Placeholder

Nur Seiten 6 bis 17 drucken!

A

Fa

Dokumentation zur betrieblichen Projektarbeit

Abgabetermin: 09. Dezember 2015

RESTful [** Produktname **] Management Konsole

API zum Steuern und Verwalten von [Produktname **] Instanzen**

Michael Kleimann

****

****

Prüflingsnummer: **********

****

****

****

****

TÜVRheinland® CERT ISO 9001

Placeholder

[** Copyrighthinweis **]

Erklärung

Den in der Themenstellung beschriebenen Auftrag/Teilauftrag habe ich eigenständig in der beschriebenen Art und Weise durchgeführt. Die Dokumentation wurde ohne fremde Hilfe verfasst. Ich habe mich keiner anderen als der von mir angegebenen Hilfsmittel bedient.

_____

Michael Kleimann (Prüfling)

Bestätigung

Hiermit wird bestätigt, dass der Auftrag/Teilauftrag der Themenstellung so durchgeführt wurde, wie er in der Dokumentation beschrieben wurde.

_____

*[**Ausbilder**] (Ausbilder)*

Inhaltsverzeichnis

1 Einleitung

Die folgende Projektdokumentation wurde im Rahmen des IHK Abschlussprojektes erstellt, welches der Autor während seiner Ausbildung zum Fachinformatiker Fachrichtung Anwendungsentwicklung durchgeführt hat.

Der Ausbildungsbetrieb war das [**Betrieb**] in *******, Luxemburg.

1.1 Abweichungen zum Projektantrag

Da der Projektantrag erst Ende September von der IHK Trier genehmigt wurde, war der geplante Bearbeitungszeitraum vom 07.09. bis zum 02.10.2015 nicht einzuhalten.

Stattdessen wurde das Projekt in dem Zeitraum vom 05.10. bis zum 03.11.2015 durchgeführt.

1.2 Projektumfeld

Der Auftraggeber des Projektes ist das [**Betrieb**].

Das [**Betrieb**] ist ein Software- und Consultingunternehmen mit etwa 30 Mitarbeitern dessen Software von der ********* vertrieben wird. Zu dem Kundenkreis der ********* gehören überwiegend Unternehmen der Prozessindustrie wie *****, ***** und *****.

Zu den wichtigsten Produkten des Softwarehauses gehört „[**Produkt**]“, bestehend aus den Grundmodulen

- MES (Manufacturing Execution System)
- OEE (Overall Equipment Effectiveness) und
- WDS (Wiege- und Dosiersystem),

welche sich je nach Kundenwünschen anpassen und erweitern lässt.

1.3 Projektziel

Ziel des Projektes ist die Implementierung einer REST konformen HTTP Schnittstelle für die [**Produkt**] Management Konsole.

Firmenintern stehen für die Entwickler des [**Betrieb**] Server zur Verfügung, auf denen sie ihre [**Produkt**] Produktentwicklungen ausführen und testen können. Pro Server werden in der Regel nicht mehr als 20 [**Produkt**] Instanzen ausgeführt.

Zum Verwalten, Überwachen und Steuern der [**Produkt**] Server steht eine Management Konsole in Form einer Weboberfläche zur Verfügung. Dort können die Entwickler unter anderem Datenbankeinstellungen vornehmen, Logfiles ansehen und das zu startende Projekt konfigurieren und kompilieren.

Die Grundfunktionalitäten dieser Management Konsole sollen nun von einer REST API übernommen werden. Diese soll zunächst als Prototyp entwickelt werden.

Zur Veranschaulichung wurde ein Use-Case Diagramm erstellt, welches im Anhang „A1 Use-Case Diagramm“ zu finden ist.

1.4 Projektbegründung

Die bereits vorhandene Lösung ist fehleranfällig und schwer zu warten. Zudem haben andere Anwendungen keine Möglichkeit mit der jetzigen Management Konsole zu kommunizieren.

Durch eine saubere REST Schnittstelle können moderne Clients entwickelt werden, z.B. ein Plugin für die Entwicklungsumgebung Eclipse oder eine mobile App für Firmenhandys.

Dies hätte auf jeden Fall eine bessere Usability und eine nicht zu unterschätzende Zeitersparnis für Entwickler im Vergleich zur derzeitigen Lösung zur Folge.

Zudem wäre eine Weiterentwicklung der Management Konsole in Zukunft einfacher sofern eine saubere Architektur erschaffen wird.

1.5 Projektschnittstellen

Der Autor wird während des Projektes von [**Ausbilder**] und [**Mitarbeiter**] betreut.

Die Endbenutzer der Anwendung sind [**Produkt**] Entwickler des [**Betrieb**].

Damit der Status der [**Produkt**] Server zuverlässig abgefragt werden kann, muss die Anwendung über die *JBoss* API mit den [**Produkt**] Servern kommunizieren. Außerdem muss die Anwendung Projekte aus dem Subversion Repository auschecken können.

1.6 Projektabgrenzung

Die zeitliche Abgrenzung wurde von der IHK Trier mit 70 Stunden festgelegt.

Das Projekt soll nicht die bestehende Management Konsole erweitern, sondern die Management Konsole soll als REST API neu entwickelt werden.

Lediglich ein paar Kernklassen, welche für die direkte Kommunikation mit *JBoss* verantwortlich sind, sowie der *SVN* Client werden aus der bestehenden Management Konsole übernommen und sind somit nicht Bestandteil des Abschlussprojektes.

2 Projektplanung

2.1 Projektphasen

Für die Umsetzung des Projektes standen 70 Stunden zur Verfügung. Vor dem Beginn des Projektes wurde diese Zeit in verschiedene Phasen aufgeteilt, die während des Softwareentwicklungsprozesses durchlaufen werden. Das Projekt wurde während der Tagesarbeitszeit beim [**Betrieb**] realisiert. Der Tabelle 1 „Grobe Zeitplanung“ lassen sich die Hauptprojektphasen inklusive grober Zeitplanung entnehmen. Eine detaillierte Zeitplanung ist im Anhang „A2 Engineering Plan“ einzusehen.

Analysephase	4 h
Entwurfsphase	13 h
Implementierung	18 h
Qualitätsmanagement	13 h
Erstellen der Dokumentationen	22 h
Gesamt	**70 h**

Tabelle 1: Grobe Zeitplanung

2.2 Ressourcenplanung

Im Anhang „A3 Verwendete Ressourcen“ sind alle Ressourcen aufgelistet, die für dieses Projekt eingesetzt wurden. Damit sind Hard- und Softwareressourcen sowie der Personaleinsatz gemeint.

Um die anfallenden Projektkosten möglichst gering zu halten, wurde bei der verwendeten Software darauf geachtet, dass diese nach Möglichkeit kostenlos zur Verfügung steht, oder das [**Betrieb**] bereits die Lizenzen für diese besitzt.

2.3 Entwicklungsprozess

Vor der Realisierung des Projektes musste sich der Autor für einen geeigneten Entwicklungsprozess entscheiden. Dieser definiert die Vorgehensweise, nach welcher die Software entwickelt werden soll.

Das Projekt wurde aufgrund des geringen Umfangs nach dem erweiterten Wasserfallmodell umgesetzt, welches im Gegensatz zu dem klassischen Wasserfallmodell Rücksprünge zu vorhergehenden Phasen erlaubt und somit iterative Aspekte einführt.

Dieses Vorgehensmodell erlaubt es, Teile der Testphase schon vor bzw. während der Implementierungsphase durchzuführen, oder nach der Testphase nochmal zur Implementierungsphase zurückzuspringen.

(Wasserfall-Modell - Scrum Kompakt, 2015)

Die Projektphasen sind dem Abschnitt „2.1 Projektphasen“ zu entnehmen.

3 Analysephase

3.1 Ist-Analyse

Wie bereits im Abschnitt „1.4 Projektbegründung“ erwähnt, werden [**Produkt**] Server Instanzen bisher über eine webbasierte Management Konsole verwaltet. Es gibt keine Schnittstelle zur Kommunikation mit anderen Anwendungen und das veraltete Webinterface ist fehleranfällig und schwer zu warten.

Da die meisten Funktionalitäten der bisherigen Management Konsole fest in *JSP* Custom Tags kodiert sind, sollen diese neu implementiert werden. Nur die Schnittstelle zu *JBoss* (dem Unterbau der [**Produkt**] Server) und der *SVN* Client sollen von der alten Management Konsole übernommen werden.

3.2 Wirtschaftlichkeitsanalyse

3.2.1 „Make or Buy“ – Entscheidung

Da die [**Produkt**] Management Konsole sehr spezifische Anforderungen hat und durch die stetige Weiterentwicklung von [**Produkt**] laufend neue Anforderungen dazukommen, gibt es kein Produkt auf dem Markt welches für diesen Einsatz in Frage kommen würde. Aus diesem Grund wird dieses Projekt intern umgesetzt.

3.2.2 Projektkosten

Im Folgenden werden die anfallenden Projektkosten berechnet.

Da die genauen Personalkosten nicht herausgegeben werden dürfen, wird die Kalkulation anhand von frei erfundenen Stundensätzen durchgeführt. Neben den reinen Personalkosten müssen auch die Aufwendungen für Ressourcen und andere Kosten in Form von Gemeinkosten berücksichtigt werden. Diese Werte sind ebenfalls fiktiv.

Es wird von folgenden Stundensätzen und Gemeinkosten ausgegangen:

- Auszubildender: 20 €
- Entwickler: 60 €
- Entwicklungsleiter: 90 €
- Lohngemeinkosten: 70 %
- Verwaltungsgemeinkosten: 10 %

Die Lohngemeinkosten enthalten neben den üblichen Gemeinkosten auch die Kosten, die durch die Nutzung von Hardware und Software entstehen. Die Verwaltungsgemeinkosten decken die Kosten die für die Verwaltung anfallen.

Eine Aufstellung der Personalkosten befindet sich in der nachfolgenden Tabelle:

Vorgang	**Mitarbeiter**	**Zeit**	**Gesamt**
Entwicklung	1x Auszubildender	70 h	1400 €
Abnahmetest	2x Auszubildender 1x Entwickler	2 h	200 €
Abnahme	1x Entwicklungsleiter	1 h	90 €
Gesamt			**1690 €**

Tabelle 2: Personalkosten

Die Personalkosten betragen insgesamt **1690 €**.

Lohngemeinkosten: 1690 € * 0,7 = 1183 €

Verwaltungsgemeinkosten: 1690 € * 0,1 = 169 €

Daraus ergibt sich die folgende Kalkulation der gesamten Projektkosten:

```
  1690 €
+ 1183 €
+  169 €
----------
  3042 €
```

Die kalkulierten Gesamtkosten des Projekts belaufen sich insgesamt auf **3042 €**.

3.2.3 Amortisation

Es ist schwierig die Wirtschaftlichkeit dieses Projektes zu errechnen, da kein sofortiger Nutzen erkennbar ist. Da es sich um ein internes Projekt handelt, werden die im vorherigen Abschnitt errechneten Projektkosten nicht durch einen Kundenauftrag gedeckt.

Das Ergebnis dieses Projektes stellt allerdings die Grundlage einer neuen Technik dar, die beispielsweise als Teil eines neuen Produktes die Entwicklungskosten wieder einbringen kann.

Dazu kommen noch einige nicht-monetäre Vorteile, wie zuvor in Abschnitt „1.4 Projektbegründung“ beschrieben.

3.3 Lastenheft

Am Ende der Analysephase wurde ein Lastenheft erstellt. Dieses umfasst alle Anforderungen des Auftraggebers an die zu erstellende Anwendung.

Ein Auszug aus diesem Lastenheft befindet sich im Anhang „A4 Lastenheft (Auszug)“.

4 Entwurfsphase

4.1 Zielplatform

Das Projekt soll, wie bereits zuvor erwähnt, als REST Webservice realisiert werden.

Da auf allen Servern auf denen eine Management Konsole läuft bereits Apache *Tomcat* vorinstalliert ist und der Kern zur Kommunikation mit *JBoss* bereits in *Java* geschrieben wurde, soll diese Software mit *Java EE* realisiert werden. Dies beschleunigt den Entwicklungsprozess und das Deployment.

