­

PAC Conferences



PAC Conferences

Service Architecture  
28.10.2013

Nicolas Moser



# Projektinformationen

|  |  |
| --- | --- |
| **Kurzbezeichnung:** | PAC Conferences – Service Architecture |
|  |  |
| **Version:** | 1.0 |
| **Stand:** | 28.10.2013 |
| **Autor(en):** | Nicolas Moser |
| **E-Mail:** | [nicolas.moser@prodyna.com](mailto:nicolas.moser@prodyna.com) |
|  |  |

# Änderungshistorie

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Datum | Autor | Änderung | Kapitel |
| 1.0 | 01.11.2013 | Nicolas Moser | Initiale Version | Alle |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

# Änderungen dieses Release

|  |  |  |
| --- | --- | --- |
| Kapitel | Abschnitt | Zusammenfassung der Änderungen |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Inhaltsverzeichnis

[1. Projektinformationen 2](#_Toc370976089)

[2. Änderungshistorie 2](#_Toc370976090)

[3. Änderungen dieses Release 2](#_Toc370976091)

[4. Inhaltsverzeichnis 3](#_Toc370976092)

[5. Abbildungsverzeichnis 4](#_Toc370976093)

[6. Tabellenverzeichnis 4](#_Toc370976094)

[7. Einleitung 5](#_Toc370976095)

[8. Architektur 6](#_Toc370976096)

[8.1 Datenmodell 6](#_Toc370976097)

[8.2 Komponenten 7](#_Toc370976098)

[8.2.1 JSF Frontend 7](#_Toc370976099)

[8.2.2 REST Interface 8](#_Toc370976100)

[8.2.3 EJB Backend 10](#_Toc370976101)

[8.3 Deployment 11](#_Toc370976102)

[9. Projektstruktur 13](#_Toc370976103)

[9.1 Repository Struktur 13](#_Toc370976104)

[9.2 Teilprojekt Struktur 14](#_Toc370976105)

[10. Coding Standards 15](#_Toc370976106)

[10.1 Naming Standards 15](#_Toc370976107)

[10.1.1 Common 15](#_Toc370976108)

[10.1.2 JSF 15](#_Toc370976109)

[10.1.3 REST 15](#_Toc370976110)

[10.1.4 EJB 16](#_Toc370976111)

[11. Verwendete Standards und Technologien 17](#_Toc370976112)

[11.1 Standards 17](#_Toc370976113)

[11.2 Laufzeitumgebung 17](#_Toc370976114)

[11.3 Bibliotheken 17](#_Toc370976115)

[12. Entwicklungstools 18](#_Toc370976116)

[12.1 Versions Kontrolle 18](#_Toc370976117)

[12.2 Build 18](#_Toc370976118)

[12.3 Test 19](#_Toc370976119)

[12.4 Continuous Integration 19](#_Toc370976120)

[12.5 Datenbank 19](#_Toc370976121)

[13. Abkürzungsverzeichnis 20](#_Toc370976122)

# Abbildungsverzeichnis

[Figure 8.1: Datenmodell 6](#_Toc370975960)

[Figure 8.2: Komponenten 7](#_Toc370975961)

[Figure 8.3: JSF Managed Beans 8](#_Toc370975962)

[Figure 8.4: Conference REST Resources 9](#_Toc370975963)

[Figure 8.5: EJB Services 10](#_Toc370975964)

[Figure 8.6: Deployment 11](#_Toc370975965)

[Figure 9.1: Project Structure 13](#_Toc370975966)

[Figure 9.2: Teilprojektstruktur 14](#_Toc370975967)

# Tabellenverzeichnis

[Table 8.1: REST Ressourcen 9](#_Toc370975968)

[Table 8.2: Deployment Specifications 12](#_Toc370975969)

[Table 12.1: Build Files 18](#_Toc370975970)

[Table 12.2: Gradle Tasks 18](#_Toc370975971)

[Table 12.3: DDL Skripte 19](#_Toc370975972)

[Table 13.1: Abkürzungen 20](#_Toc370975973)

# Einleitung

Es soll eine Java Enterprise Applikation entwickelt werden, welche von Konferenzgästen genutzt werden kann um Informationen über die jeweilige Konferenz bzw. deren Reden und Präsentanten zu erhalten.

