gesucht werden kann
- resultString: hier wird das Resultat hineingeschrieben.

Das Resultat ist das fertige Objekt das als POJO im Report verfügbar sein wird. Jedes benötigte Datenobjekt wie z.B. User, Organisation oder Order, wird eingetragen. Innerhalb dieser Objekte werden die Werte hineingeschrieben. Wenn wir also mit command1 zum Beispiel einen Benutzer von der Datenbank laden möchten sieht der command1 wie folgt aus:

```
getUser_name(UserService, 12345, "user.userName");
```

Listing 10: Beispiel-Befehl für das Laden eines Benutzernamen

Der UserService hört die Queue ab („listening“) und sieht so, dass er den Befehl getUser_name mit der ID 12345 ausführen und das Resultat in den vorgesehen Bereich „user.userName“ schreiben muss. In einem nächsten Befehl kann dann der Wert in „user.userName“ als Suchparameter verwendet und wiederum in einen neuen Resultateschlüssel geschrieben werden.

6.2.3 Report-Parameter

In dieser Variante können keine Parameter im Report-Template verwendet werden, da die Daten nicht durch das Template, sondern direkt durch das DCO aufbereitet werden. Die Verantwortlichkeit für das korrekte Filtern der Daten obliegt hier dem Reporting-Service. Es ist also bei der Implementierung eines Reports darauf zu achten, dass die benötigten Einschränkungen korrekt mitgeliefert werden.

6.3 Variante C: Data Warehouse

Die Variante C ist eine Mischform der Varianten A und B. Die Datenbanken sollen nacheinander abgearbeitet werden. Die Daten werden jedoch nicht als Objekt in den Report geladen sondern in der Reporting Datenbank abgelegt und die Reports laden die Daten aus dieser Datenbank. Sie dient also als Data Warehouse für die Reports.

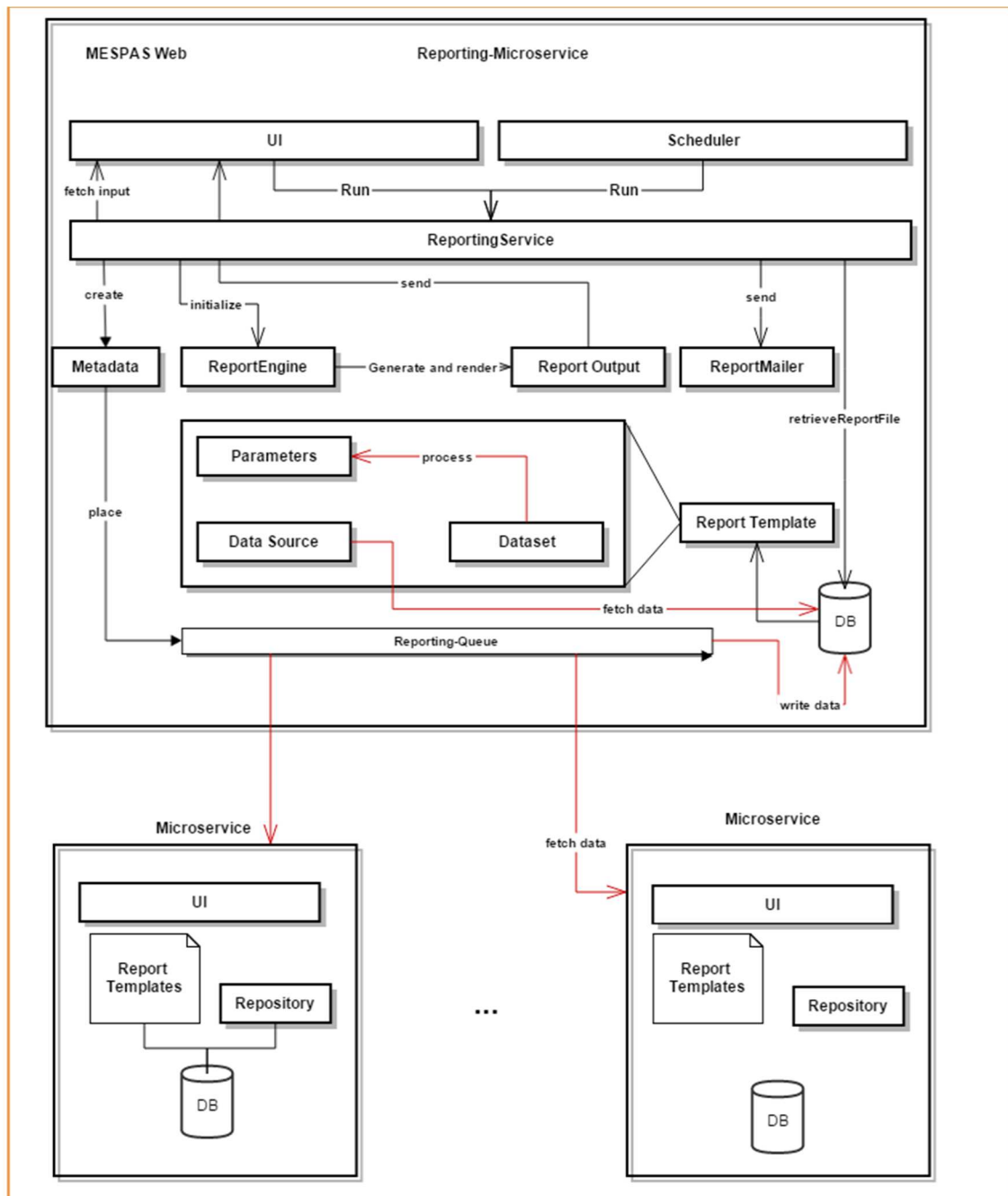


Abb. 38: Übersicht Variante C

6.3.1 Ablauf Variante C

Die Datendefinition für diese Variante erfolgt wie in der Variante A direkt im Report Template. Wenn der Report ausgeführt werden die Daten von der Reporting-Datenbank geladen und in das Dataset im Report Template geschrieben. Parameter können berücksichtigt und im Report Template definiert werden.

Die Daten müssen in Echtzeit abgefragt werden können. Daher muss gewährleistet sein, dass die Daten in der Reporting-Datenbank immer auf dem neusten Stand sind. Zu diesem Zweck wird immer ein Update der Datenbank durchgeführt bevor die Daten gelesen werden. Parallel dazu läuft ein Server-Task im Hintergrund, der ebenso regelmässig die Daten aktualisiert (ca. 1x pro Tag).

6.3.2 Daten Variante C

Wie bereits erwähnt sind werden die Daten in dieser Variante aus einem zentralen Data Warehouse geladen. Der Reporting-Service schreibt bei jeder Ausführung die benötigten Daten in das Data-Warehouse um die Aktualität der Daten zu gewährleisten. Dabei gibt es Datensätze von denen bereits im Voraus bekannt ist, dass sich diese nur sehr selten ändern. Dies ist bei der Implementierung zu berücksichtigen.

Für jeden Report muss im Report Template eine SQL Datenquelle definiert werden. Auf deren Basis wird ein Dataset erstellt in welche mittels einer SQL Query die Daten geschrieben werden.

6.3.3 Report Parameter Variante C

In dieser Lösung können Report Parameter direkt im Report Template verwendet werden (vgl. dazu Variante A). Da die Daten hier nur mit SQL und nicht programmatisch geladen werden müssen die Parameter allenfalls mittels eines Scripts angepasst werden, da optionale Parameter unterstützt werden können. BIRT stellt die dazu benötigten Funktionen bereits.

Beispiel: die Auswahl einer Installation ist optional. Somit muss die Query entsprechend gestaltet werden. Zunächst ist dabei zu prüfen, ob der Parameter einen Wert enthält:

```
if(params["pVessel"].value==null) {
```

Listing 11: Überprüfung eines Report-Parameters

Der Name dieses Parameters ist „pVessel“ und wurde im Report Template festgelegt. Wenn kein für den Parameter angegeben wurde muss dies für die Query berücksichtigt werden:

```
if(params["pVessel"].value==null) {  
    this.queryText = this.queryText + " ORDER BY ORDER_DATE, INST_NAME";  
} else if(params["pVessel"].value!=null) {  
    this.queryText = this.queryText + " AND INSTALLATION.INSTALLATION_ID =" +  
params["pVessel"].value + " ORDER BY ORDER_DATE, INST_NAME";  
}
```

Listing 12: Dynamische Anpassung einer Query mittels Script

So können optionale Parameter benutzerfreundlich eingebaut werden.

6.4 Zusammenfassung

Es wurden drei Varianten entworfen:

Variante A bietet den klassischen Ansatz mit der die Datensätze im Report Template definiert werden und die Daten direkt auf den jeweiligen Datenbanken angesteuert werden. Die Variante B sieht vor, dass die Daten dynamisch geladen werden und im Report Template alle Daten direkt in einem

einigen Dataset vorhanden sein werden. Dazu wird ein serialisiertes Objekt in eine Queue übergeben die die benötigten Datenbanken nacheinander besucht und die benötigten Daten in diesem Objekt speichert. Nachdem alle benötigten Daten geladen wurden werden diese in das Dataset geschrieben und können im Report direkt verwendet werden. Die Variante C ist eine Mischform der beiden ersten Lösungen. Die Daten werden vom Reporting-Service in ein Data-Warehouse geschrieben und sind so für die Report-Templates verfügbar um mittels SQL Query geladen zu werden. Um Echtzeit-Daten zu gewährleisten müssen die Daten jeweils bei der Ausführung in die Datenbank geschrieben werden.

Da diese drei Varianten nun erarbeitet wurden muss nun analysiert werden, welche Lösung implementiert werden soll. Dazu soll eine Nutzwertanalyse für die drei Möglichkeiten durchgeführt werden. Im nächsten Kapitel wird die NWA durchgeführt und so eine Entscheidung getroffen werden können.

6.5 Nutzwert-Analyse Varianten A, B und C

Wie in Kapitel für die Recherche muss nun auch hier eine Nutzwertanalyse durchgeführt werden, um eine Entscheidungsgrundlage darüber zu haben, welche Variante umgesetzt werden soll.

Dazu muss zunächst ermittelt werden, nach welchen Kriterien die Nutzwerteanalyse durchgeführt werden soll.

Das Nutzwert-Analyseverfahren ist das Gleiche wie in Kapitel 4 für die Recherche, allerdings mit anderen Bewertungskriterien.

6.5.1 Bewertungsschema

Das Projekt soll die im Kapitel 3 beschriebenen Anforderungen erfüllen. Daher müssen diese bei der Analyse berücksichtigt werden. Dabei werden die Kriterien in folgende Unterkriterien eingeteilt:

- Abdeckung der Funktionalität
- Komplexität der Umsetzung
- Aufwand Implementierung einzelner Reports

Folgendes Punkteschema findet hier Anwendung:

- 0 Punkte: Anforderung kann unmöglich erfüllt werden / unbrauchbar
- 1 Punkt: Anforderung kann nur unter grösstem Aufwand erfüllt werden / schlecht
- 2 Punkte: Anforderung kann unter grossem Aufwand erfüllt werden / ungenügend
- 3 Punkte: Anforderung kann mit etwas Aufwand erfüllt werden / akzeptabel
- 4 Punkte: Anforderung kann unter kleineren Anpassungen erfüllt werden / gut
- 5 Punkte: Anforderung kann ohne grössere Probleme erfüllt werden / sehr gut

Nachfolgend erfolgt eine Evaluation von Kriterien die das Konzept zu erfüllen hat. Dabei sei zu beachten, dass diese Analyse sich auf das Kernproblem dieser Arbeit, also die Aufbereitung der Daten, bezieht. Dies sind die User Stories US-01, US-05, US-09, US-11 und US-12.

6.5.2 Datenkomplexität

Ein wichtiges Kriterium für die neue Lösung ist die Datenkomplexität. Ein Report muss in der Lage sein, Daten abzubilden. Daher ist es wichtig, dass die neue Lösung die Daten korrekt aufbereiten kann, ohne dass die Aufbereitung zu komplex wird.

In der Variante A liegt die Datenaufbereitung im Report selbst. Werden SQL Daten benötigt ist nur eine Query notwendig und falls Daten programmatisch aufbereitet werden müssen kann die mit

einem Script bewerkstelligt werden. Diese Art der Aufbereitung ist einfach, erfordert allerdings mehr Aufwand bei der Implementierung der Reports: Bewertung: 4 Punkte.

In der Variante B müssen die Daten von den einzelnen Microservices bereitgestellt werden. Das DataCollectionObject arbeitet alle benötigten Datenbanken ab, um die Daten zu laden. Dieser Mechanismus erhöht die Komplexität im Vergleich zur direkten Datenaufbereitung. Bewertung: 3 Punkte.

In der Warehouse Variante C werden die einzelnen Daten von den Microservice bereitgestellt und in eine zentrale Reporting Datenbank geschrieben. Dadurch, dass einerseits sichergestellt werden muss, dass alle Datenbank besucht werden und das Schreiben in die Datenbank auch fehlerbehaftet sein kann ist gibt es hier Abzüge für die Komplexität. Bewertung: 4 Punkte.

6.5.3 Datensicherheit

In der User Story US-12 wird eine sogenannte Mandantentrennung verlangt. Das bedeutet, dass die Daten der einzelnen Kunden nicht für andere sichtbar sein dürfen. Die Aufbereitung der Daten muss daher so ausgelegt werden, dass Benutzer die Daten anderer Nutzer unter keinen Umständen sehen können.

In der Variante A muss die Mandantentrennung innerhalb der Queries gewährleistet werden. Daher muss bei der Implementierung der Reports stets darauf geachtet werden, dass die Variante eingehalten wird. Dies ist trivial erhöht jedoch die Fehleranfälligkeit. Bewertung: 3 Punkte.