Da [**Produkt**] Server nur auf *Windows* lauffähig sind, muss bei der Erstellung dieser Anwendung keine Rücksicht auf andere Betriebssysteme genommen werden.

4.2 Schnittstellendesign

Der Webservice soll nach dem REST Paradigma umgesetzt werden. REST steht für Representational State Transfer und ist keine Architektur, sondern definiert nur einige Prinzipien, welche einen Webservice „RESTful“ machen.

Diese Prinzipien wurden im Anhang „A5 REST Prinzipien“ näher erläutert und geben einige Grundfunktionalitäten der Schnittstelle vor.

4.3 Architekturdesign

JAX-RS ist eine API, die es stark vereinfacht REST Webservices in Java zu implementieren. *JAX-RS* selbst enthält nur Interfaces welche von anderen Bibliotheken, wie z.B. *Suns* Referenzimplementierung *Jersey* implementiert werden.

Zurzeit gibt es vier verbreitete *JAX-RS* Implementierungen auf dem Markt:

- *Apache CXF*
- *Sun Jersey*
- *JBoss RESTEasy*
- *Restlet*

Manche dieser Frameworks erweitern den *JAX-RS* Standard mehr oder weniger.

Da die Standard *JAX-RS* Funktionalitäten für dieses Projekt ausreichend sind, hat sich der Autor für die Referenzimplementierung *Jersey* entschieden. Der Vorteil ist, dass sich *Jersey* bei Bedarf mit einer beliebigen anderen *JAX-RS* Implementierung austauschen lässt, ohne dass der Code geändert werden muss.

Nutzt man allerdings eine Erweiterung aus einer anderen Implementierung, ist der Wechsel auf ein anderes Framework nicht ohne Anpassungen im Source Code möglich.

JAX-RS bietet Annotationen um Java Klassen als Web Ressourcen bereitzustellen. Die wichtigsten sind:

- `@Path` (gibt die relative URI zur Ressource an)
- `@Consumes` (mögliche Eingabeformate)
- `@Produces` (mögliche Ausgabeformate)
- `@GET, @POST, @PUT, @DELETE, @HEAD, @OPTIONS` (HTTP-Methode)
- `@QueryParam, @CookieParam, @FormParam, @HeaderParam, @MatrixParam, @PathParam, @BeanParam` (verschiedene Arten von Parametern)
  (javax.ws.rs API Documentation, 2015)

Mit diesen Annotationen wird die Kombination aus URL + URI und einer HTTP-Methode eindeutig auf eine Java Methode abgebildet.

Obwohl eine REST API normalerweise nur über das Verändern von Ressourcen kommuniziert, hat sich der Autor dazu entschlossen einige Operationen, wie z.B. das Starten und Stoppen von [**Produkt**] Servern, durch extra Links zu ermöglichen und damit von dem Paradigma abzuweichen. Normalerweise würde der Nutzer das Objekt so verändern, dass es den gewünschten Status erhält. Der Webservice würde daraufhin die nötigen Funktionen aufrufen, um den gesetzten Status zu erreichen. Da das Starten und Stoppen von Servern jedoch eine der Kernaufgaben der Management Konsole ist, und die gerade beschriebene Vorgehensweise nicht sehr praktikabel ist, sollten diese beiden Aktionen der Einfachheit halber über extra URIs aufzurufen sein. Wenn der Nutzer nun den Server mit der ID „******02“ starten wollte, müsste er folgende URI an die URL des Webservices anhängen: `/servers/******02/start`

4.4 Datenmodell

Um die REST Prinzipien aus dem Abschnitt „4.2 Schnittstellendesign“ umzusetzen, wurden die Ressourcen aus vier Klassen zusammengesetzt:

`<Entity>Resource.java` ← Regelt Zugriff auf die Ressource (URI, HTTP Methode, Repräsentation (mittels JAX-RS Annotationen))

`<Entity>Service.java` ← Enthält implementierte Logik der Ressourcen

`<Entity>FilterBean.java` ← Enthält gesetzte Query Parameter beim Zugriff

`<Entity>.java` ← POJO, reines Datenmodell

Ein Klassendiagramm welches dieses Schema mit zwei Ressourcen zeigt, befindet sich im Anhang „A6 Klassendiagramm: Resource/Service/Entity/FilterBean“. Dieses Klassendiagramm ließe sich beliebig durch zusätzliche Ressourcen erweitern.

Der Autor benötigt mindestens die folgenden Ressourcen um alle Prozesse darstellen zu können:

- `ServerResource` ([**Produkt**] Server, URI: `/servers`)
- `EmployeeResource` (Mitarbeiter, kann Server zugewiesen werden, URI: `/employees`)
- `CustomerResource` (Kunde, kann Server zugewiesen werden, URI: `/customers`)
- `DatabaseResource` (Datenbankkonfiguration, kann Server zugewiesen werden, URI: `/databases`)

Mitarbeiter und Kunden können Servern zugewiesen und zum Filtern benutzt werden. Die zugehörigen `Entity` Klassen hat der Autor im Anhang „A7 Klassendiagramm: Entity Datenmodell“ visuell dargestellt.

4.5 Geschäftslogik

Die Funktionalitäten zum Starten und Stoppen der [**Produkt**] Server sind bereits in der vorhandenen `Jboss` Klasse enthalten und müssen daher nicht extra implementiert werden.

Da es in der Vergangenheit ab und zu Probleme bei der Wiederverwendung einer Serverinstanz kam (z.B.: Ports falsch konfiguriert), soll das gesamte Verzeichnis der Instanz neu erstellt werden, anstatt nur das Projekt zu ändern. Zum Löschen einer Instanz muss dementsprechend das Verzeichnis der Instanz gelöscht werden, zum Erstellen muss nur ein Batch Installer Script aufgerufen werden, welches allerdings noch angepasst werden muss.

Das Installer Script war in der Vergangenheit dazu gedacht, eine bestimmte Anzahl an [**Produkt**] Instanzen in einem neu installierten *JBoss* Server anzulegen. Dazu gehört unter anderem die richtige Konfiguration der Ports, damit diese sich bei vielen Instanzen nicht überschneiden.

Statt der Schleife in dem Script welche alle Instanzen der Reihe nach anlegt, muss es so angepasst werden, dass es nur den Server anlegt dessen ID als Parameter übergeben wird.

Ein weiterer Aspekt ist das Speichern der Daten. Die Konfigurationen für Server, Datenbanken, Mitarbeiter und Kunden sollen als YAML abgespeichert werden. YAML ist ein Standard zum Serialisieren von Daten und für den Menschen leichter les- und editierbar als beispielsweise XML oder JSON. So können Konfigurationen einfach von Hand angepasst werden.

Um jedes Entity einfach speichern, laden, ändern und löschen zu können, muss eine einheitliche Schnittstelle für die YAML (De-)Serialisation entwickelt werden. Dies soll mit Hilfe der *YamlBeans* Bibliothek erfolgen.

4.6 Build Management

Für das Build Management hat sich der Autor für *Apache Maven* entschieden. *Maven* folgt dem Grundgedanken „convention over configuration“, d.h. wird nichts konfiguriert, werden die *Maven* Standard Einstellungen verwendet.

Für die Entwicklung eines Prototyps ist das optimal, denn es wird keine Konfiguration benötigt und Abhängigkeiten lassen sich in Sekundenschnelle zum Projekt hinzufügen und wieder entfernen indem XML Snippets wie das Folgende (siehe Abbildung 1) zur Konfigurationsdatei pom.xml hinzufügt werden:

```
<dependency>
  <groupId>com.owlike</groupId>
  <artifactId>genson</artifactId>
  <version>1.3</version>
</dependency>
```

Abbildung 1: Beispiel Maven Dependency

Maven lädt beim Build Vorgang automatisch alle abhänigigen Jars aus dem Maven Central Repository herunter, fügt sie dem Build Path hinzu und kompiliert das Projekt. Dies spart in der Praxis viel Zeit gegenüber dem manuellen Pflegen der Abhängigkeiten.

Für komplexere Build Vorgänge wäre *Ant* oder *Gradle* das Mittel der Wahl, jedoch ist hier *Maven* wegen der oben genannten Gründe vollkommen ausreichend. Im Gegensatz zu *Maven* muss der Entwickler bei der Nutzung von *Ant* die Build Tasks größtenteils selber schreiben, *Gradle* dagegen hat etwas von beiden Systemen. *Maven* ist sozusagen die „Plug & Play“ Variante und damit bestens zur Entwicklung eines Prototypen geeignet.

(Java Build Tools: Ant vs Maven vs Gradle, 2015)

4.7 Pflichtenheft

Am Ende der Entwurfsphase wurde mithilfe des Lastenhefts und der Vorarbeit aus der Entwurfsphase das Pflichtenheft verfasst. Dieses dient als Leitfaden für die Implementierung des Projektes.

Ein Auszug ist im Anhang „A8 Pflichtenheft (Auszug)“ zu finden.

5 Implementierungsphase

5.1 Implementierung der Datenstrukturen

Als erstes wurden abstrakte Elternklassen für die vier Typen von Klassen erstellt, die zusammen eine Ressource bilden. Dazu wurde das in der Entwurfsphase erstellte UML Klassendiagramm im Anhang „A6 Klassendiagramm: Resource/Service/Entity/FilterBean“ als Vorlage genommen. Dieses Klassendiagramm soll nur die Assoziationen zwischen diesen vier Typen von Klassen verdeutlichen. Es sind nur die Ressourcen `Server` und `Employee` in dem Diagramm abgebildet.

Danach wurden die `Entities` anhand des im Anhang „A7 Klassendiagramm: Entity Datenmodell“ abgebildeten Klassendiagramms implementiert.

5.2 Implementierung der Geschäftslogik

Die Geschäftslogik ist in den jeweiligen `Service` Klassen implementiert.

Zur (De-)Serialisation der zu speichernden Daten wurde die `YamlUtils` Klasse implementiert, welche von den `Service` Klassen aufgerufen wird. Sie enthält Methoden zum einfachen Speichern und Laden von Objekten unter Beachtung der vom Autor erstellten Runtime Annotations für Membervariablen wie `@Unique` und `@NotNull`. Bei Verletzung dieser Einschränkungen (z.B. mit `@NotNull` annotierte Membervariable ist `null`), wird eine entsprechende `Exception` geworfen und das Objekt wird nicht abgespeichert.

Um die I/O-Last möglichst gering zu halten und die Performance zu verbessern hat der Autor eine `Map` zur Realisierung eines einfachen Caches eingesetzt. Bei einem Lesevorgang wird immer zuerst geprüft, ob sich die angeforderten Daten nicht schon im Cache befinden. Falls ja, werden Sie direkt zurückgegeben. Bei jedem Schreibvorgang wird der Cache gelöscht.

Der Quellcode der `YamlUtils` Klasse ist im Anhang „A10 Quellcode: YamlUtils.java“ einzusehen.

5.2.1 Exception Handling

Die Klasse `GenericExceptionMapper` dient dazu, die bei der Nutzung der API auftretenden Exceptions im JSON Format zurückzugeben.

Sie soll nur unvorhergesehene Exceptions behandeln, alle anderen werden explizit geworfen und mit eigenen `*ExceptionMapper` Klassen behandelt und dem Nutzer angezeigt. Alle `*ExceptionMapper` Klassen implementieren `javax.ws.rs.ext.ExceptionMapper<Exception>` und sind mit `@Provider` annotiert, damit *JAX-RS* diese zur Laufzeit finden kann.

In der jeweiligen `*ExceptionMapper` Klasse wird ein `ErrorMessage` Objekt erzeugt, welches den Namen der `Exception`, den HTTP Status Code und die Beschreibung des Fehlers enthält. Diese `ErrorMessage` gibt die API dann im JSON Format zurück.

5.3 Implementierung der Schnittstellen

Die Implementierung der Schnittstellen wurde durch die abstrakten `Resource` Klassen realisiert. Eine detaillierte Beschreibung dazu befindet sich im Anhang „A9 Erläuterung der abstrakten `Resource-Klassen`“.