Zudem muss es eine Administrationsoberfläche für Redakteure umgesetzt werden um Konferenzen, Räume, Sprecher und Reden zu pflegen (anlegen, bearbeiten, löschen).

Um das System an weitere Systeme anzuschließen muss es eine REST Schnittstelle zur Verfügung stellen.

Diese und weitere Anforderungen an die Anwendung sind in <https://github.com/pd-nmoser/Conference/blob/master/com.prodyna.pac.conference/docs/Requirements.md> beschrieben.

# Architektur

Dieses Kapitel beschreibt die Architektur der Anwendung. Dabei werden die Verschiedenen Sichten auf die Anwendung hervorgehoben.

## Datenmodell

Das folgende Klassendiagramm beschreibt den Aufbau des Datenmodells der Anwendung. Hierbei ist bereits auf den Datenbankbezug Rücksicht genommen worden und u.a. M:N Relationen aufgelöst worden.

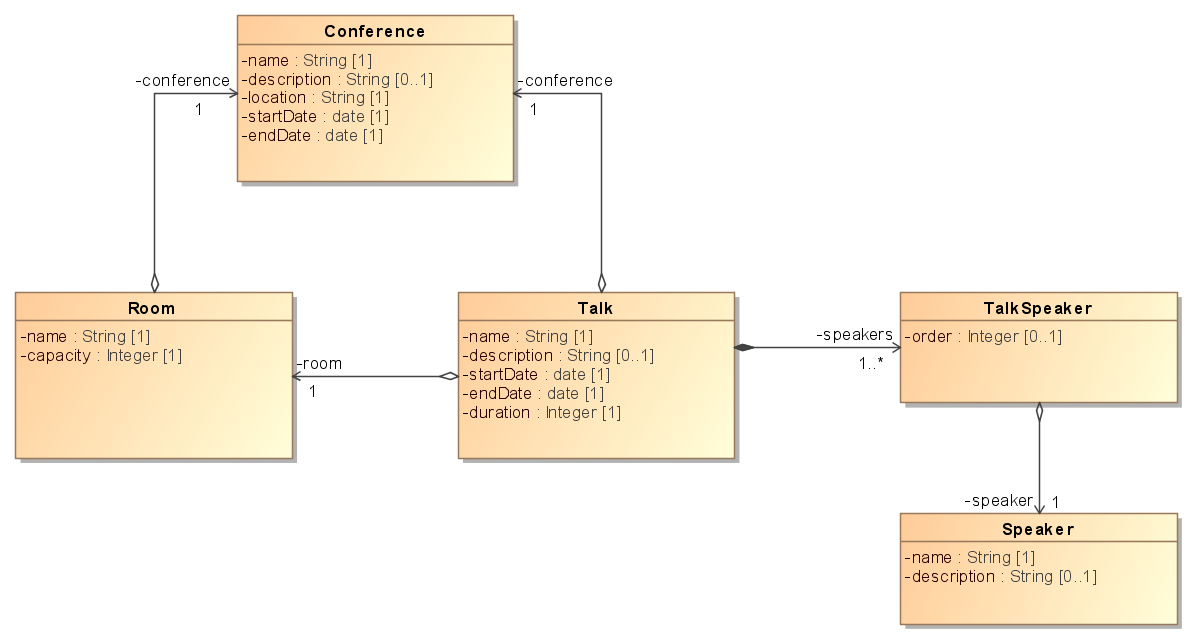


Figure 8.1: Datenmodell

Das Datenmodell ist in die folgenden fünf Entitäten aufgeteilt:

* **Conference:** Eine öffentliche Veranstaltung mit Namen, Beschreibung, Veranstaltungsort, Start-, und End-Datum
* **Room:** Ein Raum innerhalb einer Konferenz mit Namen und Anzahl Sitzplätzen
* **Talk:** Eine Rede innerhalb eines bestimmten Raumes einer Konferenz mit Name, Beschreibung, Dauer, Start- und End-Datum
* **TalkSpeaker:** Auflösung der M:N Relation zwischen Talk und Speaker
* **Speaker:** Ein Redner mit Name und Beschreibung

Die Entität TalkSpeaker wurde eingeführt um die M:N Relation zwischen Talk und Speaker aufzulösen. Ein Talk kann zugleich von mehreren Rednern gehalten werden, aber ein Redner kann ebenso mehrere Talks halten. Die TalkSpeaker Entität kann zusätzlich mit Attributen erweitert werden um die Relation zu bewerten (z.B. Reihenfolge, Priorität).