In der Variante B erfolgt die Mandantentrennung direkt während der Datenaufbereitung. Im Reporting-Service wird ein entsprechender Mechanismus eingebaut, der verhindert, dass unautorisierte Daten geladen werden können. Dies ist etwas komplexer in der Umsetzung, verhindert jedoch, dass in jedem Report die entsprechende Implementierung vorgenommen werden muss. Bewertung: 4 Punkte.

Die Variante C lädt die Daten direkt aus dem Warehouse. Das heisst, dass die Mandantentrennung entweder beim Schreiben in die Datenbank oder während der Aufbereitung vorgenommen werden muss. Das erhöht die Komplexität, kann aber den Implementierungsaufwand für die Reports reduzieren. Bewertung: 4 Punkte.

6.5.4 Report Parameter

Report müssen Argumente entgegennehmen können, um die Daten einzuschränken. Report Templates haben die Möglichkeit Parameter als Platzhalter entgegen zu nehmen die zur Laufzeit entsprechend eingesetzt werden.

In der Variante A werden Parameter direkt im Report verwendet. In SQL Queries werden können diese ohne Scripts direkt verwendet werden. Werden Daten programmatisch aufbereitet können Parameter über ein Script genutzt werden. Diese Art der Report-Parameter sind einfach zu nutzen, haben aber im Vergleich zu anderen Report Parametern den Nachteil, dass Listen von Argumenten (z.B. eine Liste von Schiffen) nur über ein Script verwendet werden kann, was einen Report wieder ziemlich fehleranfällig macht. Bewertung: 3 Punkte.

In der Variante B werden die Parameter nicht im Report direkt verwendet. Parameter können dennoch vom Benutzer genutzt werden, jedoch erfolgt das Filtern der Daten direkt im entsprechenden Microservice. Das erlaubt maximale Flexibilität erhöht aber den Komplexitätsgrad etwas. Bewertung: 4 Punkte.

In der Variante C kommen beide Formen vor. Damit die Daten entsprechend in die Datenbank geschrieben werden können müssen diese bereits wie in Variante B im Microservice gefiltert werden. Für den Report selbst können dann weitere Parameter direkt auf SQL Ebene wie in Variante A verwendet werden. Das erlaubt eine gewisse Flexibilität, macht das Ganze aber auch fehleranfällig. Zudem ist die Komplexität erhöht. Bewertung: 4 Punkte.

6.5.5 Spezielle Daten

Im Hinblick auf die Erweiterbarkeit muss die Lösung auch in der Lage sein, Spezielle Daten zu verarbeiten, zum Beispiel aus einer Textdatei oder Daten aus einem nicht-relationalen Microservice. Da weitere Module in MESPAS Web geplant sind die ihre Daten teilweise nicht aus einer Datenbank beziehen muss die neue Lösung dafür ausgerüstet sein.

Die Variante A bietet keinerlei Möglichkeiten auf andere Datenquellen als die von BIRT angebotenen zuzugreifen. Daher müssten die Daten für jeden Report programmatisch erstellt werden, was diese Variante praktisch unbrauchbar macht. Bewertung: 0 Punkte.

Da die Daten in der Variante B dynamisch geladen werden ist eine spezielle Datenquellen ohne weiteres einsetzbar. Ein Service mit speziellen Daten muss dazu die Queue abhören und die Daten in das Dataset abfüllen. Diese Variante erlaubt maximale Flexibilität. Bewertung: 5 Punkte.

In der Warehouse Variante können die Daten programmatisch aufbereitet werden. Jedoch müssen diese in die Datenbank geschrieben werden, was die Komplexität stark erhöht. Bewertung: 3 Punkte.

6.5.6 Redundanz

Datenredundanz ist immer dann ein Thema, wenn Datenbanken zum Einsatz kommen. Auf der einen Seite soll das Risiko eines Datenverlustes minimiert, auf der Daten so wenige Daten wie möglich auf Vorrat gespeichert werden. Auch wenn nur Daten gelesen werden, ist es möglich, dass dennoch Daten produziert werden. In diesem Abschnitt werden die einzelnen Varianten auf die Datenredundanz, die sie verursachen, überprüft werden. Dabei ist zu beachten, dass in diesem Zusammenhang nur von der Ausführung der Reports die Rede ist und nicht von allfälliger Speicherung von Dokumenten.

In der Variante A werden nur Daten gelesen. Da Informationen über Parameter und Datenquellen statisch festgelegt sind, werden keine weiteren Daten produziert. Daher verursacht diese Variante keine Redundanz. Bewertung: 5 Punkte.

In der Variante B werden Daten flüchtig produziert. Das bedeutet, dass gewisse Informationen in das DataCollectionObject geschrieben werden müssen, diese jedoch nicht auf den Massenspeicher geschrieben werden. Daher verursacht auch diese Variante keine Redundanzen. Bewertung: 4 Punkte.

Die Variante C liest und schreibt ständig Daten. Da ständig Daten in die Reporting-Datenbank geschrieben werden sind diese mehrfach vorhanden. Daher ist die Redundanz hier hoch. Bewertung: 2 Punkte.

6.5.7 Performance

In diesem Abschnitt soll die Performance der einzelnen Varianten bewertet werden, da es wichtig ist, dass die Daten rasch und zuverlässig aufbereitet werden können.

Die beiden ersten Varianten laden die Daten jeweils direkt aus der Datenbank. In Variante kann dies direkt mit SQL geschehen, während in der Variante die Daten über den Microservice geladen wer-

den. Jedoch ist auch in dieser Variante SQL im Hintergrund im Einsatz. Unter dem Strich verhalten sich beide Varianten ähnlich was die Performance angeht. Bewertung: je 4 Punkte.

In der Variante werden bekanntlich auch Daten in die Datenbank geschrieben. Das heisst, dass die Daten immer zuerst geladen, dann geschrieben und dann von aus der zentralen Datenbank geladen werden. Dies kann je nach Komplexität der benötigten Informationen eine längere Zeit in Anspruch nehmen. Bewertung: 3 Punkte.

6.5.8 Bestehendes System

Die neue Reporting-Lösung ist innerhalb eines bestehenden Systems zu implementieren. Daher muss die Lösung damit kompatibel sein. Zudem ist der Aufwand für die Realisierung ein Faktor, den es zu berücksichtigen gilt.

Für die Variante A ist bereits alles, was benötigt wird, vorhanden. Allerdings erlaubt BIRT nur eine Dataset pro Tabelle, was es schwierig macht, Daten aus verschiedenen Tabelle mit vernünftigem Aufwand darzustellen. Daher wird bei den meisten Reports auf Scripts zurückgegriffen werden müssen. Bewertung: 3 Punkte.

Variante B ist etwas komplexer in der Umsetzung bietet aber absolute Flexibilität. Alles, was benötigt wird, ist vorhanden. Im System ist bereits ein Framework für Messaging vorhanden (-> RabbitMQ), die auch für diese Variante verwendet werden kann. Der Aufwand für die Implementierung von Reports ist etwas grösser als in Variante A. Bewertung: 4 Punkte.

Variante C erfordert den grössten Aufwand. Zusätzlich zur Queue muss auch ein Data-Warehouse gebaut und entsprechend Queries entwickelt werden. Jedoch erlaubt auch diese Lösung eine grosse Flexibilität und Reports sind einfach zu implementieren, da auf die Daten des Warehouse zugegriffen werden kann. Bewertung: 3 Punkte.

6.5.8 Entscheidungstabelle

Die vergebenen Punkte für die Kriterien ergeben folgende Matrix:

Kriterium	Gewicht	Bewertung A (Multiple Data-set)	Bewertung B (Dynamic)	Bewertung C (Warehouse)	Gewichteter Teilnutzen A	Gewichteter Teilnutzen B	Gewichteter Teilnutzen C
Datenkomplexität	40%	4	3	4	1.6	1.2	1.6
Datensicherheit	15%	3	4	4	0.45	0.6	0.6
Report-Parameter	5%	3	4	4	0.15	0.2	0.2
Spezielle Daten	10%	0	5	3	0	0.5	0.3
Redundanz	5%	5	4	2	0.25	0.2	0.1
Performance	10%	4	4	3	0.4	0.4	0.3
Bestehendes System	15%	3	4	3	0.45	0.6	0.45
	100%						
					Nutzwert A	Nutzwert B	Nutzwert C
					3.3	3.7	3.55

Tabelle 3: Nutzwert-Tabelle Lösungsvarianten

Aus dieser Matrix ist abzulesen, dass die Variante A deutlich ausscheidet. Enger ist es zwischen den Varianten B und C. C ist in der Praxis öfter anzutreffen, weil die Daten zentral aufbereitet werden können. Allerdings führen die Redundanz und betriebliche Einschränkungen dazu, dass auch diese Variante ausscheidet. Somit bleibt nach der Analyse und Berücksichtigung der betrieblichen Faktoren nur noch die Variante B, die auch die höchste Punktzahl erreicht hat.

6.6 Weitere Analyse Variante B

Auf Basis der Nutzwertanalyse fiel die Entscheidung auf Variante B: „*Populate Dataset dynamically*“ da diese bezüglich Aufwand, Redundanz, Flexibilität und Kosten am besten in die bestehende Lösung passt.

In diesem Kapitel wird das endgültige Design nun anhand eines Klassen- und eines Sequenzdiagramms festgelegt.

6.6.1 Queue

Als Queue wird in MESPAS Web RabbitMQ verwendet. RabbitMQ erlaubt das Versenden und Lesen von Nachrichten innerhalb von Applikationen und unterstützt viele Plattformen^[8]. Die Implementierung in Java ist relativ einfach zu bewerkstelligen. Das Konzept für die neue Lösung sieht eine Queue für das Publizieren von Objekten vor, die Daten sammeln sollen. Da jeder Microservice potenziell für einen Befehl zuständig ist müssen alle Services sämtliche Nachrichten erhalten.

RabbitMQ stellt für diesen Zweck einen sogenannten Exchange bereit^[9]. Ein Exchange empfängt eine Nachricht und leitet diese an sämtliche Queue, die sich beim Exchange registriert haben weiter (publish) und so kann jeder Befehl vom jeder Queue konsumiert (consume) und weiter verarbeitet werden. Daher ist dieses Prinzip bei der Implementierung der Queue zu verwenden.

6.6.2 Datenaufbereitung

Der Kern der Lösung ist die Datenaufbereitung. Es muss möglich sein, Daten aus verschiedenen Datenbanken zu laden und weiterzuverarbeiten. Das Konzept der Variante B basiert auf der Idee, dass die Datenbank einzeln „besucht“ werden können. Nachfolgend ist der Weg, den ein solches DataCollectionObject nehmen soll, schematisch beschrieben:

Sequenzdiagramm Datenaufbereitung

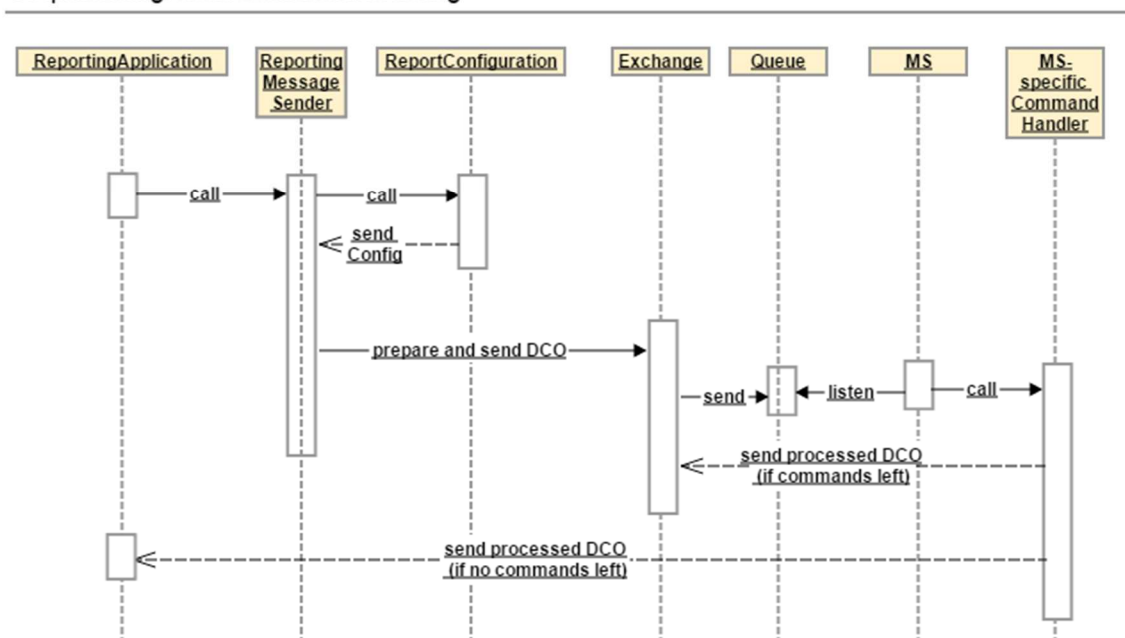


Abb. 39: Ablauf Datenaufbereitung

Wie auf Abb. 39 zu sehen ist startet der Reporting-Microservice die Datenaufbereitung. Wird ein Report ausgeführt, so wird ein report-spezifischer MessageSender aufgerufen. Das bedeutet, dass für jeden Report eine eigene solche Klasse erstellt werden muss. Diese Klasse erstellt zunächst das

DataCollectionObject (DCO). Danach wird die Klasse „*ReportConfiguration*“ aufgerufen. Dabei handelt es sich um eine Utility-Klasse in der die Konfiguration für sämtliche Reports vorhanden ist. Die Konfiguration liefert die benötigten Befehle zurück die als eine verkettete Liste (LinkedList) von Command Objekten zum DataCollectionObject hinzugefügt wird. Dies muss eine verkettete Liste sein, dass die einzelnen Befehle sequenziell gemäss der Konfiguration aufgerufen werden müssen. Nun muss das DCO auf die Reise geschickt werden. Der MessageSender sendet das DCO an den bereitgestellt werden Exchange welcher das DCO an sämtliche Microservices schickt, die eine Queue an den Exchange gehängt haben. Erhält ein solcher Microservice die Nachricht, überprüft dieser, ob sie für ihn ist. Ist dies nicht der Fall ignoriert er die Meldung. Ist die Nachricht für ihn bestimmt, so nimmt er den ersten Befehl entgegen und leitet ihn an einen microservice-spezifischen CommandHandler, der den Befehl verarbeitet und das Resultat in das DCO schreibt. Danach wird der Befehl aus der verketteten Liste gelöscht. Sind weitere Befehle vorhanden, sendet der Handler das DCO wieder an den Exchange, damit derselbe oder andere Microservices die weiteren Befehle verarbeiten können. Ist der ganze Zyklus abgeschlossen wird das DCO an den ReportingMessageSender geschickt, der ein PJO erstellt, das im Report verwendet werden kann.