Als Beispiel wurde der Quellcode der Klasse `ServerResource` im Anhang „A11 Quellcode: ServerResource.java“ hinterlegt. Der Quellcode der Elternklasse `Resource` ist im Anhang „A12 Quellcode: Resource.java“ zu finden.

5.4 Verwendete Bibliotheken

Eine Übersicht der in diesem Projekt verwendeten Bibliotheken mit den jeweiligen Lizenzen befindet sich im Anhang „A13 Verwendete Bibliotheken“.

6 Qualitätsmanagement

6.1 Statische Quellcode Analyse

Um die Applikation robuster zu machen und von sporadisch auftretenden Fehlern zu befreien, wurde der Quellcode mit *FindBugs* analysiert.

Die gefundenen Fehler bezogen sich hauptsächlich auf die implizite Benutzung des Standard-Encodings und nicht geschlossene Streams. Letzteres kann unter Umständen zu Ressourcen-Lecks führen, was gerade bei dauerhaft laufenden Anwendungen wie dieser zu Problemen führen kann.

Nach der Analyse wurden die kritischen Fehler im Source Code gefunden und korrigiert.

6.2 Automatisierte Tests

Damit sichergestellt werden kann, dass die API mit allen möglichen Anfragen korrekt umgeht, wurden automatisierte Tests mit *RestAssured* und *JUnit 4* geschrieben.

Obwohl diese Tests wie Unit Tests geschrieben wurden, handelt es sich eigentlich um Integrationstests, da die korrekte Kommunikation zwischen Client und Server festgestellt werden soll.

Vor jedem einzelnen Test wird ein eingebetteter *Tomcat* Testserver gestartet und die entsprechenden Verzeichnisse werden aufgeräumt. Nach jedem Test wird der Server wieder heruntergefahren. Obwohl das deutlich Ressourcenintensiver ist, als den Server einmal zu starten, alle Tests durchzuführen und ihn erst danach wieder herunterzufahren, ist es sinnvoll die Tests so aufzubauen. Dadurch ist sichergestellt, dass immer mit sauberen Daten getestet wird, da die benötigten Daten für jeden einzelnen Test neu angelegt werden. Zudem ist so auch jeder Test isoliert und einzeln ausführbar und nicht abhängig von anderen Tests, zumal die Reihenfolge der Testausführungen nicht garantiert werden kann. Die Korrektheit der automatisierten Tests hat Vorrang vor der Performance.

Die Testklasse `CustomerResourceTest` ist im Anhang „A14 Quellcode: CustomerResourceTest.java“ abgebildet. Das Ergebnis der JUnit Tests befindet sich im Anhang „A15 Ergebnis der automatisierten Tests (Auszug)“.

6.3 Manuelle Tests

Zusätzlich zur statischen Quellcode Analyse und den automatisierten Integrationstests wurden manuelle Tests spezifiziert, welche von den Testern von Hand ausgeführt werden sollen.

Dazu wurde ein Testserver vorbereitet, dessen Anforderungen im Anhang „A17 Testpläne“ aufgelistet sind. Dort sind auch die ausgefüllten Testpläne zu finden.

Damit die Tester die API komfortabel testen können, wurde zusätzlich eine *Postman Collection* erstellt. Diese JSON Datei steht den Testern auf dem Testserver zur Verfügung und kann in *Postman* importiert werden. Diese Datei ist im Anhang „A16 Postman Collection“ zu finden. *Postman* ist eine Erweiterung für *Google Chrome*, die es einfach ermöglicht verschiedene HTTP Requests zu senden und damit unter anderem Web APIs zu testen. In der Collection sind alle benötigten Requests für die Testschritte enthalten.

Bei dem ersten Testlauf wurde festgestellt, dass der [**Produkt**] Server sich nicht durch den entsprechenden API Aufruf stoppen ließ. Daraufhin wurde der Fehler verbessert, bevor der nächste Testlauf gestartet wurde.

Die nächsten Testläufe verliefen wie erwartet ohne weitere Fehler.

6.4 Deployment

Das Deployment der Anwendung gestaltet sich sehr einfach, da bereits die gesamte Infrastruktur die die Anwendung braucht auf den Testservern vorhanden ist. Dazu gehören *JBoss 4.2.3 GA*, *Collab SVN Client*, *JDK 6u45*, *JRE 8u65* und *Tomcat* 8.0.

Die Applikation muss nur als WAR Datei gepackt und in das webapps/ Verzeichnis des *Tomcat* Servers gelegt werden. Zusätzlich muss eine mgmt.properties Konfigurationsdatei in conf/mgmt/ im *Tomcat* Verzeichnis abgelegt werden.

7 Dokumentation

Die Dokumentation besteht aus drei Teilen: Projektdokumentation, Benutzerdokumentation und Entwicklerdokumentation. Auf der im Anhang „A21 CD“ beigefügten CD ist neben diesen Dokumentationen auch der gesamte Quelltext der Anwendung hinterlegt.

7.1 Projektdokumentation

In der Projektdokumentation sind die einzelnen Phasen der Umsetzung des Projektes ausführlich beschrieben.

7.2 Entwicklerdokumentation

Die Entwicklerdokumentation wurde aus *JavaDoc* Kommentaren generiert und liegt im PDF und HTML Format vor. In der HTML Dokumentation sind zusätzlich automatisch generierte UML Klassendiagramme vorhanden.

Ein Auszug aus der PDF Entwicklerdokumentation befindet sich im Anhang „A18 Entwicklerdokumentation PDF (Auszug)“, ein Screenshot der HTML Dokumentation im Anhang „A19 Entwicklerdokumentation HTML (Screenshot)“.

7.3 Kundendokumentation

Die Kundendokumentation richtet sich an die Benutzer der API, in diesem Fall an Entwickler des [**Betrieb**]. Sie gibt einen Überblick über alle im System verfügbaren Ressourcen sowie deren speziellen Funktionalitäten. Ebenfalls vorhanden sind Erläuterungen zu den Ressourcen und ihre Bedeutung für das System. Die vollständige Kundendokumentation befindet sich als separates Dokument im Anhang.

8 Fazit

8.1 Abnahme

Nach Fertigstellung des Projekts wurde zur Abnahme ein Fachgespräch mit dem Entwicklungsleiter geführt. Das Projektziel wurde erreicht und das Projekt wurde den Anforderungen entsprechend zur vollen Zufriedenheit des Auftraggebers umgesetzt.

8.2 Soll / Ist Vergleich

Im Folgenden wird der geplante Zeitbedarf dem tatsächlichen Zeitbedarf gegenübergestellt.

In der Entwurfsphase wurde für den Entwurf der Schnittstelle und des Datenmodells jeweils eine Stunde mehr benötigt als geplant. Diese zusätzliche Zeit wurde bei der Erstellung des Pflichtenhefts eingespart.

In der Implementierungsphase wurde der Punkt Programmierschnittstelle hinzugefügt, da dieser in der Planung nicht extra aufgeführt wurde, sondern in die Implementierung des Datenmodells und der Geschäftslogik eingeplant war. Während des Projektverlaufs wurde dieser Punkt zunehmend wichtiger, sodass sich der Autor dazu entschloss diesen separat aufzuführen. Dadurch haben sich die Implementierung des Datenmodells und der Geschäftslogik um zwei bzw. eine Stunde verkürzt. Die Implementierung der Schnittstelle hat vier Stunden gedauert, daher hat die gesamte Implementierungsphase eine Stunde mehr in Anspruch genommen als zuvor geplant.

Beim Qualitätsmanagement hat die Erstellung der Modultests eine Stunde weniger Zeit beansprucht als geplant. Die Erstellung der manuellen Tests dauerte durch die Vorbereitung des Testservers eine Stunde länger als angenommen. Dafür konnte die Fehlerbehebung aber in zwei Stunden vorgenommen werden, statt in den geplanten vier. Das Abnahmegespräch wurde bei der Zeitplanung nicht berücksichtigt, konnte aber in einer Stunde durchgeführt werden. Die gesamte Qualitätsmanagement Phase war somit eine Stunde kürzer und gleicht die verlängerte Dauer der Implementierungsphase aus.

Die Entwicklerdokumentation konnte in drei statt in fünf Stunden fertiggestellt werden, da die meisten *JavaDoc* Kommentare bereits während der Implementierung in den Code eingeflossen sind. Dafür hat allerdings die Projektdokumentation zwei zusätzliche Stunden in Anspruch genommen.

Die Gegenüberstellung der Zeiten befindet sich im Anhang „A20 Soll / Ist Vergleich“.

8.3 Ausblick

Nachdem das Projekt nach kurzer Zeit einen funktionierenden Prototyp hervorgebracht hat, welcher mit geringem Aufwand um neue Funktionen erweitert werden kann und vielseitig nutzbar ist, soll daraus eine fertige Version werden die produktiv einsetzbar ist.

Außerdem ist die Unterstützung für die nächste [**Produkt**] Version 5 geplant, die im Laufe des nächsten Jahres veröffentlicht werden soll. Diese neue Generation basiert auf moderneren Technologien als die aktuelle Version 4 und ist mit den jetzigen Tools nicht kompatibel. Des Weiteren wäre eine LDAP Integration denkbar, so dass der Mitarbeiter der die Management Konsole bedient über seinen *Windows* Account automatisch dem Server zugewiesen wird, an dem er gerade arbeitet.

Abkürzungsverzeichnis

API *Application Programming Interface*
CRUD *Create Read Update Delete*
HATEOAS *Hypermedia As The Engine Of Application State*
HTML *Hyper Text Markup Language*
HTTP *Hyper Text Transfer Protocol*
JAX-RS *Java API for RESTful Web Services*
JDK Java Development Kit
JRE Java Runtime Environment
JSON *JavaScript Object Notation*
JSP *Java Server Pages*
LDAP *Lightweight Directory Access Protocol*
MIME *Multipurpose Internet Mail Extensions*
PDF *Portable Document Format*
REST *Representational State Transfer*
SVN *Subversion*
UML *Unified Modeling Language*
URI *Uniform Resource Identifier*
URL *Uniform Resource Locator*
WAR *Web Application Archive*
WWW *World Wide Web*
XML *Extensible Markup Language*
YAML *YAML Ain't Markup Language*

Literaturverzeichnis

Java Build Tools: Ant vs Maven vs Gradle. (13. Oktober 2015). Von http://technologyconversations.com/2014/06/18/build-tools/ abgerufen

javax.ws.rs API Documentation. (6. Oktober 2015). Von https://docs.oracle.com/javaee/6/api/javax/ws/rs/package-summary.html abgerufen

RESTful Architectural Principles. (6. Oktober 2015). Von https://www.safaribooksonline.com/library/view/restful-java-with/9780596809300/ch01s02.html abgerufen

Wasserfall-Modell - Scrum Kompakt. (12. Oktober 2015). Von http://www.scrum-kompakt.de/grundlagen-des-projektmanagements/wasserfall-modell/ abgerufen

Bildverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Anhangsverzeichnis

A1 Use-Case Diagramm

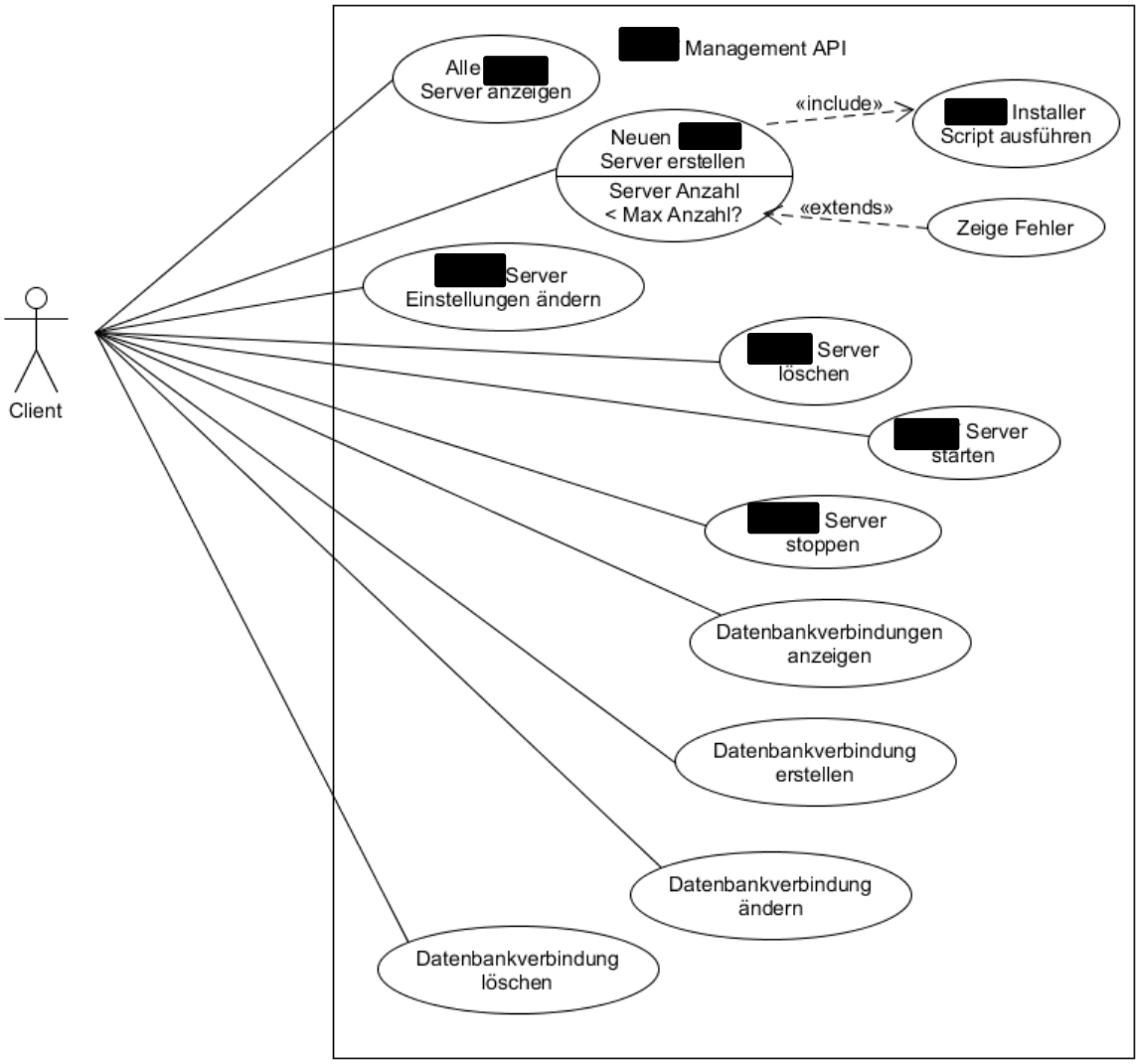

Abbildung 2: Use-Case Diagramm

A2 Engineering Plan

Projekt: FESWECM01-00
Kunde: intern

Tätigkeit	NZ	Kalenderwoche									
		38	39	40	41	42	43	44	45	46	47
Analysephase											
Soll-Analyse	MK				2,0						
Wirtschaftlichkeitsanalyse	MK				2,0						
Entwurfsphase											
Programmierschnittstelle	MK					2,0					
Datenmodell	MK				3,0	2,0					
Geschäftslogik	MK					2,0					
Pflichtenheft	MK					3,0					
Implementierung											
Datenmodell	MK						8,0				
Geschäftslogik	MK						4,0	6,0			
Qualitätsmanagement											
Modultests	MK							4,0			
Manuelle Tests erstellen	MK							5,0			
Manuelle Tests durchführen	**							1,0			
Manuelle Tests durchführen	**								1,0		
Manuelle Tests durchführen	**								1,0		
Fehlerbehebung	MK								4,0		
Abnahme	MK								1,0		
Abnahme	**								1,0		
Dokumentation											
Projektdokumentation	MK					3,0	3,0	3,0	3,0		
Entwicklerdokumentation	MK						3,0	2,0			
Benutzerdokumentation	MK								5,0		
		0,0	0,0	0,0	7,0	12,0	18,0	21,0	16,0	0,0	0,0

Tabelle 3: Engineering Plan

Namenskürzel:

MK – Michael Kleimann

** – [**Auszubildender**]

** – [**Auszubildender**]

** – [**Mitarbeiter**]

** – [**Ausbilder**]

A3 Verwendete Ressourcen

Hardware

- Büroarbeitsplatz
  - Notebook
  - Tastatur
  - Maus
  - Zweiter Monitor

Software

- Windows 7 Enterprise SP1 64 Bit
- Eclipse Java EE IDE Luna
- Java Development Kit 1.6.0 u45
- Java Development Kit 1.8.0 u65
- JBoss 4.2.3 GA
- Collab SVN Client 1.7
- Apache Tomcat 8.0
- [**Produkt**] Build Tools
- Microsoft Word 2013
- Microsoft Excel 2013
- NClass
- Paint.NET

Personal

- Entwicklungsleiter (Festlegung der Anforderungen, Abnahme Projekt)
- Auszubildender (Umsetzung des Projektes)
- 2x Auszubildender (manuelles Testen)
- Entwickler (manuelles Testen)

Alle oben aufgelisteten Ressourcen sind entweder bereits vorhanden oder unter einer freien Lizenz verfügbar, so dass nichts extra angeschafft werden muss.

A4 Lastenheft (Auszug)

Im folgenden Auszug aus dem Lastenheft werden Anforderungen definiert, die die zu realisierende Sofware erfüllen muss.

Anforderungen
Von der Anwendung müssen mindestens folgende Anforderungen erfüllt werden:

1. *Schnittstelle*
    a) Die Anwendung soll eine HTTP Schnittstelle nach dem REST Paradigma zur Verfügung stellen.
    b) Da die Anwendung nur im Intranet benutzt wird und jeder Zugriff auf alle Server haben soll, ist keine Authentifizierung notwendig.
    c) Die Schnittstelle soll einfach erweiterbar sein.

2. *Daten*
    a) Es sollen mindestens folgende Ressourcen verfügbar sein:
        1. [**Produkt**] Server
        2. Angestellter (n Angestellte nutzen m [**Produkt**] Server)
        3. Datenbank (n [**Produkt**] Server nutzen 1 Datenbank)
    b) [**Produkt**] Server sollen nach verschiedenen Attributen (z.B. Angestellter) gefiltert/gruppiert werden können.
    c) Beim Löschen eines [**Produkt**] Servers soll das entsprechende Server-Verzeichnis gelöscht werden.
    d) Beim Erstellen eines [**Produkt**] Servers soll ein Server-Verzeichnis angelegt werden.
    e) Eine maximale Anzahl an [**Produkt**] Servern muss konfiguriert werden können.
    f) Projekte soll anhand der Subversion URL ausgecheckt werden.
    g) Projekte sollen kompiliert werden können.

3. *Darstellung der Daten*
    a) Die Ressourcen sollen im JSON Format dargestellt werden.
    b) Der Server soll JSON vom Client konsumieren.

[…]

A5 REST Prinzipien

Adressierbarkeit

Die Adressierbarkeit erfordert, dass jede Ressource eines REST Webservices eindeutig über einen URI ansprechbar ist.

z.B.: `/servers/******02` ← spricht Ressource „Server“ mit der ID „******02“ an.

Einheitliche Schnittstelle

Dieses Prinzip besagt, dass die Standard Methoden des verwendeten Protokolls beim Zugriff auf Ressourcen die gewünschte Operation spezifizieren. Wird HTTP verwendet, so können die HTTP Methoden `POST`, `GET`, `PUT` und `DELETE` die CRUD Funktionen einer Ressource widerspiegeln. Hierbei muss auch darauf geachtet werden, dass `GET`, `HEAD`, `PUT` und `DELETE` laut HTTP-Spezifikation idempotent sein müssen, `POST` jedoch nicht.

Unterschiedliche Repräsentationen

Eine Ressource kann in unterschiedlichen Repräsentationen ausgeliefert werden, je nach dem welches Format der Client zurückbekommen möchte. Welches Format das ist spielt keine Rolle solange der Server dieses zurückgeben kann. Die häufigsten Formate die in REST Webservices verwendet werden sind JSON, XML, YAML, einfacher Text und HTML. Welches Format benutzt werden soll, legt der Client im HTTP Header als MIME Type fest. Für dieses Projekt ist zunächst nur die Unterstützung von JSON erforderlich.

Zustandslosigkeit

Zustandslosigkeit bedeutet, dass der Server keine Sessions kennt, sondern nur seine Ressourcen. Wenn Session spezifische Daten benötigt werden (beispielsweise zur Authentifizierung), dann muss der Client diese Daten bei jedem Request mitsenden.

HATEOAS

HATEOAS steht für Hypertext as the Engine of Application State. Die Idee kommt von den Anfängen des WWW. Der Anwender kennt nur die Adresse der Homepage einer Webseite und navigiert ab da nur noch via Hyperlinks durch die Unterseiten.

Mit HATEOAS wird angestrebt eine REST API navigierbar zu gestalten, das heißt eine Ressource enthält auch Links zu Subressourcen, verschiedenen Aktionen u.a.. Der Entwickler der später mit der API arbeitet, muss die URLs die er abfragen will nicht selbst zusammenbauen und weiß durch die beigefügten Links immer welche Aktionen erlaubt sind und welche nicht. Dies verhindert Anwenderfehler bei der Nutzung der API und verbessert die Usability.

(RESTful Architectural Principles, 2015)

A6 Klassendiagramm: Resource/Service/Entity/FilterBean

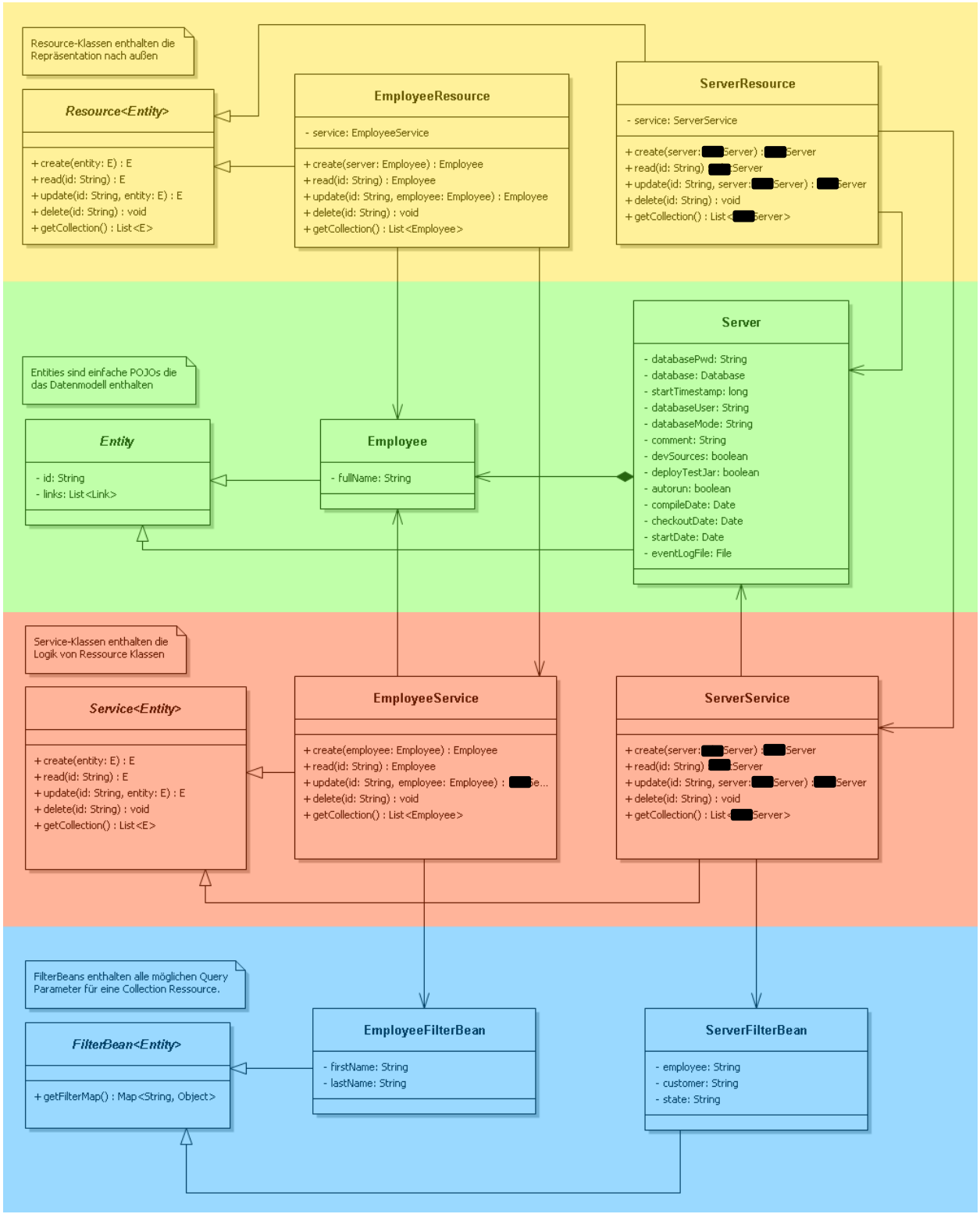

Abbildung 3: Klassendiagramm Resource/Service/Entity/FilterBean

- Gelb: `Resource` - Klassen
- Grün: `Entity` – Klassen (ohne get/set Methoden)
- Rot: `Service` - Klassen
- Blau: `FilterBean` – Klassen