## Komponenten

Die Bausteinsicht beschreibt die statische Zerlegung des Systems in Bausteine (Module, Komponenten, Schichten etc.) sowie deren Beziehungen.

Die Anwendung ist in drei Grundbausteine geteilt:

* JSF Frontend
* REST Interface
* EJB Backend

Die JSF und REST Komponenten stellen HTTP Schnittstellen zur Verfügung.

Die EJB Komponente kommuniziert mit der Datenbank unter Verwendung von JPA und JDBC. Dabei wird das Datenmodell durch den Einsatz von Objekt-Relationalem Mapping in relationale Datenbank Tabellen überführt bzw. Einträge der Datenbank Tabellen in Java Objekte konvertiert.

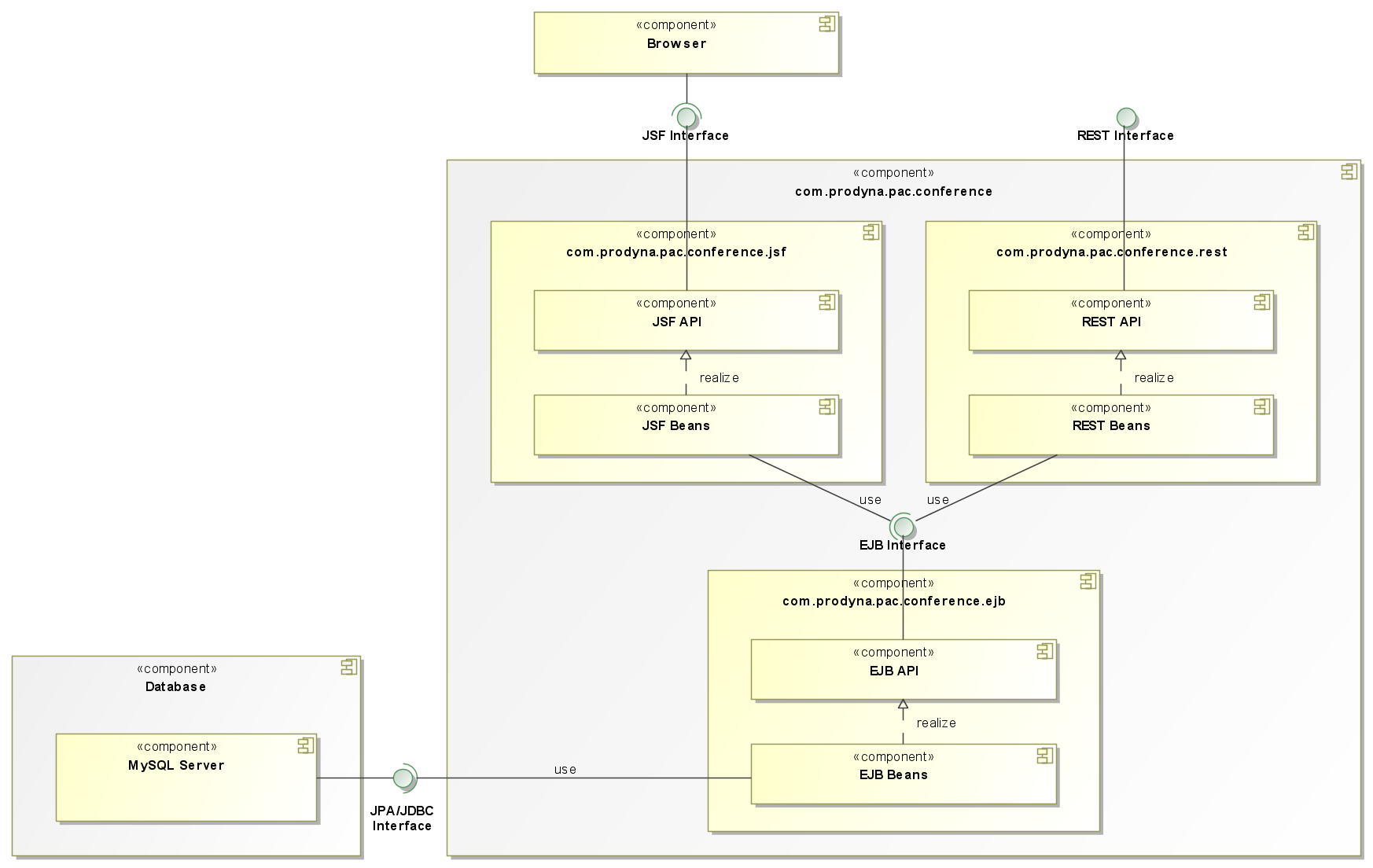


Figure 8.2: Komponenten

### JSF Frontend

Das Web-Frontend der Anwendung basiert auf JSF 2.0 (Java Server Faces), dem Java Standard für webbasierte Java Anwendungen. JSF setzt auf den Standards für Java Servlets und Java Server Pages auf und erweitert diesen durch die Verwendung einer Expression Language und Navigation Rules.

Im folgenden Diagramm wird beispielhaft anhand der Konferenzbearbeitung dargestellt welche Managed Beans benötigt werden um Konferenzen anzuzeigen und zu administrieren. Folgende Beans werden dabei implementiert:

* Zur Anzeige von Datentyp Listen wird eine ListBean verwendet
* Zum Laden eines einzelnen Datentyps wird eine Bean mit zugehörigem IdConverter verwendet
* Zum Administrieren wird eine AdminBean verwendet