6.6.3 Sequenzdiagramm

Sequenzdiagramm Reporting

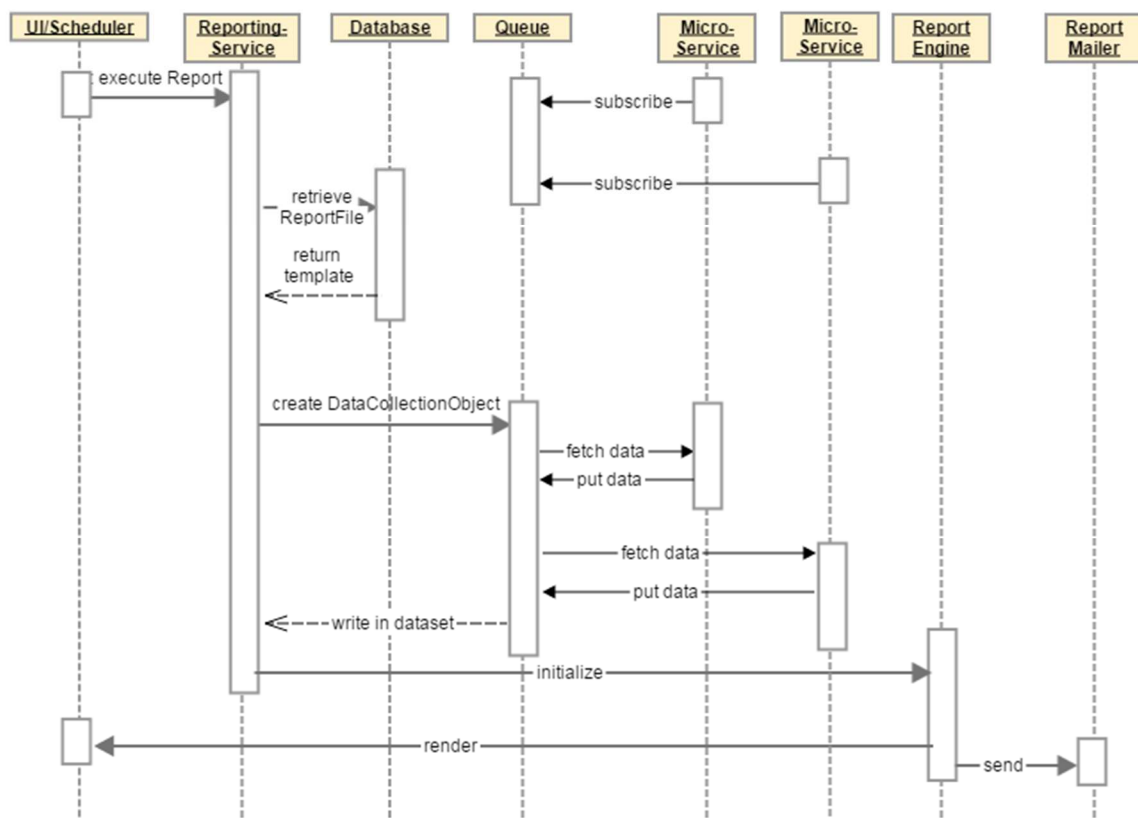


Abb. 40: Sequenzdiagramm Variante B

6.6.4 Klassendiagramm

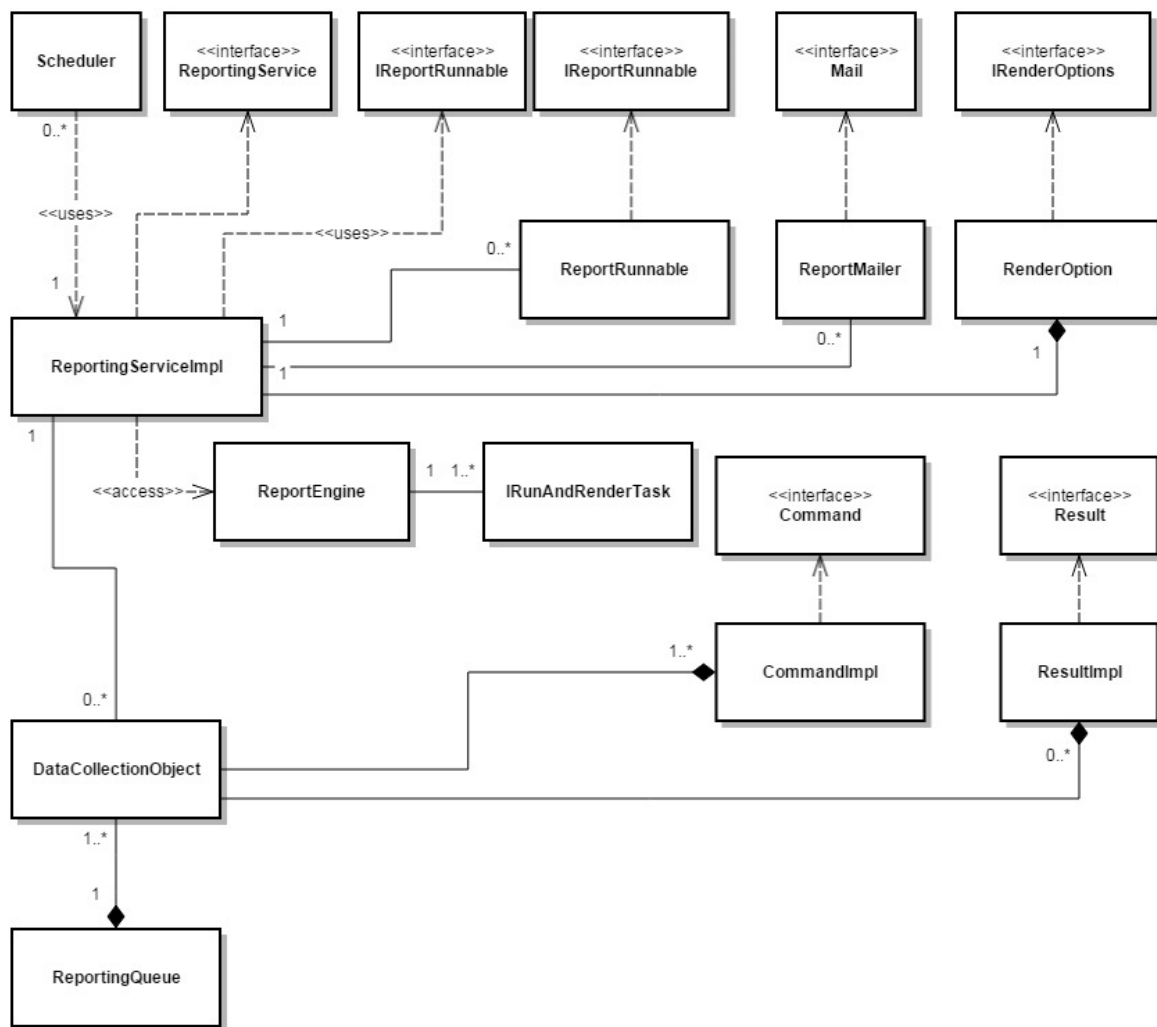


Abb. 41: Klassendiagramm Variante B

7 Proof of Concept

7.1 Bau eines Prototypen

7.1.1 Einleitung

Da eine Entscheidung für ein Konzept getroffen und dieses etwas vertieft wurde muss ein Prototyp geschrieben werden. Ziel des Prototypen ist es, das die Funktionsfähigkeit des Konzeptes zu untersuchen, und idealerweise zu beweisen (-> Proof of Concept). Zu diesem Zweck soll ein Report erstellt werden, der eine Art Visitenkarte darstellt. Die Daten dazu sollen vollständig mit der neuen Lösung aufbereitet werden. Das bedeutet, dass anhand einer Id, die vorgegeben wird, die Benutzerdaten wie Name, Vorname, Adresse, PLZ und Ort von der Datenbank geladen und an den Report übergeben werden sollen, der eine Visitenkarte als PDF Datei erstellt. In diesem Kapitel soll nun der Bau dieses Prototyps schrittweise beschrieben werden.

7.1.2 Abgedeckte Anforderungen

In diesem Prototyp wird nur eine kleine Teilmenge der Anforderungen abgedeckt, da die meisten Anforderungen das komplette System und nicht nur den Kern der Lösung betreffen. Der Kern der Lösung und somit der Prototyp soll folgende Anforderungen abdecken, oder zumindest erläutern:

- R-001: Report ausführen
- R-004: Report Formate
- R-013: Report Parameter – allgemein
- R-014: Report Parameter – optionale Parameter
- R-020: Daten - Datenbank
- R-021: Daten - Datenobjekte

7.1.3 Bereitstellen der Umgebung

Da die neue Reporting-Lösung in eine bestehende eingebaut werden soll, muss keine ganze neue Umgebung erstellt werden. Dennoch gibt es einige Einstellungen vorzunehmen und es muss ein neues Unter-Projekt angelegt werden.

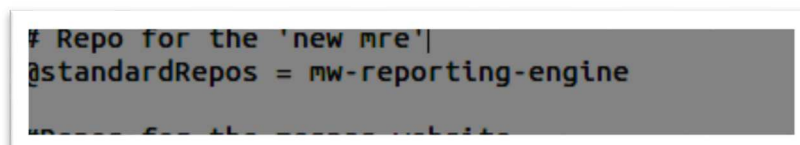
7.1.3.1 Git Repository

Sämtliche Projekte im Unternehmen des Auftraggebers werden auf einem Git-Server gehostet. Die Verwaltung dieser Repositories erfolgt über gitolite, das wiederum ein git Repository. Daher muss zunächst gitolite ausgecheckt werden:

```
severin@mespas:~/git$ git clone git@mancini.mespas.com:gitolite-admin.git
```

Listing 13: Auschecken von gitolite

Danach muss in der Konfiguration von gitolite ein neuer Eintrag für das neue Repository eingetragen werden.



```
# Repo for the 'new mre'|
standardRepo = mw-reporting-engine
```

Abb. 42: Neuer Eintrag in Gitolite-Config

Sobald dieser Eintrag erstellt wurde können die Änderungen gepusht werden.

```
severin@mespas:~/git/gitolite-admin/conf$ git add gitolite.conf
severin@mespas:~/git/gitolite-admin/conf$ cd ..
severin@mespas:~/git/gitolite-admin$ git commit -m "Added mw-reporting-engine to gitolite.conf"
[master 6c0b75e] Added mw-reporting-engine to gitolite.conf
1 file changed, 3 insertions(+)
severin@mespas:~/git/gitolite-admin$ git push origin master
Counting objects: 7, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 418 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Initialized empty Git repository in /apps/git/repositories/mw-reporting-engine.git/
To git@mancini.mespas.com:gitolite-admin.git
8b76356..6c0b75e master -> master
severin@mespas:~/git/gitolite-admin$
```

Listing 14: Änderungen gitolite-Config pushen

Nun kann das Git Repository mit „git clone“ gecloned werden.

7.1.3.2 Benötigte MESPAS-Web Projekte

Damit die neue Reporting-Engine in der MESPAS-Web Umgebung funktionieren kann, werden einige Unterprojekte benötigt. Diese sind:

- mw-base: Basis-Projekt
- mw-common-domain: Projekt für Ressourcen die MS-übergreifend benötigt werden.
- mw-microservice-parent: Hält die Hauptklasse für die Microservices
- mw-gateway: Service um Requests zwischen den einzelnen Komponenten von MESPAS Web weiterzuleiten

Diese Unterprojekte müssen mit git clone ausgecheckt werden. Später werden weitere Microservices benötigt, da in diesen die Datenbanken für die in den Reports benötigten Daten gespeichert sind.