A7 Klassendiagramm: Entity Datenmodell

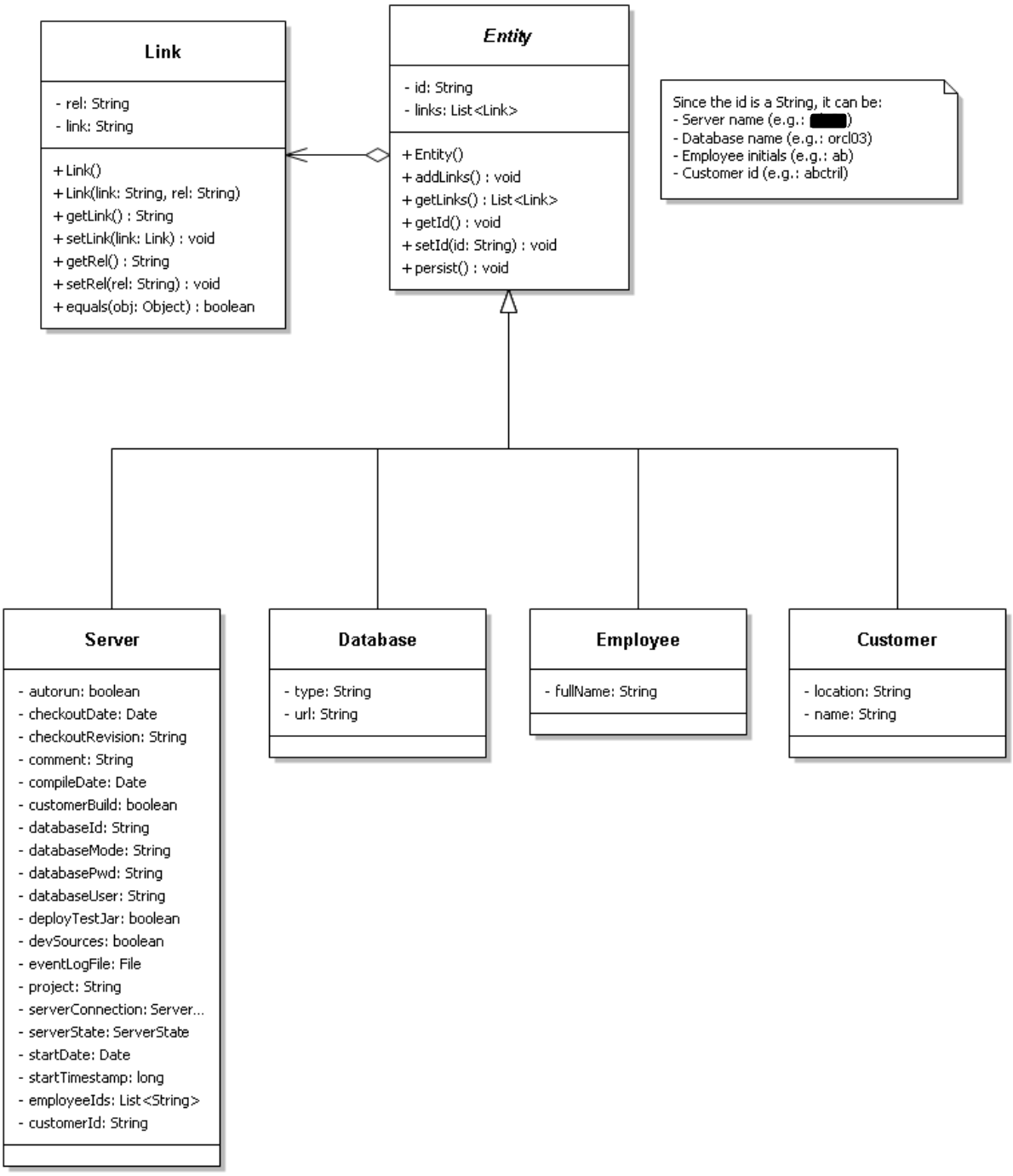

Abbildung 4: Klassendiagramm Entity Datenmodell (ohne get/set Methoden)

A8 Pflichtenheft (Auszug)

Zielbestimmung

1. *Musskriterien*

   a) Die Anwendung wird auf Basis des Jersey Frameworks implementiert.

   b) Um einheitliche Ressourcen zu gewährleisten, wird eine abstrakte Elternklasse `Resource` erstellt. Alle Ressourcen sollen von `Resource` ableiten. `Resource` definiert die CRUD Methoden mit generischen Rückgabetypen für jede Ressource. Zusätzlich wird eine Methode zur Rückgabe einer Liste mit allen Ressourcen eines Typs definiert. Zu jeder Ressource gehören ein `Service`, ein `FilterBean` und ein `Entity`.

   c) `Resource` - Klassen sollen selbst keine Logik implementieren. Sie rufen lediglich die entsprechenden Funktionen ihrer `Service` – Klasse.

   d) Das Filtern von Listen soll über Query Parameter erfolgen, welche an die URL angehangen werden. Die Verfügbaren Query Parameter werden in einer `FilterBean` Klasse definiert. Wird eine Liste abgefragt, werden die genutzten Query Parameter in der entsprechenden `FilterBean` gesetzt und vom `Service` ausgewertet.

   e) Zum Löschen und Anlegen von [**Produkt**] Server Instanzen inklusive der richtigen Konfiguration der Ports existiert bereits ein Batch Skript, welches lediglich angepasst werden muss.

   f) Über eine Konfigurationsdatei wird die maximale Anzahl von [**Produkt**] Servern konfiguriert werden können.

   g) Für das auschecken von Servern über Subversion sollte ein abstrahierter SVN Client implementiert werden.

   h) Exceptions werden gemappt, sodass sie im JSON Format zurückgegeben werden.

   i) Durch die Genson API werden Objekte einfach in das JSON Format serialisiert.

[…]

A9 Erläuterung der abstrakten Resource-Klassen

Eine einzelne Ressource setzt sich aus den vier Klassen `Entity`, `Resource`, `FilterBean` und `Service` zusammen. Welche Funktionen die einzelnen Klassen haben und wie sie zusammenhängen wird im Folgenden genauer beschrieben.

Entity.java

Alle `Entity` POJOs leiten von dieser Klasse ab. `Entity` implementiert `Serializable` um sicherzustellen, dass alle `Entity` POJOs in JSON serialisiert werden können.

Außerdem enthält `Entity` eine Liste von Links die beispielsweise auf die Ressource selbst und auf die Subressourcen verweisen. Dies dient dem HATEOAS Prinzip welches bereits im Abschnitt „A5 HATEOAS“ beschrieben wurde.

Resource.java

Alle Klassen die von `Resource` ableiten, müssen die CRUD Methoden implementieren. Die Klasse wird mit zwei Typen parametriert, dem zugehörigen `Entity` und der `FilterBean`:

```
@Produces(MediaType.APPLICATION_JSON)
@Consumes(MediaType.APPLICATION_JSON)
public abstract class Resource<E extends Entity, F extends FilterBean<E>>
```

Wird von `Resource` abgeleitet, steht z.B. in der folgenden Methode an Stelle des `E` eine konkrete `Entity` Klasse als Rückgabetyp:

```
@GET
@Path("/{id}")
public abstract E read( @PathParam( "id" ) String id );
```

Neben den CRUD Methoden wird noch `getCollection` definiert:

```
@GET
public abstract List<E> getCollection( @BeanParam F filterBean );
```

Da bei dieser Methode das Filtern der Ergebnisse über Query Parameter unterstützt werden soll, bekommt diese Methode eine `FilterBean` als Parameter reingereicht.

`Resource` Klassen enthalten keine Logik, sondern benutzen die entsprechenden `Service` Klassen. `Resource` definiert lediglich die Repräsentation und den Zugriff.

FilterBean.java

Um sicherzustellen, dass `FilterBean` Klassen nur mit `Entity` Klassen genutzt werden können, wird `FilterBean` ebenfalls mit einem `Entity` Typ parametrisiert.

```
public abstract class FilterBean<T extends Entity>
```

In `FilterBean` Klassen werden mögliche Query Parameter definiert, welche beispielsweise eine Liste von Ressourcen filtern können. Dazu müssen nur Member als `String` deklariert und mit `@QueryParam(<param-name>)` annotiert werden.

`FilterBean` implementiert selbst nur eine Methode, welche über die Java Reflection API die Member mit ihren Werten in einer `HashMap` speichert und zurückgibt. Mit dieser kann die Liste dann in der Service Klasse gefiltert werden.

Damit die Werte beim Zugriff gesetzt werden, muss die Methode in Resource, die Query Parameter unterstützen soll, `@BeanParam FilterBean` als Parameter nehmen, wie zuvor im Abschnitt „Resource.java“ beschrieben. Durch die `@BeanParam` Annotation und die `@QueryParam` Annotationen innerhalb der `FilterBean` Klasse werden die Parameter vom Framework durchgereicht und gesetzt.

Service.java

Das Interface `Service` ist der `Resource` Klasse sehr ähnlich. Auch `Service` nimmt ein `Entity` als Typparameter:

```
public interface Service<E extends Entity>
```

Die `Service` Klasse definiert die CRUD Methoden sowie die `getCollection` Methode, äquivalent zu `Resource`. Der Unterschied ist, dass das `Service` Interface keine JAX-RS Komponenten enthält, sondern nur die Logik für die Ressourcen bereitstellt.

Durch diese strikte Trennung von Logik und Repräsentation der Ressourcen ist der Code übersichtlich und leicht wart- und erweiterbar.