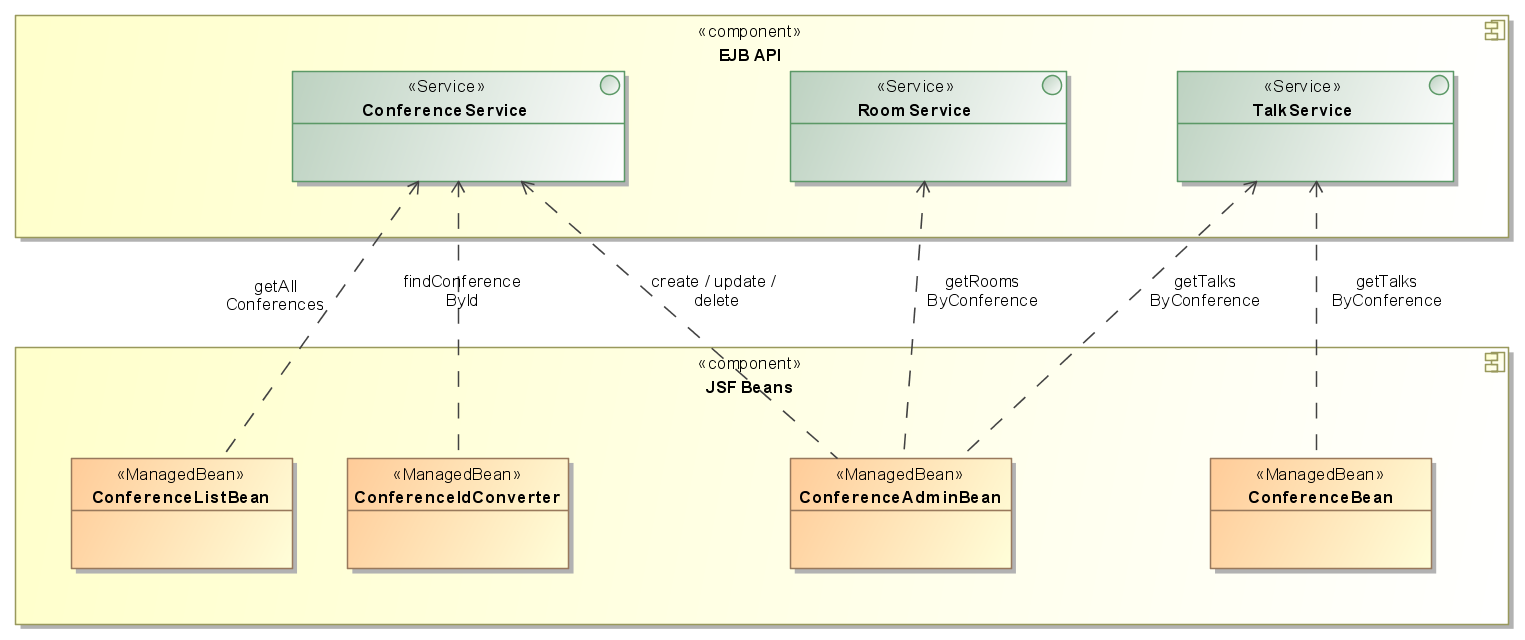


Figure 8.3: JSF Managed Beans

### REST Interface

Das REST Interface basiert auf dem Java Standard JAX-RS 1.1 (Java API for RESTful Web Services) und der Implementierung RestEasy von JBoss (siehe Java Community Prozess JAX-RS Spezifikation: <http://www.jcp.org/en/jsr/detail?id=339>).

Ein REST Service wird mittels einer konkreten URL angesprochen. Dabei wird durch die JAX-RS Implementierung die jeweilige Ressource identifiziert und aufgerufen. Je nach HTTP Request Type (GET, PUT, POST, DELETE, etc.) wird die spezifische Operation der Ressource ausgeführt. Je nach Austauschformat, findet ein Mapping zwischen Java Objekten und dem Zielformat (z.B. JSON) statt.

Die folgende Abbildung stellt die Architektur der REST Ressourcen (beispielhaft an der ConferenceResoure) dar. Jede Ressource kommt in zwei Ausprägungen vor, eine Ressource mit Lese-Operationen und eine Ressource mit Lese- und Schreib-Operationen. Dadurch lassen sich Lese- und Schreib-Operationen sicherheitstechnisch trennen. Die Ressource mit Schreib-Operationen erhält das Präfix Admin um dessen administrative Fähigkeit zu unterstreichen.