7.1.3.3 Eclipse

Das Projekt wird in Java umgesetzt und Eclipse als Integrierte Entwicklungsumgebung (Integrated Development Environment, IDE) verwendet. Alle MESPAS Web Unterprojekte sind mit **maven** zu erstellen. Daher muss ein neues Maven Projekt angelegt werden. Dafür wird zunächst eine neue Konfigurationsdatei (pom.xml) erstellt. Dazu kann eine bereits existierende Datei verwendet werden, da alle Dateien dieselbe Struktur aufweisen. Diese Struktur sieht (schematisch) wie folgt aus:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <parent>
    <groupId>com.mespas.web</groupId>
    <artifactId>mw-base</artifactId>
    <version>0.0.1-SNAPSHOT</version>
  </parent>
  <artifactId>mw-reporting-engine</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Mespas Reporting Engine</name>
  <description>Component for running and scheduling reports</description>
  <dependencies>
    <!-- wird gleich näher erläutert -->
  </dependencies>
</project>
```

Listing 16: XML Struktur für Maven-Projekt

Im Element **parent** wird das Hauptprojekt festgelegt, in diesem Fall mw-base, das die Basis für das gesamte MESPAS Web Projekt bildet. Das Element **artifactId** bestimmt des Namen des Artefaktes, das während des Build-Prozesses erstellt wird. Mit **version**, **name** und **description** werden Basis-Informationen über das Projekt festgelegt.

Im Element **dependencies** können Verweise auf andere Projekte oder Bibliotheken eingefügt werden. Dies wird im Verlaufe der Entwicklung des Projektes notwendig werden.

Sobald das pom.xml erstellt wurde, kann das Projekt in Eclipse importiert werden:

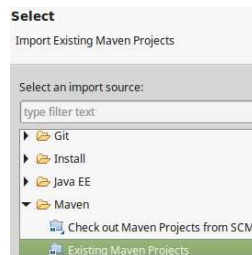


Abb. 43: Projekt in Maven importieren

7.1.3.4 Test-Datenbank

Für jeden Microservice muss eine Testdatenbank eingerichtet werden. Vom Hauptprojekt wird dafür ein Script bereitgestellt, das automatisch ausgeführt wird. Folgende SQL-Query muss in diesem Script eingetragen werden:

```
CREATE DATABASE
IF NOT EXISTS mw_reporting_engine CHARACTER SET utf8 COLLATE utf8_general_ci;
```

Listing 17: SQL-Query zur Erstellung der Reporting-Datenbank

7.1.3.5 Infrastruktur

Um MESPAS WEB auf einer lokalen Maschine laufen zu lassen wird Docker benötigt. Eine komplett lauffähige Umgebung steht bereits bereit und kann mittels einem Script gestartet werden. Daher wird an dieser Stelle nicht näher darauf eingegangen.

7.1.3.6 Dependency Injection

Einer der Vorteile am Arbeiten mit Spring ist die Fähigkeit, benötigte Variablen automatisch zu initialisieren und zuzuweisen. In einer Konfigurationsdatei (application.yml) können benötigte Werte eingestellt werden, die dann mittels einer Java-Annotation verwendet werden können.

7.1.4 Bereitstellen des Reporting-Microservice

Für die Reporting-Engine muss ein neuer Microservice erstellt werden. Dieser wird verantwortlich für das Scheduling und die Verwaltung der Reports sein. Für den Prototypen ist dieser jedoch nur von untergeordneter Bedeutung. In der Microservice Architektur stellt jeder Service eine eigene Applikation, die gestartet werden muss. Jeder Microservice erbt von der Klasse *MicroServiceApplication*, die im Unterprojekt „mw-microservice-parent“ definiert ist.

Um also den Microservice für die neue Reporting-Engine zu erstellen muss eine Main-Klasse erstellt werden, die von *MicroServiceApplication* erbt. Mit der Methode *setupAndRun* wird der Microservice mittels dem Spring-Framework hochgefahren und gestartet.


```

public class ReportingEngineApplication extends MicroserviceApplication {

    private static final Logger LOG = LoggerFac-
toy.getLogger(ReportingEngineApplication.class);

    public static void main(String[] args) {
        setupAndRun(ReportingEngineApplication.class, args, "mw-reporting-engine");
    }

    @Override
    protected Logger getLog() {
        return LOG;
    }
}

```

Listing 18: Starten des Reporting-Microservice

Zunächst müssen der Docker Container und die Microservice mw-common-ui und mw-gateway gestartet werden. Danach werden weitere Microservice nach Bedarf gestartet, darunter auch der Reporting-Microservice:

```

ExceptionHandlerExceptionResolver {257} - Looking for exception mappings: org.sprui
ExceptionHandlerExceptionResolver {267} - Detected @ExceptionHandler methods in con
- Started ReportingEngineApplication in 11.431 seconds (JVM running for 12.277)

```

Abb. 44: Gestarteter Reporting-Microservice

Grundsätzlich müssen alle Microservices, aus denen benötigt werden, gestartet werden, da diese die Befehle für das Laden der Daten liefern.

7.1.5 Vorbereitung DataCollectionObject

Reports sind nutzlos ohne Daten. Daher müssen diese nun aufbereitet werden, um sie in den Reports verwenden zu können. Daher folgt nun die Implementierung des DataCollectionObjects. Wie im Konzept beschrieben besteht das DataCollectionObject aus mehreren Komponenten. Benötigt werden Befehle und ein Objekt um die Ergebnisse zu speichern, die dann an den Report als POJO übergeben werden.

7.1.5.1 Command

Das DataCollectionObject muss eine Reihe von Befehlen aufnehmen können, die dann von den betroffenen Microservices ausgeführt werden sollen. Daher sollte die Implementierung als Liste erfolgen, die sequenziell abgearbeitet wird. Ein Befehl benötigt folgende Informationen:

- der Name der Methode, die ausgeführt werden soll
- der Key, mit dem der Datensatz identifiziert wird (z.B. eine ID oder ein Zwischenresultat von vorhergehenden Befehlen)
- der betroffene Microservice
- der Pfad des Zwischenresultates, da diese Information evtl. später benötigt wird
- die Klasse des Resultates
- eine Information darüber, ob ein Zwischenresultat oder ein fix definiert Key für die Suche verwendet wird.

Die Implementierung dieser Klasse erfolgt als herkömmliches Java Objekt in der keine Logik vorhanden ist, sondern die Daten lediglich gekapselt werden.

7.1.5.2 Result

Bei jeder Ausführung eines Befehls muss das Resultat im DCO gespeichert werden. Um komplexe Datentypen und den Resultate-Pfad berücksichtigen zu können muss ein entsprechendes Objekt

gebaut werden. Die Art dieses Objektes unterscheidet sich von Fall zu Fall. Für jeden Report muss eine neue Klasse für das Resultat erstellt werden, da die Art der Informationen die gespeichert werden müssen jeweils unterschiedlich ist. Um diese Verschiedenartigkeit, also Polymorphie^[10], zu unterstützen hat die Instanzvariable für das Resultat im DCO den Typ `ReportResult`. `ReportResult` ist lediglich ein Wrapper der ein Objekt vom Typ `Object` hält. So kann ein Resultat eines beliebigen Typs im DCO abgelegt werden. Das Resultat muss dann allerdings in einer konkreten Klasse implementiert werden. Dazu folgt später mehr.

7.1.5.3 Entwurf DCO

Nachdem die Befehle und das Resultat modelliert wurden kann ein erster Entwurf für das DCO erstellt werden:

```
private ReportResult result;  
private LinkedList<Command> commands;
```

Listing 19: Erster Entwurf DCO (Ausschnitt)

7.1.6 Report Konfiguration

Damit das DCO weiss, welche Befehle ausgeführt werden sollen, wird eine Konfiguration benötigt, das Command Objekt im DCO abfüllt. Für jeden Report wird eine eigene Methode erstellt, die eine verkettete Liste von Befehlen zurückliefert, welche im DCO gesetzt wird. In einer `HashMap` wird diese Liste und ein Identifier als String abgelegt, über die die korrekte Liste zurückgegeben werden kann. Folgendes Beispiel soll das Prinzip verdeutlichen:

```
public class ReportingConfiguration {  
    // Config Map  
    protected final Map<String, LinkedList<Command>> reportConfigMap;  
  
    // Lists used in Reports  
    protected LinkedList<Command> testReportCommands;  
  
    private Command testCommand;  
  
    public ReportingConfiguration() {  
        reportConfigMap = new HashMap<String, LinkedList<Command>>();  
    }  
    public void prepareTestCommands() {  
  
        testReportCommands = new LinkedList<Command>();  
  
        testCommand = new Command();  
        testCommand.setCommandName("getUserById");  
        testCommand.setMicroService("mw-organization");  
        testCommand.setResultPath("user");  
        testCommand.setResultClass(UserInfoDto.class);  
        testCommand.setKey("8b33b4f2042c4bbcb901e1e866d64b0b8");  
        testCommand.setUsePath(false);  
        testReportCommands.add(testCommand);  
    }  
    public LinkedList<Command> getCommandsForReport(String reportKey) {  
        return reportConfigMap.get(reportKey);  
    }  
}
```

Listing 20: Beispiel Report-Konfiguration

Mit der Methode `getCommandsForReport` kann die benötigte Liste von Befehlen aus der `HashMap` geladen werden. Im Beispiel in Listing 19 würde die Anwendung wie folgt ablaufen:

```

DataCollectionObject dco = new DataCollectionObject();
ReportingConfiguration config = new ReportingConfiguration();
config.prepareTestCommands();
dco.setCommands(config.getCommandsForReport("testReport"));

```

Listing 21: Anwendungsbeispiel Report- Konfiguration

Mit einem Befehl „testReport“ kann so die Liste mit den Befehlen für diesen Report geladen und in das DCO geschrieben werden.

7.1.7 Messaging

Das DCO wurde nun erstellt, und ist bereit, auf „die Reise“ zu gehen und Daten zu sammeln. In den Kapitel 6.2 und 6.6.1 wurde beschrieben, dass das DCO über eine Queue an die benötigten Microservices geschickt werden soll. Im bestehenden wird RabbitMQ bereits an diversen Stellen verwendet, weshalb teilweise auf die bestehende Implementierung zurückgegriffen werden kann.

Eine Message Queue dient dem Zweck, Nachrichten zu senden und zu verarbeiten. In vorliegenden Fall sollen mehrere Microservices in der Lage sein, Nachrichten zu empfangen. Im Konzept wurde beschrieben, dass die Nachrichten an einen Exchange gesendet werden, an den wiederum die Queue angeschlossen sind. Diese Zusammensetzung muss im Prototyp zuerst konfiguriert werden.

7.1.7.1 Messaging-Konfiguration

Eine funktionstüchtige Messaging-Umgebung benötigt diverse Komponenten um zu funktionieren:

- Nachrichten-Konverter um Objekte in ein handliches Format zu bringen.
- Exchange, um das Senden an mehrere Queues gleichzeitig zu ermöglichen
- Container als Gefäß für die Nachrichten
- Listener, um Nachrichten zu empfangen und verarbeiten
- Adapter, der sich aus dem Listener und dem Konverter zusammensetzt.

Mittels der „@Configuration“ Annotation kann die Klasse mit den benötigten Komponenten generisch zur Laufzeit erzeugt werden, ohne, dass diese explizit aufgerufen werden muss^[11]. Um die Komponenten automatisch zu generieren werden diese als „Bean“ annotiert^[12]. Die fertige Messaging-Konfiguration sieht wie folgt aus (Auszug):

```

@Configuration
public class ReportMessagingConfiguration {

    @Bean
    public Queue reportEngineQueue() {
        return new Queue(Constants.MESSAGE_QUEUE_REPORTING_ENGINE, true);
    }

    @Bean
    public FanoutExchange reportingExchange() {
        return new FanoutExchange("reportExchange");
    }
}

```

Listing 22: Beispiel Messaging-Konfiguration (Auszug)

Eine solche Konfiguration muss für jeden Microservice erstellt werden, der Nachrichten empfangen können muss. Ist eine Queue oder ein Exchange bereits vorhanden wird die existierende Komponente verwendet. Dies ist vor allem für den Exchange der Fall, da jeder Microservice seine Queue an den gleichen Exchange koppelt. Somit ist die Messaging-Infrastruktur grundsätzlich bereitgestellt.

7.1.7.2 Senden von Nachrichten

Um das DCO bestücken zu können muss dieses mittels einer Nachricht an den Exchange gesendet werden. Es wird also eine Klasse benötigt, die diese Aufgabe übernimmt.

Dieser MessageSender ist verantwortlich für die Erstellung und das Senden des DCO. Neben dem DCO wird ein sogenanntes RabbitTemplate benötigt. Dieses verschafft dem Sender den Zugang zu RabbitMQ, ohne, dass noch etwas implementiert werden muss. Über die Annotation „Autowired“ wird das Template automatisch durch Spring instanziiert und kann sofort verwendet werden.

Das DCO kann innerhalb des Senders erstellt werden. Wichtig ist dabei, dass die Klasse (das POJO) gesetzt wird, bevor das DCO zum ersten Mal in Queue geschickt wird. Das folgende Beispiel verdeutlicht das Prinzip:

```
@Component
public class ReportEngineMessageSender {

    //private static final Logger = LoggerFactory.getLogger(ReportEngineMessageSender.class);

    @Autowired
    private RabbitTemplate rabbitTemplate;

    public void sendMessage()
        throws IllegalAccessException, InvocationTargetException {

        TestReportResult testReportResult = new TestReportResult();
        DataCollectionObject dco = new DataCollectionObject();
        ReportingConfiguration config = new ReportingConfiguration();

        config.prepareTestCommands();

        dco.setCommands(config.getCommandsForReport("testReport"));
        dco.setResult(testReportResult);

        rabbitTemplate.convertAndSend("reportExchange", "", dco);
    }
}
```

Listing 23: Beispiel Report Nachrichten Sender

In diesem Beispiel soll ein Objekt des Typs „TestReportResult“ erstellt werden. Dazu muss ein solches Objekt instanziiert und im DCO gesetzt werden. Danach werden mit prepareTestCommand() die Befehle für diesen Report geladen. Und im DCO als verkettete Liste gesetzt. Danach kann das Objekt der Queue übergeben werden.

Zu beachten ist der Aufruf für das Senden, „convertAndSend“. Das DCO ist ein eigenes Objekt, das erstellt wurde. Um die Performance zu verbessern und die Komptabilität zu gewährleisten muss das DCO in ein Objekt des Typs „Message“ konvertiert werden. RabbitMQ nutzt dazu einen eigenen Converter. Dieser serialisiert das Objekt in einen JSON String und speichert die Information in einem neu angelegten Message Objekt.

Die De-Serialisierung der Nachricht überführt das Nachrichten-Objekt in ein Objekt vom Typ Object. Um zu gewährleisten, dass die Elemente der DCO jeweils den korrekten Typ aufweisen muss die Typendefinition mitgegeben werden. RabbitMQ verwendet Jackson als JSON Converter^[13]. Aufgrund der Konfiguration von Jackson in Mespas Web muss allerdings ein eigener Converter geschrieben werden, da die Standard-Variante – mitgeben des Klassentyps per Java-Annotation – nicht korrekt funktioniert. Der neue Converter muss zunächst mit dem RabbitTemplate vertraut gemacht werden.

RabbitMQ stellt mit „setMessageConverter“ eine entsprechende Methode bereit. Daher muss das RabbitTemplate Bean entsprechend angepasst werden:

```
@Bean
public RabbitTemplate mespasRabbitTemplate(ConnectionFactory connectionFactory)
{
    RabbitTemplate rabbitTemplate = new RabbitTemplate(connectionFactory);
    // set JSON message converter by default
    rabbitTemplate.setMessageConverter(jsonMessageConverter());
    return rabbitTemplate;
}
```

Listing 24: Festlegen des Converters im RabbitTemplate Bean

Die eigentliche Implementierung des Converters benötigt folgende Informationen:

- Instanz des JsonMessageConverters (wird von Spring bereitgestellt)
- ObjectMapper (wird von Jackson bereitgestellt)
- Objekt der Klasse SimpleModule (ebenfalls von Jackson)
- Spezifischer De- und Serializer für das DataCollectionObject.

Der De- und der Serializer werden dem SimpleModule hinzugefügt. Der ObjectMapper wird erzeugt und registriert das Modul und schliesslich wird der fertige ObjectMapper in der Instanz des Converters registriert. Da es sich bei diesem Converter um ein Bean handelt wird dieses von Spring bei der Verwendung automatisch instanziiert. Mit der Verwendung dieses Converters werden die Untertypen des DCO korrekt deserialisiert und können weiter verarbeitet werden. Dies ist notwendig um auf die Zwischenresultate zugreifen zu können. Mehr dazu folgt im nächsten Kapitel.

7.1.7.3 Nachrichtenverarbeitung

Dieser Teil ist das Herzstück der Lösung. Nachrichten müssen gelesen und verarbeitet werden. Das DCO ist nun fertig konfiguriert und wird, wie in Listing 22 beschrieben, an die Queue gesendet. Über die web-basierte Administration-Konsole von RabbitMQ kann man sehen, ob sich eine Nachricht in der Queue befindet:

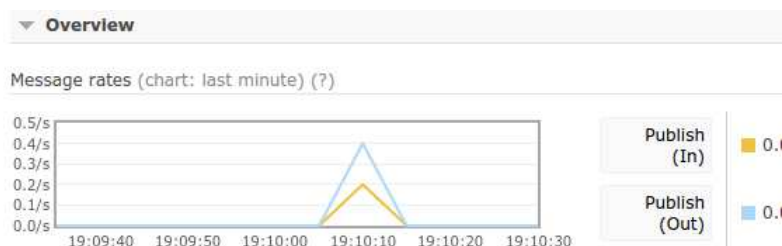


Abb. 45: Ausschnitt Admin Web-Konsole RabbitMQ

Um diese Nachrichten konsumieren zu können, benötigt jeder involvierte Microservice einen sogenannten MessageReceiver. Da alle dieser MessageReceiver teilweise über die gleiche Funktionalität verfügen wird eine abstrakte Klasse erstellt, die die gemeinsamen Methoden beinhaltet. Jeder MessageReceiver erweitert dann diese Klasse. Die Methoden, die für jeden MessageReceiver spezifisch implementiert werden sollen, werden als abstrakte Methoden in der Superklasse definiert.

```
protected abstract void doProcessMessage(T message) throws Exception;
```

Listing 25: Ausschnitt aus AbstractMessageReceiver

Listing 24 zeigt eine Methode, die zwingend von den erbbenden Klassen implementiert werden Es handelt sich dabei um die Methode „doProcessMessage“. Sie ist verantwortlich für das Verarbeiten von eingehenden Nachrichten. Ein Microservice, der auf Nachrichten hören soll erhält also eine eige-

ne Implementierung einer solchen Klasse. Das Prinzip ist dabei jeweils das gleiche, welches in diesem Kapitel erläutert wird.

7.1.7.3.1 Laden des Befehls

Zuerst wird der erste Befehl aus der Befehlsliste geladen. Je nach Konfiguration enthält dieser einen festen Schlüssel als Suchbegriff oder ein Zwischenresultat aus einem vorhergehenden Befehl. Beim ersten Befehl handelt es sich in der Regel um einen festen Schlüssel, sozusagen als Eingangspunkt. Der Befehl enthält einen Namen, mit dem ein Befehl dynamisch identifiziert und ausgeführt werden soll. Um dies zu bewerkstelligen, müssen drei Klassen implementiert werden:

- ReportCommandFactory: dient zur dynamischen Identifikation eines Befehls
- ReportCommand: Interface für die Implementierung eines auszuführenden Befehls
- Subklasse, die ReportCommand implementiert

Da das Interface ReportCommand für alle Microservices sichtbar sein soll, wird dieses im Projekt mw-common-domain erstellt, da auf die Pakete in diesem Projekt von allen anderen Projekten aus zugegriffen werden kann. Das Interface benötigt eine Methode, die überschrieben werden soll:

```
public interface ReportCommand {  
    /**  
     * Command executed by a ReportingMessageReceiver to retrieve data.  
     * @param argument Key or ReportResult object used to identify a record.  
     * @return Instance of the implementing class  
     */  
    public Object execute(Object... argument);  
}
```

Listing 26: ReportCommand Interface

Die Implementierung des Interfaces erfolgt dann im jeweiligen Projekt. Im Prototyp wird zum Beispiel ein Befehl zur Identifikation eines Benutzers benötigt. Daher wird dafür eine Klasse „GetUserCommand“ erstellt, welche das Interface ReportCommand erstellt. Der Befehl execute muss überschrieben und sinnvoll implementiert werden. Da GetUserCommand im Prototyp der erste Befehl ist, der verarbeitet wird, wird als Argument die UserId mitgegeben. So kann über die bereits vorhandene Schnittstelle ein BenutzerObjekt von der Datenbank geladen und zurückgegeben werden.

```
@Component  
public class GetUserCommand implements ReportCommand {  
  
    @Autowired  
    private UserRepository userRepo;  
  
    @Override  
    public Object execute(Object... argument) {  
        return UserInfoDtoFactory.valueOf(  
            userRepo.findById(argument[0].toString())  
        );  
    }  
}
```

Listing 27: GetUserCommand

Damit die Klasse identifiziert werden kann, ohne, dass der Microservice den genauen Typ weiss, wird die ReportCommandFactory benötigt. Diese hält jeweils eine Instanz der Befehlsklassen als Instanzvariablen und liefert diese anhand des Befehlsnamens (String im Objekt Command) zurück.

7.1.7.3.2 Verarbeitung des Befehls

Der erste Befehl ist nun geladen und kann entsprechend ausgeführt werden. Da als Identifikation ein Key verwendet wird, kann der Befehl direkt ausgeführt werden. Die Rückgabe hat den Typ Object, da der MessageReceiver nicht weiss, welcher Typ zurückgegeben wird.

Damit das DCO das Resultat erhält wird dieses mit der Klasse BeanUtils gesetzt. Mit der Methode „setProperty“ kann ein Wert als Property in einem Objekt gesetzt werden^[14]. Mit dieser Methode können Werte direkt wieder aus dem Bean ausgelesen werden, und zwar in einer Vordefinierten Struktur. Beispiel: Die Klasse „Benutzer“ verfügt über eine Instanzvariable namens „Student“, welches ein Objekt von Student ist. Student selber verfügt über Felder namens Vorname und Nachname. Mit setProperty können nun Werte direkt in diesem Bean gesetzt werden. Über die jeweiligen Instanzvariablen können so direkt ganze Objekte abgelegt werden. Gegeben sei beispielweise folgender Code-Ausschnitt:

```
public class Benutzer {
    private Student student;
}

public class Student {
    private String vorname;
    private String nachname;
}

public static void main() {
    Benutzer benutzer = new Benutzer();
    Student student = new Student();
    student.setNachName("Mustermann");
    student.setVorname("Max");
    BeanUtils.setProperty(benutzer, "student", student);
}
```

Listing 28: Beispiel BeanUtils

Der Aufruf von BeanUtils.setProperty speichert das Objekt als Property im Bean Benutzer. Das erste Argument ist dabei das Zielobjekt. Das zweite Argument ist der Schlüssel. Dieser muss mit dem Namen der Variabel in der Zielklasse übereinstimmen („student“). Das dritte Argument ist das zu speichernde Objekt. Nun kann man mit getProperty auf sämtliche Daten wieder zugreifen. Angenommen, man interessiert sich für den Nachnamen. Dann kann man mit folgendem Aufruf an diese Information kommen:

```
BeanUtils.getProperty(benutzer, "student.nachname");
```

Listing 29: Beispiel GetProperty

Der Punkt (.) stellt dabei eine Art Trenner dar. Mit jedem Punkt kann man in ein Objekt einsteigen. Sollte in der Klasse Student also ein weiteres Objekt, wie z.B. Adresse vorkommen, das Information die z.B. Strasse beinhaltet, kann man als Argument „student.adresse.strasse“ verwenden. So kann in der vorliegenden Lösung auf einfache Weise ein Zwischenresultat gesetzt werden. Wenn also ein Befehl ein entsprechendes Objekt zurückliefert, kann so das Zwischenresultat einfach gesetzt werden, das später mit einem String wieder identifiziert werden kann. Wird im Prototyp also der Befehl GetUserCommand ausgeführt, wird ein entsprechendes Objekt zurückgeliefert. Mit BeanUtils.setProperty werden die Werte dieses Objektes als Properties im DCO gespeichert.

Wir nun der nächste Befehl verarbeitet, der als Key eine Information aus dem Benutzerobjekt benötigt, kann über BeanUtils.getProperty diese Information direkt geholt werden. Damit der MessageReceiver nicht für jeden möglichen Datentypen eine Fallunterscheidung machen muss, wird die Infor-

mation über den Typen im Befehl konfiguriert. Über ein dynamisches Casting kann das Objekt dann in den korrekten Typen überführt werden:

```
command.getResultClass().cast(res)
```

Listing 30: Dynamisches Type-Casting

7.1.7.3.3 Weitere Schritte

Wurde ein Befehl erfolgreich verarbeitet wird dieser aus der Liste entfernt. Sind weitere Befehle in der Liste vorhanden, wird das DCO wieder an den Exchange gesendet, damit diese wie oben beschrieben abgearbeitet werden können. Liegen keine Befehle mehr vor, ist das Resultat komplett und kann an den Reporting-Microservice übergeben werden. Dazu ist für jeden Report als letzter Befehl ein Command mit dem Namen „finalize“ mit „mw-reporting-engine“ als relevanten Microservice zu konfigurieren. Der MessageReceiver für den Reporting-Microservice erkennt dann diesen Befehl und übergibt das fertige Resultat-Objekt der Reporting-Engine, die den Report dann erstellen soll.

7.1.8 Vorbereitung Report-Template für den Prototypen

Damit die Daten als BIRT Report ausgegeben werden können muss zunächst ein Report-Template erstellt werden. Mit dem BIRT Designer kann ein solches erstellt werden. Dieser ist als Open-Source Projekt verfügbar und ist basiert auf Eclipse. Zunächst wird ein neues Report-Template erstellt. Wie in Kapitel 2.2 beschrieben, benötigt das Template eine Datenquelle Da die Daten für den Report als POJO aufbereitet werden wird hier die POJO Date Source definiert.

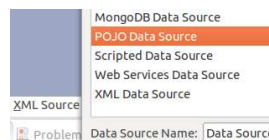


Abb. 46: Auswahl der Datenquelle in BIRT

Im Gegensatz zu einer JDBC Datenquelle verfügt diese Datenquelle über keine Treiber die benötigt werden. Jedoch können so die Datasets, in denen sich die Daten befinden, als POJO Dataset definiert werden. Dies ist wichtig für die Konfiguration des Reports. Als nächstes wird das Dataset erstellt. Das Dataset wird als POJO Dataset definiert. Zudem muss ein Kontext Key definiert werden. Dieser wird benötigt, damit die Applikation das Dataset von aussen identifizieren bzw. die Daten an das Dataset übergeben kann. Weiter muss die Klasse des POJO mit dem qualifizierten Namen angegeben werden. Damit werden beim Laden der Daten die benötigten Methoden mit dem Report bekannt gemacht. Da das BIRT Template keine direkte Verbindung mit dem Web Projekt hat wird es nun etwas umständlich, denn die Spalten für das Dataset müssen manuell in der XML Datei des Templates konfiguriert werden. Im Element „data-sets“ findet sich ein Subelement „oda-data-sets“. Dort müssen nun die Definitionen für die Spalten eingetragen werden.