A10 Quellcode: YamlUtils.java

```
package com.***********.mgmt.api.utils;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.Field;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.stream.Stream;

import com.esotericsoftware.yamlbeans.YamlException;
import com.esotericsoftware.yamlbeans.YamlReader;
import com.esotericsoftware.yamlbeans.YamlWriter;
import com.***********.mgmt.api.annotations.Ignore;
import com.***********.mgmt.api.annotations.NotNull;
import com.***********.mgmt.api.annotations.Unique;
import com.***********.mgmt.api.entity.Entity;
import com.***********.mgmt.api.exception.NotNullViolatedException;
import com.***********.mgmt.api.exception.ResourceNotFoundException;
import com.***********.mgmt.api.exception.UniqueViolatedException;
import com.***********.mgmt.jboss.Jboss;

/**
 * The Singleton Class YamlUtils.
 */
public class YamlUtils
{

  /** The single instance. */
  private static YamlUtils instance;

  /** The cache. */
  private Map<Class<? extends Entity>, List<? extends Entity>> cache;

  /**
   * Instantiates YamlUtils.
   *
   * This constructor is private to prevent abuse. Use instance() to get the
instance.
   */
  private YamlUtils()
  {
    cache = new HashMap<Class<? extends Entity>, List<? extends Entity>>();
  }

  /**
   * Gets the single instance.
   *
   * @return the yaml utils
   */
  public static YamlUtils instance()
  {
    return instance == null ? ( instance = new YamlUtils() ) : instance;
  }
```

Listing 1: YamlUtils.java

```
  /**
   * Create an entity.
   *
   * @param Entity the entity to create
   * @throws NotNullViolatedException the not null violated exception
   * @throws UniqueViolatedException the unique violated exception
   */
  public void createEntity( Entity entity ) throws NotNullViolatedException,
UniqueViolatedException
  {
    try
    {
      if ( entity != null && annotationCheckOk( entity ) )
      {
        try (FileWriter fw = new FileWriter( getFilePath( entity.getClass()
), true ))
        {
          // add yaml object seperator
          fw.append( "---\r\n" );

          // write the object
          YamlWriter writer = new YamlWriter( fw );
          writer.write( entity );
          writer.close();

          // clear the cache
          clearCache();
        }
        catch ( IOException e1 )
        {
          e1.printStackTrace();
        }
      }
    }
    catch ( IllegalArgumentException | IllegalAccessException |
NoSuchFieldException
        | SecurityException e1 )
    {
      e1.printStackTrace();
    }
  }

  /**
   * Update entity.
   *
   * This method replaces an Entity, so partial update is not supported!
   *
   * @param e the e
   */
  public void updateEntity( Entity e )
  {
    deleteEntity( e.getClass(), e.getId(), false );
    createEntity( e );
  }

  /**
   * Get a single entity by class and id.
   *
   * @param clazz the clazz
   * @param id the id
   * @param throwOnError the throw on error
   * @return the entity */
```

```
  public Entity readEntity( Class<? extends Entity> clazz, String id, boolean
throwOnError )
  {
    List<? extends Entity> entities = readEntites( clazz );
    for ( Entity e : entities )
      if ( e != null && e.getId().equals( id ) )
        return e;

    if ( throwOnError )
      throw new ResourceNotFoundException( clazz, id );

    return null;
  }

  /**
   * Get all entities of a type.
   *
   * @param clazz the clazz
   * @return the entity[]
   */
  public List<? extends Entity> readEntites( Class<? extends Entity> clazz )
  {
    // try to read the entity from cache
    if ( cache.containsKey( clazz ) )
    {
      return cache.get( clazz );
    }
    else
    {
      String path = getFilePath( clazz );
      List<Entity> entities = new ArrayList<Entity>();

      // check if the config file exists
      if ( new File( path ).exists() )
      {
        try
        {
          // read and add the entities
          YamlReader reader = new YamlReader( new FileReader( path ) );
          Entity e = null;
          while ( ( e = reader.read( clazz ) ) != null )
          {
            entities.add( e );
          }
        }
        catch ( FileNotFoundException | YamlException e )
        {
          e.printStackTrace();
        }

        // write to cache
        cache.put( clazz, entities );
      }
      else
      {
        System.out.println( "Could not find file " + path );
      }
      return entities;
    }
  }
```

```
  /**
   * Get all entities with the specified ids.
   *
   * @param clazz the clazz
   * @param ids the ids
   * @param throwOnError the throw on error
   * @return the list<? extends entity> of ids
   */
  public List<? extends Entity> readEntites( Class<? extends Entity> clazz,
List<String> ids,
      boolean throwOnError )
  {
    List<Entity> entitiesById = new ArrayList<Entity>();
    for ( String id : ids )
      entitiesById.add( readEntity( clazz, id, throwOnError ) );

    return entitiesById;
  }

  /**
   * Delete entity.
   *
   * @param clazz the clazz
   * @param id the id
   * @param throwOnError throw exception on error
   */
  public void deleteEntity( Class<? extends Entity> clazz, String id, boolean
throwOnError )
  {
    readEntity( clazz, id, throwOnError );

    List<? extends Entity> entities = readEntites( clazz );
    clearConfig( clazz );
    for ( Entity e : entities )
      if ( !e.getId().equals( id ) )
        createEntity( e );
  }

  /**
   * Check if the entity can be serialized according to the constraint
annotations.
   *
   * @param entityToCheck the entity to check
   * @return true, if successful
   * @throws NotNullViolatedException the not null violated exception
   * @throws UniqueViolatedException the unique violated exception
   * @throws IllegalArgumentException the illegal argument exception
   * @throws IllegalAccessException the illegal access exception
   * @throws NoSuchFieldException the no such field exception
   * @throws SecurityException the security exception
   */
  private boolean annotationCheckOk( Entity entityToCheck ) throws
NotNullViolatedException,
      UniqueViolatedException, IllegalArgumentException,
IllegalAccessException,
      NoSuchFieldException, SecurityException
  {
    // collect all declared fields from class and superclass
  Field[] fields = Stream.concat( Arrays.stream(
entityToCheck.getClass().getDeclaredFields() ), Arrays.stream(
entityToCheck.getClass().getSuperclass().getDeclaredFields() ) ).toArray(
        Field[]::new );
```

```
    for ( Field f : fields )
    {
      // allow access to private fields
      f.setAccessible( true );

      // @Ignore annotation
      if ( f.getAnnotation( Ignore.class ) != null )
      {
        // set the value to null to prevent serialization
        f.set( entityToCheck, null );
      }

      // @NotNull annotation
      if ( f.getAnnotation( NotNull.class ) != null )
      {
        // throw Exception if value is null
        Object valueToCheck = f.get( entityToCheck );
        if ( valueToCheck == null )
          throw new NotNullViolatedException( f.getName() );
      }

      // @Unique annotation
      if ( f.getAnnotation( Unique.class ) != null )
      {
        // retrieve all entities of the same type
        List<? extends Entity> list = readEntites( entityToCheck.getClass()
);

        // check the value against all other entities
        Object valueToCheck;
        valueToCheck = f.get( entityToCheck );
        for ( Entity other : list )
        {
          Field otherField = f.getDeclaringClass().getDeclaredField(
f.getName() );
          otherField.setAccessible( true );
          Object otherValue = otherField.get( other );

          // throw exception if the value is not unique
          if ( otherValue.equals( valueToCheck ) )
            throw new UniqueViolatedException( f.getName(), otherValue );
        }
      }
    }
    // annotation check was ok
    return true;
  }

  /**
   * Gets the file path.
   *
   * It is constructed as follows: Jboss.instance().getConfigPath() +
   * clazz.getSimpleName().toLowerCase() + ".yml"
   *
   * @param clazz the clazz
   * @return the file path
   */
  private String getFilePath( Class<? extends Entity> clazz )
  {
    String path = Jboss.instance().getConfigpath();
```

```
    // add a slash if it is missing
    path = ( path.endsWith( "/" ) || path.endsWith( "\\" ) ) ? path : path +
"/";
    String fileName = clazz.getSimpleName().toLowerCase() + ".yml";
    return path + fileName;
  }

  /**
   * Clear the config for a type.
   *
   * @param clazz the clazz
   */
  private void clearConfig( Class<? extends Entity> clazz )
  {
    try (FileWriter fw = new FileWriter( getFilePath( clazz ), false ))
    {
      fw.write( "" );
      clearCache();
    }
    catch ( IOException e )
    {
      e.printStackTrace();
    }
  }

  /**
   * Clear cache.
   */
  public void clearCache()
  {
    cache.clear();
  }
}
```

A11 Quellcode: ServerResource.java

```
package com.***********.mgmt.api.resource;

import java.util.List;

import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriInfo;

import com.***********.mgmt.api.entity.Server;
import com.***********.mgmt.api.filter.ServerFilterBean;
import com.***********.mgmt.api.model.Link;
import com.***********.mgmt.api.service.ServerService;

/**
 * The Class ServerResource represents a [** PRODUKT **] server.
 */
@Path( "/servers" )
public class ServerResource extends Resource<Server, ServerFilterBean>
{

  /** The service. */
  ServerService service = new ServerService();

  /*
   * (non-Javadoc)
   *
   * @see
com.***********.mgmt.api.resources.Resource#create(com.***********.mgmt.api.e
ntity.Entity,
   * javax.ws.rs.core.UriInfo)
   */
  @Override
  public Response create( Server entity )
  {
    Server server = service.create( entity );
    addLinks( server );
    return Response.created( getUriInfo().getAbsolutePathBuilder().path(
server.getId() ).build() )
        .type( MediaType.APPLICATION_JSON ).entity( server ).build();
  }

  /*
   * (non-Javadoc)
   *
   * @see com.***********.mgmt.api.resources.Resource#read(java.lang.String,
   * javax.ws.rs.core.UriInfo)
   */
  @Override
  public Server read( String id )
  {
    Server server = service.read( id );
    addLinks( server );
    return server;
  }
```

Listing 2: ServerResource.java

```
  /*
   * (non-Javadoc)
   * @see [...]
   */
  @Override
  public Server update( String id, Server entity )
  {
    Server server = service.update( id, entity );
    addLinks( server );
    return server;
  }

  /*
   * (non-Javadoc)
   * @see [...]
   */
  @Override
  public void delete( String id )
  {
    service.delete( id );
  }

  /*
   * (non-Javadoc)
   * @see [...]
   */
  @Override
  public List<Server> getCollection( ServerFilterBean filterBean )
  {
    List<Server> result = service.getCollection( filterBean );
    result.forEach( s -> addLinks( s ) );
    return result;
  }

  /**
   * Start the ****** server.
   *
   * @param id the id
   * @return the ****** server
   *
   * @statuscode 200 if the start command was sent successfully.
   * @statuscode 404 if the server was not found.
   */
  @POST
  @Path( "/{id}/start" )
  @Produces( MediaType.APPLICATION_JSON )
  public Server start( @PathParam( "id" ) String id )
  {
    return service.start( id );
  }

  /**
   * Stop the ****** server.
   *
   * @param id the id
   * @return the ****** server
   *
   * @statuscode 200 if the stop command was sent successfully.
   * @statuscode 404 if the server was not found.
   */
```

```
  @POST
  @Path( "/{id}/stop" )
  @Produces( MediaType.APPLICATION_JSON )
  public Server stop( @PathParam( "id" ) String id )
  {
    return service.stop( id );
  }

  /**
   * Builds the project.
   *
   * @param id the id
   * @return the server
   *
   * @statuscode 200 if the project was build successfully.
   * @statuscode 400 if the server has some information missing which are
needed for compilation.
   * @statuscode 404 if the server was not found.
   */
  @POST
  @Path( "/{id}/build" )
  @Produces( MediaType.APPLICATION_JSON )
  public Server build( @PathParam( "id" ) String id )
  {
    return service.build( id );
  }

  /**
   * Adds the links.
   *
   * @param server the server
   * @param uriInfo the uri info
   */
  private void addLinks( Server server )
  {
    UriInfo uriInfo = getUriInfo();
    String selfLink = uriInfo.getBaseUriBuilder().path( ServerResource.class
)
        .path( server.getId() ).build().toString();
    server.addLinks( new Link( selfLink, "self" ) );
    if ( server.getProject() != null )
    {
      server.addLinks( new Link( selfLink + "/build", "build" ) );
      if ( server.startupIsEnabled() )
        server.addLinks( new Link( selfLink + "/start", "start" ) );
    }
    if ( server.shutdownIsEnabled() )
      server.addLinks( new Link( selfLink + "/stop", "stop" ) );
    if ( server.getDatabase() != null )
      server.addLinks( new Link( uriInfo.getBaseUriBuilder().path(
DatabaseResource.class )
          .path( server.getDatabase().getId() ).build().toString(),
"database" ) );
  }
}
```