```
<data-sets>
  <oda-data-set extensionID="org.eclipse.birt.data.oda.pojo.dataSet" name="userDataset" id="8">
    <property name="nullsOrdering">nulls lowest</property>
    <list-property name="columnHints">
      <structure>
        <property name="columnName">userId</property>
        <property name="analysis">dimension</property>
        <property name="onColumnLayout">false</property>
        <text-property name="heading">userId</text-property>
      </structure>
    </list-property>
  </oda-data-set>
</data-sets>
```

Listing 31: Definition einer Spalte in einem BIRT Dataset

Zusätzlichen müssen diese Spalten auch weiteren Elementen im XML definiert werden, um das Resultat abbilden zu können.

Wenn das XML konfiguriert ist, sind die Spalten korrekt im Dataset sichtbar:

Name	Type	Ali
userId	String	
firstName	String	
lastName	String	
street	String	
zipCode	String	
city	String	

Abb. 47: Spaltendefinition in einem BIRT Dataset

Nach der Spaltendefinition müssen auch die Methodennamen der Klasse in der XML Datei definiert werden. Diese Definition wird im Bereich „queryText“ zu finden. Die Methodennamen sind als eine Art Query zu definieren. Beispiel:

```
<xml-property name="queryText">
  <![CDATA[<?xml version="1.0" encoding="UTF-8" standalone="no"?>
    <PojoQuery appContextKey="USER_REPORT" dataSetClass="TestReportResult" versi-
    on="1.0">
      <ColumnMapping index="1" name="userId" odaDataType="String">
        <Method name="getUserId" />
      </ColumnMapping>
    </PojoQuery>
  ]]>
</xml-property>
```

Listing 32: Beispiel Methoden-Definition in einem BIRT Dataset

Sind die Methoden korrekt definiert worden, sind diese entsprechend im Dataset zu sehen:

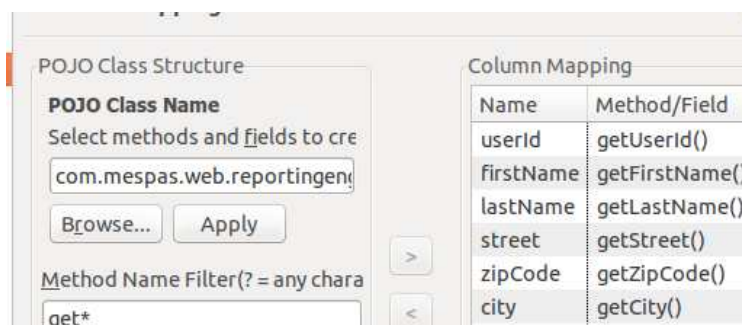


Abb. 48: Methodennamen in einem BIRT Dataset

Nun ist das Dataset fertiggestellt und kann verwendet werden. Nun muss das eigentliche Design erstellt werden. Dies erfolgt regulär mit dem BIRT Report Designer. Mit Drag and Drop können die benötigten Elemente platziert und formatiert werden. Für den Prototypen soll das eine Visitenkarte sein. Wie in Abb. 48 zu sehen ist, sind alle benötigten Methoden und somit auch die benötigten Datenfelder vorhanden.



Abb. 49: Report-Design

Zu beachten ist dabei folgendes: um Vor- und Nachname sowie PLZ und der Ort in jeweils einer Spalte dargestellt werden können, werden die Spalten „firstName“ und „zipCode“ jeweils mittels einem Script komplettiert:

```
1 dataSetRow["firstName"] + " " + dataSetRow["lastName"]
```

Abb. 50: Bearbeiten eines Datenfeldes mittels Script

Somit ist das Report-Template fertig und kann von der Engine verarbeitet werden. Die notwendigen Schritte dafür werden im nächsten Kapitel erläutert.

7.1.9 Ausführen eines Reports

Da die Daten und das Template nun bereit sind, geht es darum, den Report auszuführen. Da auch für die Ausführung BIRT verwendet wird ist der Implementierungsaufwand nicht so gross. Es werden jedoch einige Komponenten benötigt.

7.1.9.1 Laden und vorbereiten des Templates

Zunächst wird das Template geladen. Im Prototyp gibt es nur eines, daher kann der Dateiname statisch festgelegt werden. Im Kapitel 8 (Ausblick) wird eine Möglichkeit aufgezeigt, wie dies dynamisch erfolgen kann. Das Template wird als InputStream direkt vom Dateisystem geladen. Mit diesem Stream kann von der Engine ein IReportRunnable Objekt (sogenanntes Design Handle) geladen werden. Die ReportEngine liefert mit dem Stream als Argument ein solches Objekt zurück. Mit diesem Objekt wird ein neuer Task erstellt, der am Ende dann ausgeführt wird.

```
InputStream reportTemplate = new FileInputStream(  
    new File("reports/UserReport.rptdesign")  
);  
IReportRunnable design = reportEngine.openReportDesign(reportTemplate);  
final IRunAndRenderTask task = reportEngine.createRunAndRenderTask(design);
```

Listing 33: Vorbereiten des Report Templates im Code

Wenn ein Report Parameter benötigt, müssen diese ebenfalls aus dem Design Handle geladen werden.

```
IGetParameterDefinitionTask parameterDefinitionTask = de-  
sign.getReportEngine().createGetParameterDefinitionTask(design);  
parameterDefinitionTask.evaluateDefaults();  
HashMap<String, String> reportParams = parameterDefinition-  
Task.getDefaultValues();  
parameterDefinitionTask.setParameterValues(reportParams);
```

Listing 34: Vorbereiten der Report-Parameter

7.1.9.2 Übergabe der Daten an den Report

Der Report ist nun vollständig geladen. Nun müssen die Daten, die mit diesem Konzept aufbereitet wurden an den Report übergeben werden. In Kapitel 7.1.7 wurde beschrieben, dass ein Kontext Key benötigt wird, um das Dataset identifizieren zu können. Hier kommt dieser Kontext wieder zum Zug. Benötigt wird eine HashMap in der der Kontext Key als Key und das ResultObject als Wert abgelegt wird damit das Dataset im Report entsprechend mit Daten bestückt werden kann. Im Fall des Prototyps kann dies relativ statisch erfolgen. Im allgemeinen Fall wird eine Liste von Objekten für jedes Dataset erwartet wobei jeder Listeneintrag eine Zeile im Dataset repräsentiert.

```
Map<String, Object> contextMap = new HashMap<>();
List<ReportResult> reportResult = new ArrayList<ReportResult>();
reportResult.add(result);
contextMap.put("USER_REPORT", reportResult);
task.setAppContext(contextMap);
```

Listing 35: Übergabe der Daten an ein Dataset

7.1.9.3 Rendering

Als letzter Schritt muss die Reporting Engine Den Report generieren und als PDF Datei speichern (Prototyp) oder zurücksenden (In der fertigen Lösung über eine REST-Schnittstelle). Zunächst muss eine Variable vom Typ `ByteArrayOutputStream` instanziiert werden. Diese wird an die Klasse `IRenderOption` übergeben um die Details der Generierung festzulegen, wie z.B. den Datentyp (im Falle des Prototyps ist das PDF). Das zurückgelieferte `IRenderOption` Objekt wird im Task gesetzt, und dieser kann nun mit der Methode `run()` ausgeführt werden. Nach der Ausführung wird der Inhalt des `ByteArrayOutputStream` in ein `ByteArray` geschrieben mit welchen dann die PDF Datei erstellt wird. Treten bei der Ausführung Fehler auf, wird eine Exception geworfen und die Engine schreibt Fehler in eine Liste von Objekten. Das Nachfolgende Listing zeigt das Rendering etwas deutlicher:

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
task.setRenderOption(getOptions(baos));

task.run();
List<?> errors = task.getErrors();
task.close();
if ((errors != null) && !errors.isEmpty()) {
    throw new MespasRuntimeException("Found errors generating the report: " +
    errors);
}
// copy
byte[] content = baos.toByteArray();

OutputStream o = new FileOutputStream("report.pdf");
o.write(content);
o.close();
```

Listing 36: Erstellen der PDF Datei

Ein Blick in das Projektverzeichnis bestätigt, dass die PDF Datei nun vorhanden ist:

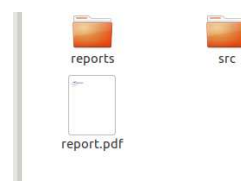


Abb. 51: Fertiger Beispiel-Report im Projektordner

Wird die Datei geöffnet, sieht man, dass der Report ordnungsgemäss erstellt wurde.



Abb. 52: Fertiger Beispiel-Report

7.2 Zusammenfassung Proof of Concept

Anhand des Prototyps konnte aufgezeigt werden, dass das Konzept grundsätzlich funktioniert. Auch für komplexere Datengebilde kann es verwendet werden, da im Resultate Objekt polymorphe Objekte unterstützt werden. Dies wird auch nötig sein, sobald ganze Listen von Daten verarbeitet werden müssen. Dazu muss in den jeweiligen MessageReceivern eine entsprechende Schlaufe eingebaut werden, die mittels BeanUtils.setProperty die Werte setzen kann.

7.2.1 Anforderungen

Im Kapitel 7.1.2 wurden folgende Anforderungen, die abgedeckt oder beschrieben werden sollen, festgelegt:

- R-001: Report ausführen
- R-004: Report Formate
- R-013: Report Parameter – allgemein
- R-014: Report Parameter – optionale Parameter
- R-020: Daten - Datenbank
- R-021: Daten - Datenobjekte

Es soll nun kurz untersucht werden, ob diese Anforderungen abgedeckt oder beschrieben werden konnten.

7.2.1.1 Report ausführen

Der Report konnte korrekt ausgeführt werden. Das korrekte PDF Dokument wurde erzeugt und kann korrekt geöffnet werden.

7.2.1.2 Report Formate

In Kapitel 7.1.9.3 wurde beschrieben, dass mit IRenderOption für den Prototypen festgelegt wurde, dass das zu erzeugende Format PDF sein soll. In dieser Option kann auch festgelegt werden, dass das Format Excel sein soll. Es wird von BIRT in beiden Formaten (xls und.xlsx) unterstützt. Diese Anforderungen kann somit erfüllt werden.

7.2.1.3 Parameter

Der Beispiel-Report im Prototyp verwendet keine Parameter. In Listing 33 ist jedoch zu sehen, dass eine Liste von Parametern verarbeitet werden kann. Somit müssen die Parameter nur noch im Request mitgeliefert werden, damit diese verarbeitet werden können. Somit ist auch diese Anforderung für sowohl optionale als auch obligatorische Parameter erfüllt.

7.2.1.4 Daten

In Kapitel 7.1.9.2 konnte dargelegt werden, wie die fertigen Daten an ein bestimmtes Dataset im Report übergeben werden können. Somit ist auch diese Anforderung erfüllt.

7.2.2 Weiteres

Der Prototyp wurde erfolgreich erstellt und die definierten Anforderungen für diesen konnten erfolgreich erfüllt werden. Im nächsten Kapitel soll beschrieben werden, wie sich das System für komplexe Daten und als System an sich weiter ausgebaut werden kann.

8. Ausblick

Mit der Erstellung des Prototyps konnte aufgezeigt werden, dass das Konzept grundsätzlich funktioniert. Da das Beispiel im Prototyp relativ simpel aufgebaut ist, wird sich die Frage aufdrängen, wie sich die Lösung mit komplexeren Daten verhält.

8.1 Komplexere Daten

Natürlich sollen als zwischen Resultate auch Listen möglich sein. Zum Beispiel kann es sein, dass man alle Schiffe für eine Reederei laden, und diesen weitere Informationen laden. Dazu muss überprüft werden, ob es sich beim Objekt um eine Liste haben. Das nachfolgende Listing zeigt eine mögliche Lösung, wobei „argument“ ein Element aus einem String Array ist, das aus dem Property Key von BeanUtils erstellt wurde und „path“ das Array selbst ist.

```
if (argument.getClass().isAssignableFrom(List.class)) {  
    // check if we have more properties in // the path  
    if (i < path.length - 1) {  
        List args = (List) argument;  
        List<Object> result = new LinkedList<Object>();  
        String propertyName = pathElement + 1;  
        for (Object argObject : args) {  
            Object listEntry = BeanU-  
tils.getProperty(argObject, propertyName);  
            result.add(listEntry);  
        }  
        argument = result;  
        break;  
    }  
}
```

Listing 37: Mögliche Lösung für das Handling von Listen

Auch Listen können demnach korrekt verarbeitet werden. Die Implementierung von komplexeren Daten ist soweit möglich, da für jeden Report ein Resultat und somit eine implementierte Klasse benötigt wird. Der Aufwand für die Implementierung kann zwar je nach Komplexität sehr aufwändig werden, jedoch gibt es grundsätzlich keine Einschränkungen. Werden verschachtelte Daten benötigt können auch entsprechend mehrere Dataset angelegt werden. Das Zusammenspiel zwischen dem Report und der Applikation funktioniert gut, da mittels dem ApplicationContext Informationen direkt über die bereitgestellten Schnittstellen ausgetauscht werden können.

8.2 Weiterentwicklung System

Der Prototyp soll nun schrittweise weiter entwickelt werden, sodass am Ende ein komplettes Reporting-System zur Verfügung steht. Im Kapitel 3 sind die Anforderungen umfassend beschrieben und im Kapitel 5 wurde das Design für das System erstellt. Um das System als Ganzes bauen zu können sind einige Komponenten erforderlich. In diesem Kapitel soll einführend erläutert werden, was grundsätzlich für den Bau dieses Systems noch benötigt wird.

8.2.1 Umgebung

Der Microservice wurde bereits im Rahmen des Prototyps erstellt. Zusätzlich muss eine grafische Oberfläche bereitgestellt werden. Dazu wird ein Klasse für die Konfiguration benötigt. Diese wird mit @Configuration annotiert, damit Spring diese Klasse automatisch lädt. Sie enthält Informationen die über eine Web-Applikation wichtig sind, wie zum Beispiel Port, Name des Service, etc. Das folgende Listing zeigt ein Beispiel, wie eine solche Konfiguration aussehen könnte:

```
ServiceConfiguration config = ServiceConfiguration.createNew()  
    .withBaseUrl(serverAddress + ":" + port)  
    .withShortName(Constants.REPORTING_ENGINE)  
    .withName("MESPAS Reporting Engine")  
    .withType(ServiceType.RELATIONAL)  
    .withMenuItems(getMenuItems())  
    .withJavaScriptResources(javaScriptResources())  
    .withCssResources(cssResources());  
configPublisher.publish(config)
```

Listing 38: Beispiel UI Konfiguration

Die Methoden, die aufgerufen werden, laden weitere Konfigurationselemente wie das Menu, Stylesheets, etc.

In diesen Bereich fällt auch die Rolle. Da nicht jeder Benutzer einen Zugang zur neuen Reporting-Lösung haben soll, ist dies über Rollen zu steuern. In MESPAS geschieht diese Rollensteuerung über die Menu-Elemente. Dort kann eine entsprechende Option hinzugefügt werden, die die berechtigte Benutzergruppe festlegt. In Listing 38 wird ein Aufruf auf die Methode „getMenuItems“ gezeigt. In dieser Methode sollen Menu-Elemente hinzugefügt werden und können dabei eine Rolle als Option aufnehmen:

```
LinkMenuItem reportingMenu = new LinkMenuItem("Reporting",  
    "menu.reporting.admin",  
    UserRoles.REPORTING_ADMIN,  
    "/reporting/admin",  
    "/app/components/reporting/reporting.html",  
    "ReportingController",  
    ICON_REPORTING,  
    1);
```

Listing 39: Hinzufügen eines Menu-Elementes mit entsprechender Rolle

Im obigen Beispiel handelt es sich um die Standard-Implementierung für das Hinzufügen eines Menu-Elementes in MESPAS Web. Es ist zu sehen, wie über die Konstante `UserRoles.REPORTING_ADMIN` eine entsprechende Rollen-Id festgelegt wird.

8.2.2 UI

Über die Konfiguration des Microservice können auch optische Aspekte eingestellt werden. Der Aufruf der Methode „cssResources“ in Listing 38 zeigt das Prinzip. Sie liefert eine Liste von Pfaden zu entsprechenden Style-Dateien (Cascading Stylesheet, CSS) zurück. Die Implementierung des UI selbst erfolgt jeweils nach demselben Schema wie die bereits entwickelten Applikationsteile. Daher wird an dieser Stelle nicht näher darauf eingegangen.

8.2.3 Datenbank

Die Reports sollen bekanntlich auch terminiert werden können. Zu diesem Zweck ist eine Datenbank zu erstellen, die es erlaubt, die Schedules entsprechend zu speichern. Die benötigten Tabellen wurden bereits in Kapitel 5.3 beschrieben.

9 Testing

9.1 Testkonzept

Wie jedes Software-Projekt muss auch der Prototyp der Reporting-Lösung getestet werden. Dazu soll ein entsprechendes Testkonzept erarbeitet werden welches die wichtigsten Testarten behandelt.

Folgende Tests sollen konkret abgedeckt werden:

- Unit Tests für die Logik
- Integrationstests um die Funktionalität und das Zusammenspiel des DCO und den Queues zu testen
- Regression Tests für Änderungen
- Acceptance Tests um die Anforderungen zu testen.

In den folgenden Kapiteln sollen die wichtigsten Tests beschrieben werden. Für die Acceptance Test wird zusätzlich ein Testprotokoll für die Anforderungen, die in der Kernlösung abgedeckt werden konnten, erstellt.

Um die Tests einfach zu integrieren, werden Test-Reports definiert, anhand derer die Lösung getestet werden soll.

9.2 Unit Tests

Mit den Unit Tests soll die Logik der einzelnen Komponenten getestet werden. In der testgetriebenen Entwicklung (Test-Driven Development, TDD) geschieht dies idealerweise vor der Implementierung. Da bereits vorher bekannt ist, wie sich die Logik zu verhalten hat kann die Implementierung aufgrund der Testergebnisse erfolgen.

In diesem Projekt werden die Unit Tests mit JUnit und Mockito geschrieben. Mockito ist ein Framework zum Erstellen von Mock-Objekten. Dies ist nützlich, da so nicht jedes Objekt mühsam instanziiert werden muss, sondern direkt verwendet werden kann. Über die Annotation „@Mock“ kann eine Klasse, die im Test benötigt wird, als Mock definiert werden. Dies ist vor allem für Container-Klassen nützlich. Daher weisen alle Unit Tests, die für dieses Projekt geschrieben werden eine ähnliche Struktur auf. Diese Struktur soll an einem Beispiel erläutert werden.

9.2.1 GivenWhenThen Prinzip

Die Unit Tests sollen nach dem „GivenWhenThen“ aufgebaut werden. Dieses Prinzip wurde von Dan North, dem Begründer des „Behaviour-Driven Development“ (BDD), beschrieben^[15]. Es handelt sich dabei um ein strukturiertes Format um mit Beispieldaten Szenarien auszudrücken die Vor- und Nachbedingungen beinhalten^[16]. Dabei weisen die Unit Tests immer folgende Struktur auf:

- Given: welche Bedingungen müssen erfüllt sein, damit der Test durchgeführt werden kann?
- When: was wird getestet? Welche Eingabe sollen getestet werden, when die Aktion auftritt? (when)
- Then: Was wird als Ausgang des Tests erwartet? Was sind die Nachbedingungen dieses Tests?

9.2.2 Vorbereitung des Tests

Mockito bietet Methoden an, dieses Prinzip anwenden zu können. Als Beispiel sei die Klasse „GetCityCommand“ zu testen. Diese Klasse wird vom MessageReceiver aufgerufen um anhand eines Benutzerobjektes dessen Stadt zurückzuliefern. Somit ist auch bereits bekannt, was als Resultat erwartet wird. Um an diese Information zu kommen, werden Instanzen einiger Klasse benötigt. Wie bereits erwähnt können diese mit der Annotation „@Mock“ gemockt werden:

```
@Mock
private UserRepository userRepo;

@Mock
private PostalAddressRepository addressRepo;
```

Listing 40: Mock-Up von Klassen

Damit diese auch verwendet werden können müssen die Mock initialisiert und mit der zu testenden Klasse bekannt gemacht werden:

```
@InjectMocks
private GetCityCommand command = new GetCityCommand();

@Before
public void setUp() {
    MockitoAnnotations.initMocks(this);
}
```

Listing 41: Initialisierung des Test-Objekts und der Mocks

Nun können die Mocks verwendet werden. Jedoch werden auch Daten benötigt, damit die zu testende Klasse auch sinnvolle Resultate zurückliefern kann.

9.2.3 Simulieren von benötigten Daten

Damit mit der Methode „execute“ in der Klasse „GetCityCommand“ ein Resultat zurückgeliefert werden kann müssen Daten vorliegen. Die Stadt eines Benutzers kann über die Klasse „PostalAddress“ zurückgegeben werden. Um von einem User-Objekt an den korrekten PostalAddress Eintrag zu kommen muss zunächst ein Objekt der Klasse „Individual“ gemockt werden, da die PostalAddress dort zu finden ist. Mit den Methoden „mock“ und „when“ von Mockito können solche benötigten Objekte mit den Daten angelegt werden:

```
Individual individual = mock(Individual.class);
when(individual.getEmailAddresses()).thenReturn(Arrays.asList(new
EmailAddress(true, "email@email.com") ));
when(individual.getFirstName()).thenReturn("firstName");
when(individual.getLastName()).thenReturn("lastName");
when(individual.getPostalAddresses()).thenReturn(Arrays.asList(new Posta-
lAddress(true, "street", "2", "city", "state", Country.AL, "pobox", "34567")));
when(individual.getTitle()).thenReturn>Title.MISTER);
```

Listing 42: Anlegen benötigter Objekte mit Daten

Zunächst wird mit „mock“ das Objekt instanziiert. Mit „when“ können nun Daten simuliert werden. Es wird der Wert festgelegt, der zurückgegeben werden soll, wenn die entsprechende Methode aufgerufen wird. Für das benötigte PostalAddress Objekt wird also eine neue ArrayList erzeugt, welche beim Aufruf von „getPostalAddresses“ zurückgegeben wird. Somit existiert nun ein Objekt der Klasse *Individual*. Dieses muss nun in ein Objekt der Klasse *User* eingefügt werden, denn über dieses Objekt soll letztendlich die Stadt gefunden werden. MESPAS Web stellt eine Testklasse zur Verfügung, mit der solche Objekte erzeugt werden können:


```
User user = anUser()  
    .withId("1")  
    .withLogin("first user")  
    .withIndividual(individual)  
    .build();
```

Listing 43: Erzeugen eines Testobjekts der Klasse User

Hier kann auch das benötigte Individual-Objekt gesetzt werden.

9.2.4 Erstellen des Unit Tests

Da die simulierten Daten nun verfügbar, kann der Test erstellt werden. Da die Klasse „UserRepository“ in der zu testenden Klasse verwendet wird, muss nun festgelegt, welches Objekt diese zurückgibt, da im Unit Test keine Datenbankverbindung erstellt werden soll. Mit der Methode „given“ von Mockito kann das geregelt werden:

```
given(userRepo.findById(user.getId())).willReturn(user);  
PostalAddress postalAddress = user.getIndividual().getPostalCodes().get(0);  
given(addressRepo.findOne("1")).willReturn(postalAddress);
```

Listing 44: Simulieren von Objekten die von der Datenbank geladen werden

Wenn also eine Methode in einem Objekt, das in der zu testenden Klasse verwendet wird, aufgerufen wird, kann so die Rückgabe bestimmt werden. Nun sind alle Daten vorhanden und das Verhalten der zu testenden Klasse kann eruiert werden. Im Beispiel soll bekanntlich die Klasse „GetCityCommand“ getestet werden. Die zu testende Methode ist hier „execute“. Mit der JUnit Methode „assertTrue“ kann nun überprüft werden, ob der erwartete Wert mit dem tatsächlichen Wert übereinstimmt:

```
Object testobj = command.execute("1");  
String city = testobj.toString();  
assertTrue("city".equals(city));
```

Listing 45: Überprüfen der Test-Resultate

Wenn der Unit Test nun erfolgreich durchläuft ist die getestete Logik korrekt implementiert, ansonsten müssen entsprechende Anpassungen vorgenommen werden. Diese Art von Test wird idealerweise immer dort eingesetzt, wo Logik implementiert wird. Zusätzlich kann auch die Testabdeckung mit entsprechenden Plug-Ins in Eclipse überprüft werden.

9.3 Integrationstest

Ein Integrationstest dient dazu, mehrere Komponenten, die in einem Gesamtkontext zusammengefügt werden, im gemeinsamen Zusammenspiel zu testen. Bei der Verwendung von Spring hat man den Vorteil, dass mit einem Integrationstest eine komplette Umgebung hochgefahren werden kann.

9.4 Regression Test

Jegliche Art von Software unterliegt Änderungen. Daher muss jedes System bei Änderungen einem Änderungstest, einem sogenannten Regressionstest unterzogen werden. Der Regressionstest ist ein erneuter Test eines bereits getesteten Programms nach dessen Modifikation mit dem Ziel, nachzuweisen, dass durch die Vorgenommenen Änderungen keine neuen Defekte eingebaut oder bisher maskierte Fehlerzustände freigelegt wurden^[17].

Das zu entwickelnde System muss daher über einen Regressionstest verfügen, damit Änderungen an der Applikation getestet und allfällige Fehler rasch gefunden werden.

Die automatische Ausführung von Unit und Integrationstests ist im Projekt bereits gewährleistet da sämtliche Tests beim Erstellen der Artefakte mittels Maven getestet werden. Daher müssen lediglich die Acceptance Tests noch manuell durchgeführt werden.

9.5 Acceptance Test

Zu guter Letzt muss ein Abnahmetest, ein sogenannter Acceptance Test durchgeführt werden. Dabei geht es darum, die Anforderungen in punkto Erfüllung zu testen. Zu beachten ist dabei, dass der nicht alle Anforderungen im Prototyp vorhanden sind. Daher muss unbedingt dafür gesorgt werden, dass bei der Implementierung des Gesamtprojektes die Anforderungen, die noch nicht getestet werden konnten, noch abgenommen werden.

Die Testkriterien sind bereits in den Anforderungen als Akzeptanzkriterien beschrieben. Um die Acceptance Tests verwalten zu können wird eine Software benötigt, in der die Tests erstellt, ausgeführt und dokumentiert werden können. Hier bietet sich die Lösung von Testlog an. Testlog ermöglicht die Erstellung und Verwaltung von Testdatenbanken. Nach Installation der Applikation muss dafür zunächst eine neue Datenbank erstellt werden:

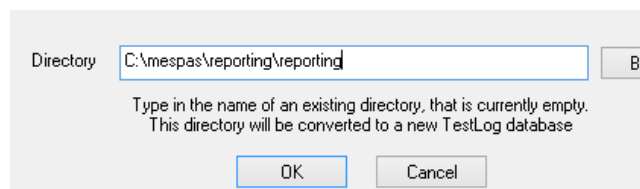


Abb. 53: Erstellen einer Test-Datenbank in Testlog

Dann müssen Test-Benutzer erstellt werden. Über den Menüpunkt „Create“ kann dazu „Tester“ ausgewählt werden. Es erscheint der folgende Dialog, der ausgefüllt werden kann:

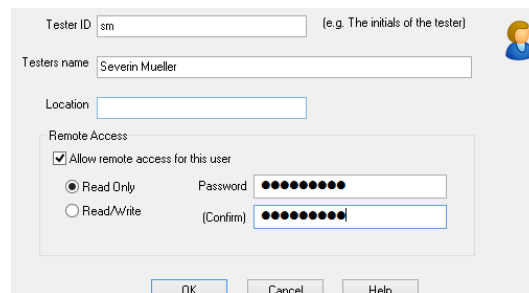


Abb. 54: Erstellen eines Benutzers in Testlog

Für diese Datenbank muss nun ein Test-Projekt angelegt werden. Testprojekte bilden jeweils eine eigene Einheit, die entsprechende Testfälle beinhalten. Innerhalb des Projekts wird dann eine sogenannte Test Suite angelegt. Diese kann für die Versionierung nützlich sein, wenn zum Beispiel verschiedene Branches getestet werden sollen.