A12 Quellcode: Resource.java

```
package com.***********.mgmt.api.resource;

import java.util.List;

import javax.ws.rs.BeanParam;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriInfo;

import com.***********.mgmt.api.entity.Entity;
import com.***********.mgmt.api.filter.FilterBean;

/**
 * The Class Resource represents web resources. The methods are annotated
with the corresponding
 * HTTP method and a path. This way, each resource method has it's unique
combination of path and
 * HTTP method and is called when sending a HTTP request to it.
 *
 * @param <E> the Entity
 * @param <F> the FilterBean
 */
@Produces( MediaType.APPLICATION_JSON )
@Consumes( MediaType.APPLICATION_JSON )
public abstract class Resource<E extends Entity, F extends FilterBean<E>>
{

  /** The uri info contains information about the uri and is used to build
the links for a resource. */
  @Context
  private UriInfo uriInfo;

  /**
   * Creates the resource.
   *
   * @param entity the entity
   * @return the response
   *
   * @statuscode 201 if the resource was created successfully.
   * @statuscode 400 if the id of the resource already exists.
   */
  @POST
  public abstract Response create( E entity );

  /**
   * Gets the resource.
   *
   * @param id the id
   * @return the entity
   *
```

Listing 3: Resource.java

```
     * @statuscode 200 if the resource was found.
     * @statuscode 404 if the resource does not exist.
     */
    @GET
    @Path( "/{id}" )
    public abstract E read( @PathParam( "id" ) String id );

    /**
     * Update the resource.
     *
     * @param id the id
     * @param entity the entity
     * @return the entity
     *
     * @statuscode 200 if the resource was found and modified.
     * @statuscode 404 if the resource does not exist.
     */
    @PUT
    @Path( "/{id}" )
    public abstract E update( @PathParam( "id" ) String id, E entity );

    /**
     * Delete the resource.
     *
     * @param id the id
     *
     * @statuscode 204 if the resource was found and deleted.
     * @statuscode 404 if the resource does not exist.
     */
    @DELETE
    @Path( "/{id}" )
    public abstract void delete( @PathParam( "id" ) String id );

    /**
     * Gets the collection. Query parameters are supported depending on the
according FilterBean.
     *
     * @param filterBean the filter bean
     * @return the collection
     *
     * @statuscode 200 if the collection was returned.
     */
    @GET
    public abstract List<E> getCollection( @BeanParam F filterBean );

    /**
     * Gets the uri info.
     *
     * @return the uri info
     */
    public UriInfo getUriInfo()
    {
      return uriInfo;
    }
}
```

A13 Verwendete Bibliotheken

Die untenstehende Übersicht wurde automatisch mit dem Maven Goal `project-info-reports:dependencies` generiert:

GroupId	ArtifactId	Version	Typ	Lizenz
com.esotericsoftware.yamlbeans	yamlbeans	1.09	jar	New BSD License
com.google.code.gson	gson	2.4	jar	The Apache Software License, Version 2.0
com.jayway.restassured	rest-assured	2.5.0	jar	Apache 2.0
com.outr.javasysmon	javasysmon_2.10	0.3.4	jar	NetBSD
com.owlike	genson	1.3	jar	Apache Software License Version 2.0
com.sun.jna	jna	3.0.9	jar	null
commons-beanutils	commons-beanutils	1.9.2	jar	The Apache Software License, Version 2.0
javax.servlet	servlet-api	2.5	jar	-
jboss	jbossall-client	4.2.2.GA	jar	-
junit	junit	4.12	jar	Eclipse Public License 1.0
org.apache.tomcat	tomcat-jasper	8.0.28	jar	Apache License, Version 2.0
org.apache.tomcat	tomcat-jasper-el	8.0.28	jar	Apache License, Version 2.0
org.apache.tomcat	tomcat-jsp-api	8.0.28	jar	Apache License, Version 2.0
org.apache.tomcat.embed	tomcat-embed-core	8.0.28	jar	Apache License, Version 2.0
org.apache.tomcat.embed	tomcat-embed-jasper	8.0.28	jar	Apache License, Version 2.0
org.apache.tomcat.embed	tomcat-embed-logging-juli	8.0.28	jar	Apache License, Version 2.0
org.glassfish.jersey.containers	jersey-container-servlet-core	2.22	jar	CDDL+GPL License
org.json	json	20141113	jar	The JSON License
org.slf4j	slf4j-simple	1.7.12	jar	MIT License

Abbildung 5: Übersicht verwendeter Bibliotheken und Lizenzen

A14 Quellcode: CustomerResourceTest.java

```
package com.***********.mgmt.api.test.resource;

import static com.jayway.restassured.RestAssured.delete;
import static com.jayway.restassured.RestAssured.get;
import static com.jayway.restassured.RestAssured.given;
import static javax.ws.rs.core.MediaType.APPLICATION_JSON;
import static org.hamcrest.Matchers.equalTo;

import org.json.JSONObject;
import org.junit.Test;

import com.***********.mgmt.api.entity.Customer;
import com.jayway.restassured.response.Response;

/**
 * The Class CustomerResourceTest.
 */
public class CustomerResourceTest extends ResourceTest
{
  /** The Constant URI_SERVER. */
  private static final String URI_CUSTOMER = "customers";

  /**
   * Test create.
   *
   * Expected: <br>
   * <ul>
   * <li>http status = 201</li>
   * <li>customer id = abctri</li>
   * <li>content type = application/json</li>
   * </ul>
   */
  @Test
  public void testCreate()
  {
    Response response = createTestCustomer();
    Customer customer = response.as( Customer.class );

    response.then().statusCode( 201 ).contentType( APPLICATION_JSON )
        .body( "id", equalTo( customer.getId() ) );
  }

  /**
   * Test read.
   *
   * * Expected: <br>
   * <ul>
   * <li>http status = 200</li>
   * <li>customer id = abctri</li>
   * <li>content type = application/json</li>
   * </ul>
   */
  @Test
  public void testRead()
  {
    // create test customer
    Customer customer = createTestCustomer().as( Customer.class );
```

Listing 4: CustomerResourceTest.java

```
    // test read
    get( URI_CUSTOMER + "/" + customer.getId() ).then().statusCode( 200 )
        .contentType( APPLICATION_JSON ).body( "id", equalTo(
customer.getId() ) );
  }

  /**
   * Test update.
   *
   * * Expected: <br>
   * <ul>
   * <li>http status = 200</li>
   * <li>customer id = abctri</li>
   * <li>content type = application/json</li>
   * </ul>
   */
  @Test
  public void testUpdate()
  {
    // create test customer
    Customer customer = createTestCustomer().as( Customer.class );
    customer.setLocation( "changed" );
    String updateJson = new JSONObject( customer ).toString();

    // update the customer
    given().contentType( APPLICATION_JSON ).body( updateJson )
        .put( URI_CUSTOMER + "/" + customer.getId() ).then().statusCode( 200
).contentType( APPLICATION_JSON ).body( "id", equalTo( customer.getId() ) )
        .body( "location", equalTo( "changed" ) );
  }

  /**
   * Test delete.
   *
   * * * Expected: <br>
   * <ul>
   * <li>http status = 204</li
   * <li>then: http status = 404</li>
   * </ul>
   */
  @Test
  public void testDelete()
  {
    // create test customer
    Customer customer = createTestCustomer().as( Customer.class );

    // delete the customer
    delete( URI_CUSTOMER + "/" + customer.getId() ).then().statusCode( 204 );

    // test read, should fail
    get( URI_CUSTOMER + "/" + customer.getId() ).then().statusCode( 404 );
  }
```

```
  /**
   * Creates the test customer object.
   *
   * @return the test customer
   */
  private Response createTestCustomer()
  {
    Customer customer = new Customer( "abctri", "ABC", "Trier" );
    String json = new JSONObject( customer ).toString();
    System.out.println( json );
    return given().body( json ).contentType( APPLICATION_JSON ).post(
URI_CUSTOMER ).andReturn();
  }
}
```

A15 Ergebnis der automatisierten Tests (Auszug)

```
<?xml version="1.0" encoding="UTF-8"?>
<testrun name="ResourceTestSuite" project="RestMgmt" tests="16" started="16"
failures="0" errors="0" ignored="0">
  <testsuite name="com.***********.mgmt.api.test.ResourceTestSuite"
time="168.733">
    <testsuite
name="com.***********.mgmt.api.test.resource.DatabaseResourceTest"
time="48.837">
      <testcase name="testRead"
classname="com.***********.mgmt.api.test.resource.DatabaseResourceTest"
time="23.837"/>
      <testcase name="testCreate"
classname="com.***********.mgmt.api.test.resource.DatabaseResourceTest"
time="8.59"/>
      <testcase name="testDelete"
classname="com.***********.mgmt.api.test.resource.DatabaseResourceTest"
time="8.004"/>
      <testcase name="testUpdate"
classname="com.***********.mgmt.api.test.resource.DatabaseResourceTest"
time="8.406"/>
    </testsuite>
    <testsuite
name="com.***********.mgmt.api.test.resource.EmployeeResourceTest"
time="29.749">
      <testcase name="testRead"
classname="com.***********.mgmt.api.test.resource.EmployeeResourceTest"
time="7.722"/>
      <testcase name="testCreate"
classname="com.***********.mgmt.api.test.resource.EmployeeResourceTest"
time="7.498"/>
      <testcase name="testDelete"
classname="com.***********.mgmt.api.test.resource.EmployeeResourceTest"
time="7.132"/>
      <testcase name="testUpdate"
classname="com.***********.mgmt.api.test.resource.EmployeeResourceTest"
time="7.397"/>
    </testsuite>
    <testsuite
name="com.***********.mgmt.api.test.resource.CustomerResourceTest"
time="29.375">
      <testcase name="testRead"
classname="com.***********.mgmt.api.test.resource.CustomerResourceTest"
time="7.573"/>
      <testcase name="testCreate"
classname="com.***********.mgmt.api.test.resource.CustomerResourceTest"
time="7.181"/>
      <testcase name="testDelete"
classname="com.***********.mgmt.api.test.resource.CustomerResourceTest"
time="7.131"/>
      <testcase name="testUpdate"
classname="com.***********.mgmt.api.test.resource.CustomerResourceTest"
time="7.49"/>
    </testsuite>
    <testsuite
name="com.***********.mgmt.api.test.resource.ServerResourceTest"
time="40.047">
      <testcase name="testRead"
```

Listing 5: Ergebnis der automatisierten Tests

A16 Postman Collection


```
{
    "id": "bb4414fa-92f1-0f53-7017-78198f944aa2",
    "name": "RestMgmt",
    "description": "",
    "order": [
        "6c562e08-153e-214e-a95c-f6b194a540c4",
        "884b270f-07f1-5a4d-dd45-0a7dec96af56",
        "02ea5de1-dd7b-e6ad-a944-92aec955e8e0",
        "8ddaba20-e0c6-ec48-74c6-9215a326a684",
        "33bdf363-b4c5-cd9b-ec55-031698799ad7",
        "7d9a163b-d1e8-a7cd-61d5-75b0cc8abfac",
        "cde4e72d-dbfa-2105-caab-e08fcf3337b1",
        "38a30fe7-0ed7-2ec2-2771-e56be0d38bc7",
        "cfd609c6-9042-0b0c-2ed8-33daf08f85b5",
        "07818b45-b7b6-da84-2f42-967a116c2dce",
        "d432e940-f3e0-4a6b-7f12-9a35e3bb26c6",
        "23439ef8-2843-ca86-b4a7-8179e9b1f3ac",
        "bbd36bff-553e-b25a-561f-ac633c3b32c7"
    ],
    "folders": [],
    "timestamp": 1445876369665,
    "owner": 0,
    "remoteLink": "",
    "public": false,
    "requests": [
        {
            "id": "02ea5de1-dd7b-e6ad-a944-92aec955e8e0",
            "headers": "Content-Type: application/json\n",
            "url": "http://restmgmt:88/mgmt/api/employees",
            "preRequestScript": "",
            "pathVariables": {},
            "method": "GET",
            "data": [],
            "dataMode": "params",
            "version": 2,
            "tests": "",
            "currentHelper": "normal",
            "helperAttributes": {},
            "time": 1445932797182,
            "name": "Test #1.3 - Get all employees",
            "description": "",
            "collectionId": "bb4414fa-92f1-0f53-7017-78198f944aa2",
            "responses": []
        },
        {
            "id": "07818b45-b7b6-da84-2f42-967a116c2dce",
            "headers": "Content-Type: application/json\n",
            "url": "http://restmgmt:88/mgmt/api/servers/******00/build",
            "preRequestScript": "",
            "pathVariables": {},
            "method": "POST",
            "data": [],
            "dataMode": "raw",
            "version": 2,
            "tests": "",
            "currentHelper": "normal",
```


Listing 6: Postman Collection für manuelle Tests

```
            "helperAttributes": {},
            "time": 1445934599105,
            "name": "Test #8.1 - Build (******00)",
            "description": "",
            "collectionId": "bb4414fa-92f1-0f53-7017-78198f944aa2",
            "responses": [],
            "rawModeData": ""
        },
        {
            "id": "23439ef8-2843-ca86-b4a7-8179e9b1f3ac",
            "headers": "Content-Type: application/json\n",
            "url": "http://restmgmt:88/mgmt/api/servers/******00/start",
            "preRequestScript": "",
            "pathVariables": {},
            "method": "POST",
            "data": [],
            "dataMode": "raw",
            "version": 2,
            "tests": "",
            "currentHelper": "normal",
            "helperAttributes": {},
            "time": 1445936737377,
            "name": "Test #9 - Start (******00)",
            "description": "",
            "collectionId": "bb4414fa-92f1-0f53-7017-78198f944aa2",
            "responses": [],
            "rawModeData": ""
        },
        {
            "id": "33bdf363-b4c5-cd9b-ec55-031698799ad7",
            "headers": "Content-Type: application/json\n",
            "url": "http://restmgmt:88/mgmt/api/customers",
            "preRequestScript": "",
            "pathVariables": {},
            "method": "POST",
            "data": [],
            "dataMode": "raw",
            "version": 2,
            "tests": "",
            "currentHelper": "normal",
            "helperAttributes": {},
            "time": 1446029130185,
            "name": "Test #3 - Add customer",
            "description": "",
            "collectionId": "bb4414fa-92f1-0f53-7017-78198f944aa2",
            "responses": [],
            "rawModeData": "{\"id\":\"abcsome\", \"name\":\"ABC\",
\"location\":\"Some Town\"}"
        },
        {
            "id": "38a30fe7-0ed7-2ec2-2771-e56be0d38bc7",
            "headers": "Content-Type: application/json\n",
            "url": "http://restmgmt:88/mgmt/api/servers/",
            "pathVariables": {},
            "preRequestScript": "",
            "method": "POST",
            "collectionId": "bb4414fa-92f1-0f53-7017-78198f944aa2",
            "data": [],
            "dataMode": "raw",
            "name": "Test #6 - Add server (******01)",
```

```
            "description": "",
            "descriptionFormat": "html",
            "time": 1445933902175,
            "version": 2,
            "responses": [],
            "tests": "",
            "currentHelper": "normal",
            "helperAttributes": {},
            "rawModeData": "{\"employeeIds\":[\"tg\"]}"
        },
        {
            "id": "6c562e08-153e-214e-a95c-f6b194a540c4",
            "headers": "Content-Type: application/json\n",
            "url": "http://restmgmt:88/mgmt/api/employees",
            "preRequestScript": "",
            "pathVariables": {},
            "method": "POST",
            "data": [],
            "dataMode": "raw",
            "version": 2,
            "tests": "",
            "currentHelper": "normal",
            "helperAttributes": {},
            "time": 1445939796740,
            "name": "Test #1.1 - Add employee (tg)",
            "description": "",
            "collectionId": "bb4414fa-92f1-0f53-7017-78198f944aa2",
            "responses": [],
            "rawModeData": "{\"id\":\"tg\", \"fullName\":\"Torsten
Grunwald\"}"
        },
        {
            "id": "7d9a163b-d1e8-a7cd-61d5-75b0cc8abfac",
            "headers": "Content-Type: application/json\n",
            "url": "http://restmgmt:88/mgmt/api/servers",
            "preRequestScript": "",
            "pathVariables": {},
            "method": "POST",
            "data": [],
            "dataMode": "raw",
            "version": 2,
            "tests": "",
            "currentHelper": "normal",
            "helperAttributes": {},
            "time": 1445933461953,
            "name": "Test #4 - Add server (******00)",
            "description": "",
            "collectionId": "bb4414fa-92f1-0f53-7017-78198f944aa2",
            "responses": [],
            "rawModeData": "{\"employeeIds\":[\"tg\", \"fe\"],
\"databaseId\":\"hsqldb\"}"
        },
        {
            "id": "884b270f-07f1-5a4d-dd45-0a7dec96af56",
            "headers": "Content-Type: application/json\n",
            "url": "http://restmgmt:88/mgmt/api/employees",
            "preRequestScript": "",
            "pathVariables": {},
            "method": "POST",
            "data": [],
            "dataMode": "raw",
            "version": 2,
```

```
            "tests": "",
            "currentHelper": "normal",
            "helperAttributes": {},
            "time": 1445932558064,
            "name": "Test #1.2 - Add employee (fe)",
            "description": "",
            "collectionId": "bb4414fa-92f1-0f53-7017-78198f944aa2",
            "responses": [],
            "rawModeData": "{\"id\":\"fe\", \"fullName\":\"Felix Eberhard\"}"
        },
        {
            "id": "8ddaba20-e0c6-ec48-74c6-9215a326a684",
            "headers": "Content-Type: application/json\n",
            "url": "http://restmgmt:88/mgmt/api/databases",
            "pathVariables": {},
            "preRequestScript": "",
            "method": "POST",
            "collectionId": "bb4414fa-92f1-0f53-7017-78198f944aa2",
            "data": [],
            "dataMode": "raw",
            "name": "Test #2 - Add database",
            "description": "",
            "descriptionFormat": "html",
            "time": 1445933011091,
            "version": 2,
            "responses": [],
            "tests": "",
            "currentHelper": "normal",
            "helperAttributes": {},
            "rawModeData": "{\"id\":\"hsqldb\", \"type\":\"hsqldb\"}"
        },
        {
            "id": "bbd36bff-553e-b25a-561f-ac633c3b32c7",
            "headers": "Content-Type: application/json\n",
            "url": "http://restmgmt:88/mgmt/api/servers/******00/stop",
            "preRequestScript": "",
            "pathVariables": {},
            "method": "POST",
            "data": [],
            "dataMode": "raw",
            "version": 2,
            "tests": "",
            "currentHelper": "normal",
            "helperAttributes": {},
            "time": 1445934996686,
            "name": "Test #10 - Stop (******00)",
            "description": "",
            "collectionId": "bb4414fa-92f1-0f53-7017-78198f944aa2",
            "responses": [],
            "rawModeData": ""
        },
        {
            "id": "cde4e72d-dbfa-2105-caab-e08fcf3337b1",
            "headers": "Content-Type: application/json\n",
            "url": "http://restmgmt:88/mgmt/api/servers/******00",
            "preRequestScript": "",
            "pathVariables": {},
            "method": "PUT",
            "data": [],
            "dataMode": "raw",
```

```
            "version": 2,
            "tests": "",
            "currentHelper": "normal",
            "helperAttributes": {},
            "time": 1446029156022,
            "name": "Test #5 - Update server",
            "description": "",
            "collectionId": "bb4414fa-92f1-0f53-7017-78198f944aa2",
            "responses": [],
            "rawModeData": "{\"customerId\":\"abcsome\",
\"project\":\"https://ewsvr.***********.local:8443/svn/******-
core/pmscada/branches/4.15.1/\"}"
        },
        {
            "id": "cfd609c6-9042-0b0c-2ed8-33daf08f85b5",
            "headers": "",
            "url": "http://restmgmt:88/mgmt/api/servers?employee=fe",
            "pathVariables": {},
            "preRequestScript": "",
            "method": "GET",
            "collectionId": "bb4414fa-92f1-0f53-7017-78198f944aa2",
            "data": [],
            "dataMode": "params",
            "name": "Test #7 - Filter servers",
            "description": "",
            "descriptionFormat": "html",
            "time": 1445936026981,
            "version": 2,
            "responses": [],
            "tests": "",
            "currentHelper": "normal",
            "helperAttributes": {}
        },
        {
            "id": "d432e940-f3e0-4a6b-7f12-9a35e3bb26c6",
            "headers": "Content-Type: application/json\n",
            "url": "http://restmgmt:88/mgmt/api/servers/******00/",
            "preRequestScript": "",
            "pathVariables": {},
            "method": "PUT",
            "data": [],
            "dataMode": "raw",
            "version": 2,
            "tests": "",
            "currentHelper": "normal",
            "helperAttributes": {},
            "time": 1445934665433,
            "name": "Test #8.2 - Add missing information (******00)",
            "description": "",
            "collectionId": "bb4414fa-92f1-0f53-7017-78198f944aa2",
            "responses": [],
            "rawModeData": "{\"databaseMode\":\"CREATE\"}"
        }
    ]
}
```

A17 Testpläne

*** ENTFERNT ***

A18 Entwicklerdokumentation PDF (Auszug)

com.[redacted].mgmt.api.filter
Class FilterBean

```
java.lang.Object
   |
   +-com.[redacted].mgmt.api.filter.FilterBean
```

Direct Known Subclasses:
CustomerFilterBean, DatabaseFilterBean, EmployeeFilterBean, ServerFilterBean

public abstract class **FilterBean**
extends java.lang.Object

The Class FilterBean. In FilterBean subclasses you can define fields, which can be used as query parameters then (e.g. "/someurl?queryparam=value"). The corresponding Service class has access to these parameters by accessing the SomeFilterBean.getFilterMap() method.
Parameters:
E - the entity

Constructor Summary

public	FilterBean()

Method Summary

java.util.Map	getFilterMap() Get all set query parameters as HashMap.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, registerNatives, toString, wait, wait, wait

Constructors

FilterBean

```
public FilterBean()
```

Methods

getFilterMap

```
public java.util.Map getFilterMap()
```

Get all set query parameters as HashMap.
Note: This method uses reflection to provide the values.

Returns:
the filtered resources

Page 50 of 97

A19 Entwicklerdokumentation HTML (Screenshot)

Abbildung 6: Entwicklerdokumentation HTML (Screenshot)

A20 Soll / Ist Vergleich

Phase	Geplant	Tatsächlich	Differenz
Analysephase	**4 h**	**4 h**	
Soll-Analyse	2 h	2 h	
Wirtschaftlichkeitsanalyse	2 h	2 h	
Entwurfsphase	**13 h**	**13 h**	
Programmierschnittstelle	2 h	3 h	+ 1
Datenmodell	3 h	4 h	+ 1
Geschäftslogik	4 h	4 h	
Pflichtenheft	4 h	2 h	- 2
Implementierungsphase	**18 h**	**19 h**	
Programmierschnittstelle[1]	-	4 h	+ 4
Datenmodell	8 h	6 h	- 2
Geschäftslogik	10 h	9 h	- 1
Qualitätsmanagement	**13 h**	**12 h**	
Modultests	4 h	3 h	- 1
Manuelle Tests	5 h	6 h	+ 1
Fehlerbehebung	4 h	2 h	- 2
Abnahme	-	1 h	+ 1
Dokumentation	**22 h**	**22 h**	
Projektdokumentation	12 h	14 h	+ 2
Entwicklerdokumentation	5 h	3 h	- 2
Benutzerdokumentation	5 h	5 h	
Gesamt	***70 h***	***70 h***	

Tabelle 4: Soll / Ist Vergleich

Obwohl es kleinere zeitliche Differenzen gab, wurde das Projekt wie geplant in insgesamt 70 Stunden fertiggestellt.

[1] Da die Implementierung der Programmierschnittstelle in der frühen Planung nicht extra gefasst wurde, sondern den Implementierungsphasen Datenmodell und Geschäftslogik zugeordnet war, fehlt die Zeitplanung für diesen Punkt.

A21 CD

****

****

****

Tel: **********

**********.com