Für die erstellte Suite können nun Testfälle erstellt werden. Dabei ist es für dieses Projekt sicherlich sinnvoll, wenn für jede Anforderung ein entsprechender Test erstellt wird. Dabei können die Angaben aus den Anforderungen direkt übernommen werden, wobei noch eine Beschreibung der Testschritte eingetragen werden muss. Dazu kommen weitere Informationen wie erwartete und tatsächliche Testdauer, empfohlene Konfiguration (welche separat eingestellt werden können), Test-Benutzer, Ein Beispiel Testfall könnte daher so aussehen:

The screenshot displays the 'Testlog' application interface, specifically the 'Test Description' tab. The interface is divided into two main sections: 'Test Case Summary' and 'Execution result'.

Test Case Summary:

- Test-ID:** Reporting-001
- Expected Duration:** 0 Hrs 0 Mins
- Title:** Report ausführen
- Test Type:** Reliability (dropdown menu)
- Test phase:** Acceptance test (dropdown menu)
- Recommended Config. IDs:** (input field) << Add
- Requirements:** (input field)
- Resource IDs Required:** (input field) << Add
- Buttons:** Browse, Launch

Execution result:

- Status:** Not yet attempted (dropdown menu)
- Test Attempts:** 0 (spinner)
- Last Attempt:** 23:08:28, 18-Okt-2015 (calendar icon)
- Actual Duration:** 0 Hrs 0 Mins
- Version Tested:** (dropdown menu)
- Build Tested:** (dropdown menu)
- Testers:** REEFMASTER (input field) << Add
- Tester to Add:** sm (dropdown menu)
- Fault Report ID:** (input field) View Add Del
- Change Request ID:** (input field)
- Results Obtained:** (large text area)
- External Result Link:** (input field) Browse Launch

Abb. 55: Testfall in Testlog

In der Registerkarte „Test Description“ sind die Informationen aus der Anforderung, sowie eine Testanleitung zu finden und in der Registerkarte „Test History“ sind die bisherigen Aufzeichnungen in dieser Test Suite zu finden. Somit eignet sich diese Applikation sehr gut für die Durchführung von Acceptance Tests.

10 Fazit und Erkenntnisse

Dieses Projekt war vom Schwierigkeitsgrad her eher komplex. Es wurden fundierte Java und BIRT Kenntnisse benötigt, um dieses Problem zu lösen. Durch das Studium an der ZHAW mit den Schwerpunkten Software Engineering, Java und Informationssysteme, sowie die berufliche Tätigkeit des Autors waren das erforderliche Know-how jedoch vorhanden. Schwierigkeiten entstanden hauptsächlich bei der Serialisierung der Datenobjekte, da die verwendete Jackson Version eine etwas spezielle Konfiguration aufweist. Mit einer eigenen Lösung für die Serialisierung konnte dieses Problem jedoch behoben werden.

Die vorliegende Lösung erlaubt es, die Daten in der gewünschten Form aufzubereiten und dabei die Typensicherheit zu bewahren. Auch komplexere Datenobjekte können mit dieser Lösung aufbereitet werden, da ganze Objekte mittels BeanUtils als Property im DataCollectionObject gespeichert werden können.

Im Grossen und Ganzen konnte eine verträgliche Lösung entwickelt werden, die es erlaubt eine Plattform innert nützlicher Frist zu bauen und zu betreiben.

Abschliessend bleibt zu sagen, dass im Unternehmen des Autors durchaus gewünscht wird, die Umsetzung dieses Systems voranzutreiben, denn der Bedarf an handlichen, ansprechend aufbereiteten Daten, wird in den nächsten Jahren sicherlich noch weiter zunehmen.

11 Anhang

11.1 Anhang A: Bilderverzeichnis

Abb. 1: Bestehende Lösung – Übersicht.....	9
Abb. 2: Reports in MESPAS Web (vereinfacht)	9
Abb. 3: Generierung von Reports.....	10
Abb. 4: Task Liste in r5.....	10
Abb. 5: Beispiel Report Task Liste	10
Abb. 6: Report-Button	11
Abb. 7: Ausgabeformat.....	11
Abb. 8: Dokumentenstruktur in der MRE.....	11
Abb. 9: Auswahl Beispiel-Report: Übersicht über die Wartungen der Motoren eines Schiffes	11
Abb. 10: Auswahl eines erforderlichen Parameters für einen MRE Report.....	12
Abb. 11: Resultat Beispiel-Report: Übersicht über die Wartungen der Motoren eines Schiffes.....	12
Abb. 12: Symbol zur Terminierung von MRE Reports.....	12
Abb. 13: Terminierung von Reports (Einstellungen)	12
Abb. 14: Terminierung von Reports - Optionen für den Output.....	13
Abb. 15: Übersicht der Offerten.....	13
Abb. 16: Beispiel-Report Web-Plattform: Übersicht Offerten	14
Abb. 17: Übersicht User Stories	15
Abb. 18: Import der BIRT Packages	30
Abb. 19: Aufbau Hauptseiten	33
Abb. 20: Mock-Up neue Navigationsleiste	33
Abb. 21: Mock-Up neuer Header und Suche.....	34
Abb. 22: Mock-Up Anzeige „My Reports“	35
Abb. 23: Mock-Up Anzeige "Standard Reports"	35
Abb. 24: Mock-Up Jobs.....	35
Abb. 25: Eingabe Report Parameter.....	35
Abb. 26: Mock-Up Job Details	36
Abb. 27: Mock-Up Jobs für Administratoren.....	37
Abb. 28: Übersicht Variante A	38
Abb. 29: SQL Datenquelle im BIRT Designer	39
Abb. 30: SQL Dataset.....	39
Abb. 31: Ausschnitt aus einem Scripted Dataset	40
Abb. 32: Import von Java Packages in BIRT.....	40
Abb. 33: Phasen eines BIRT Scripts	40
Abb. 34: Initialisieren eines Zählers	40
Abb. 35: Report-Parameter erstellen	41
Abb. 36: Übersicht Variante B	42
Abb. 37: DataCollectionObject	43
Abb. 38: Übersicht Variante C	45
Abb. 39: Ablauf Datenaufbereitung	51
Abb. 40: Sequenzdiagramm Variante B.....	52
Abb. 41: Klassendiagramm Variante B	53
Abb. 42: Neuer Eintrag in Gitolite-Config.....	54
Abb. 43: Projekt in Maven importieren.....	56
Abb. 44: Gestarteter Reporting-Microservice.....	57
Abb. 45: Ausschnitt Admin Web-Konsole RabbitMQ.....	61

Abb. 46: Auswahl der Datenquelle in BIRT.....	64
Abb. 47: Spaltendefinition in einem BIRT Dataset.....	65
Abb. 48: Methodennamen in einem BIRT Dataset.....	65
Abb. 49: Report-Design	65
Abb. 50: Bearbeiten eines Datenfeldes mittels Script	66
Abb. 51: Fertiger Beispiel-Report im Projektordner	67
Abb. 52: Fertiger Beispiel-Report	67
Abb. 53: Erstellen einer Test-Datenbank in Testlog.....	74
Abb. 54: Erstellen eines Benutzers in Testlog	74
Abb. 55: Testfall in Testlog	75

11.2 Anhang B: Listings

Listing 1: Instanzieren eines IReportRunnable Objektes.....	30
Listing 2: Erstellen eines BIRT Tasks	30
Listing 3: Übergabe von Parametern an einen Report.....	30
Listing 4: Output-Format eines Reports festlegen	31
Listing 5: Implementierung einer Mail-Klasse	31
Listing 6: Erstellen einer Session für den Mail-Versand	32
Listing 7: Erstellen eines Mail-Objektes	32
Listing 8: Erstellen eines Mail-Objektes mit Attachment.....	32
Listing 9: Daten-Aufbereitung mittels Script in BIRT	41
Listing 10: Beispiel-Befehl für das Laden eines Benutzernamen.....	43
Listing 11: Überprüfung eines Report-Parameters	46
Listing 12: Dynamische Anpassung einer Query mittels Script	46
Listing 13: Auschecken von gitolite	54
Listing 14: Änderungen gitolite-Config pushen	55
Listing 15: XML Struktur für Maven-Projekt.....	55
Listing 16: XML Struktur für Maven-Projekt.....	55
Listing 17: SQL-Query zur Erstellung der Reporting-Datenbank	56
Listing 18: Starten des Reporting-Microservice	57
Listing 19: Erster Entwurf DCO (Ausschnitt).....	58
Listing 20: Beispiel Report-Kongfiguration.....	58
Listing 21: Anwendungsbeispiel Report- Konfiguration	59
Listing 22: Beispiel Messaging-Konfiguration (Auszug).....	59
Listing 23: Beispiel Report Nachrichten Sender	60
Listing 24: Festlegen des Converters im RabbitTemplate Bean	61
Listing 25: Ausschnitt aus AbstractMessageReceiver	61
Listing 26: ReportCommand Interface	62
Listing 27: GetUserCommand.....	62
Listing 28: Beispiel BeanUtils.....	63
Listing 29: Beispiel GetProperty	63
Listing 30: Dynamisches Type-Casting.....	64
Listing 31: Definition einer Spalte in einem BIRT Dataset.....	64
Listing 32: Beispiel Methoden-Definition in einem BIRT Dataset	65
Listing 33: Vorbereiten des Report Templates im Code.....	66
Listing 34: Vorbereiten der Report-Parameter	66
Listing 35: Übergabe der Daten an ein Dataset	67
Listing 36: Erstellen der PDF Datei	67
Listing 37: Mögliche Lösung für das Handling von Listen.....	69

Listing 38: Beispiel UI Konfiguration.....	70
Listing 39: Hinzufügen eines Menu-Elementes mit entsprechender Rolle	70
Listing 40: Mock-Up von Klassen.....	72
Listing 41: Initialisierung des Test-Objekts und der Mocks.....	72
Listing 42: Anlegen benötigter Objekte mit Daten.....	72
Listing 43: Erzeugen eines Testobjekts der Klasse User	73
Listing 44: Simulieren von Objekten die von der Datenbank geladen werden.....	73
Listing 45: Überprüfen der Test-Resultate	73

11.3 Anhang C: Tabellenverzeichnis

Tabelle 1: Verwendete BIRT Version	14
Tabelle 2: Nutzwert-Analyse Scheduler	29

11.4 Anhang D: Abkürzungsverzeichnis

BIRT	Business Intelligence Reporting Tools. Tool von für die Erstellung von Reports
CSS	Stylesheet Datei für Web Dokumente (Cascading Stylesheet)
DCO	DataCollectionObject
DMS	Document Management System
DIE	Integrated Development Environment
IDM	Installation Data Management.
KSR	Kundenspezifischer Report
MRE	MESPAS Reporting Engine
MW	MESPAS Web (Webplattform der MESPAS AG)
MS	Microservice
NWA	Nutzwertanalyse
PMS	Planned Maintenance System (System für geplante Wartungen)
POJO	Plain old Java Object
TDD	Test Driven Development

11.5 Anhang E: Quellen- und Literaturverzeichnis

- [1]: <http://quartz-scheduler.org/documentation/quartz-2.1.x/migration-guide>, abgerufen am 05.08.2015
- [2]: <https://de.wikipedia.org/wiki/Nutzwertanalyse>, abgerufen am 05.08.2015
- [3]: [https://de.wikipedia.org/wiki/Quartz_\(Framework\)](https://de.wikipedia.org/wiki/Quartz_(Framework)), abgerufen am 05.08.2015
- [4]: <https://de.wikipedia.org/wiki/BIRT>, abgerufen am 05.08.2015
- [5]: http://www.tutorialspoint.com/javamail_api/index.htm, abgerufen am 06.08.2015
- [6]: <http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.birt.doc%2Fbirt%2Fcon-HowToAddAJDBCdriver.html>, abgerufen am 11.08.2015
- [7]: <https://eclipse.org/birt/documentation/integrating/scripting.php>, abgerufen am 11.08.2015
- [8]: <https://www.rabbitmq.com/>, abgerufen am 04.09.2015
- [9]: <https://www.rabbitmq.com/tutorials/tutorial-three-java.html>, abgerufen am 04.09.2015
- [10] Sierra, Kathy, Bates, Bert: Java von Kopf bis Fuss, Köln, Bundesrepublik Deutschland, O'Reilly, 1. Auflage, 2006, Seite 186. ISBN 978-3-89721-448-4
- [11]: <http://docs.spring.io/autorepo/docs/spring/4.1.1.RELEASE/javadoc-api/org.springframework.context.annotation.Configuration.html>, abgerufen am 23.09.2015
- [12]: <https://de.wikipedia.org/wiki/JavaBeans>, abgerufen am 23.09.2015
- [13]: http://docs.spring.io/spring-amqp/reference/html/reference.html#jsonmessageconverter_and_jackson2jsonmessageconverter, abgerufen am 27.09.2015
- [14]: <https://commons.apache.org/proper/commons-beanutils/apidocs/org/apache/commons/beanutils/BeanUtils.html>, abgerufen am 27.09.2015

- [15]: <https://en.wikipedia.org/wiki/Given-When-Then>, abgerufen am 02.10.2015
- [16]: <http://www.ebgconsulting.com/blog/using-given-when-then-to-discover-and-validate-requirements-2/>, abgerufen am 02.10.2015
- [17]: Spillner, Andreas, Linz, Tilo: Basiswissen Softwaretest, Heidelberg, Bundesrepublik Deutschland, dpunkt.Verlag GmbH, 5. Auflage, 2012, Seite 77. ISBN 978-3-86490-024-2