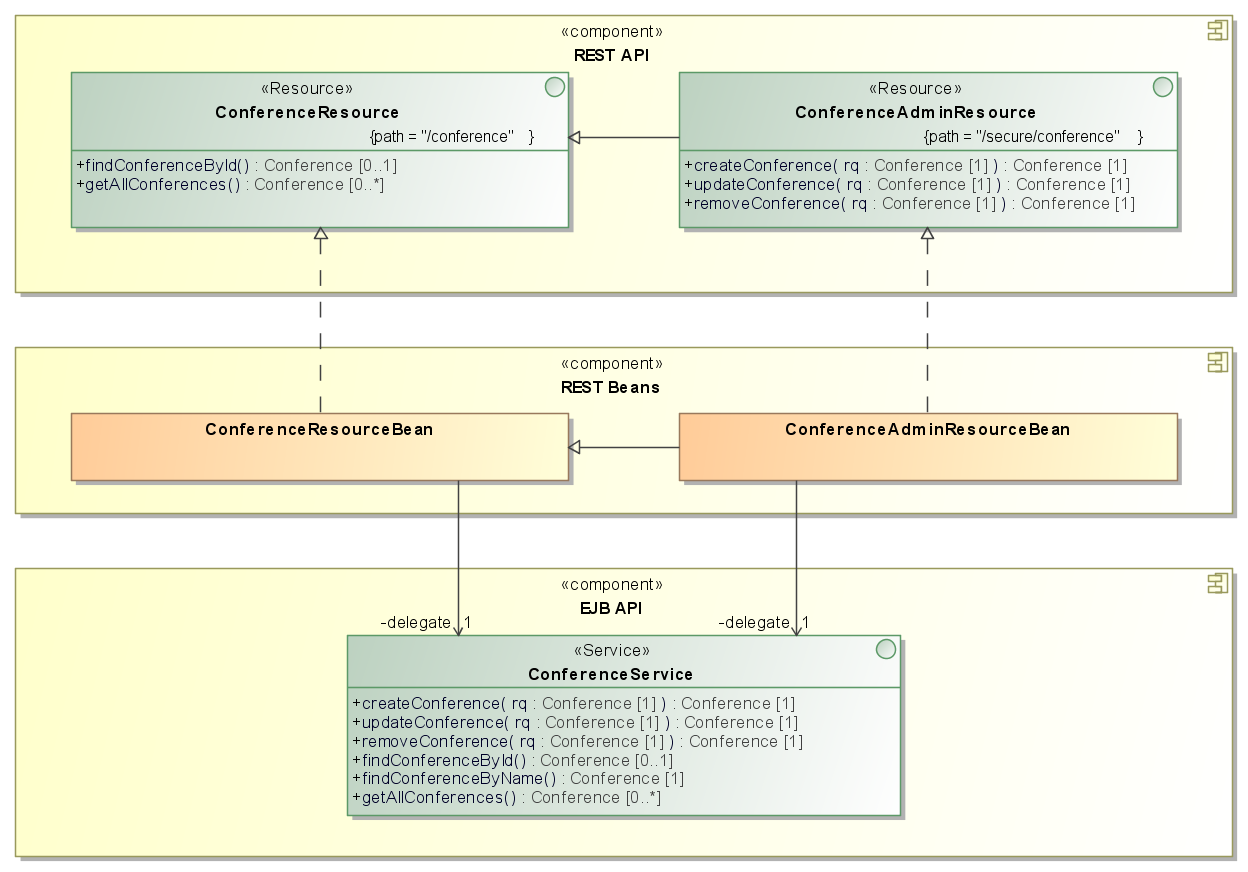


Figure 8.4: Conference REST Resources

Eine Ausprägung teilt sich auf in Interface und Implementierung. Die administrative Ausprägung erweitert dabei stets die einfache Ausprägung um ebenfalls Zugriff auf Lese-Operationen zur Verfügung zu stellen. Beide Ressource Ausprägungen delegieren die REST Operationen an die jeweilige Service Operation der EJB Komponente.

Die Benutzung von RESTEasy ist unter <http://www.jboss.org/resteasy> beschrieben.

Die folgenden Ressourcen werden von der Anwendung zur Verfügung gestellt und können unter den relatien URLs aufgerufen werden:

|  |  |  |
| --- | --- | --- |
| Ressource | Parameter | URL |
| ConferenceResource | GET | /rest/conference |
| RoomResource | GET | /rest/room |
| SpeakerResource | GET | /rest/speaker |
| TalkResource | GET | /rest/talk |
| ConferenceAdminResource | GET / PUT / POST / DELETE | /rest/secure/conference |
| RoomAdminResource | GET / PUT / POST / DELETE | /rest/secure/room |
| SpeakerAdminResource | GET / PUT / POST / DELETE | /rest/secure/speaker |
| TalkAdminResource | GET / PUT / POST / DELETE | /rest/secure/talk |

Table 8.1: REST Ressourcen

### EJB Backend

Das EJB Backend beschreibt Services zur Ausführung von Business Logik oder Persistenz von Datentypen. Diese Services basieren auf Operationen zum Suchen und Bearbeiten der Datentypen bzw. Entitäten. Die folgende Abbildung zeigt die Services und Implementierungen des EJB Backends inklusive deren Operationen.

Die Implementierungen basieren auf EJB (Enterprise Java Beans) 3.1 Session Beans. Jede SessionBean Instanz wird vom EJB Container verwaltet und kann von diesem zur Verfügung gestellt werden. Die Annotationen an den Service Interfaces beschreiben ob ein Service Lokal oder Remote verfügbar sein soll. Im Falle der Conference Application werden ausschließlich Lokale SessionBeans verwendet, da diese nur von Komponenten innerhalb des gleichen Application Containers referenziert werden.

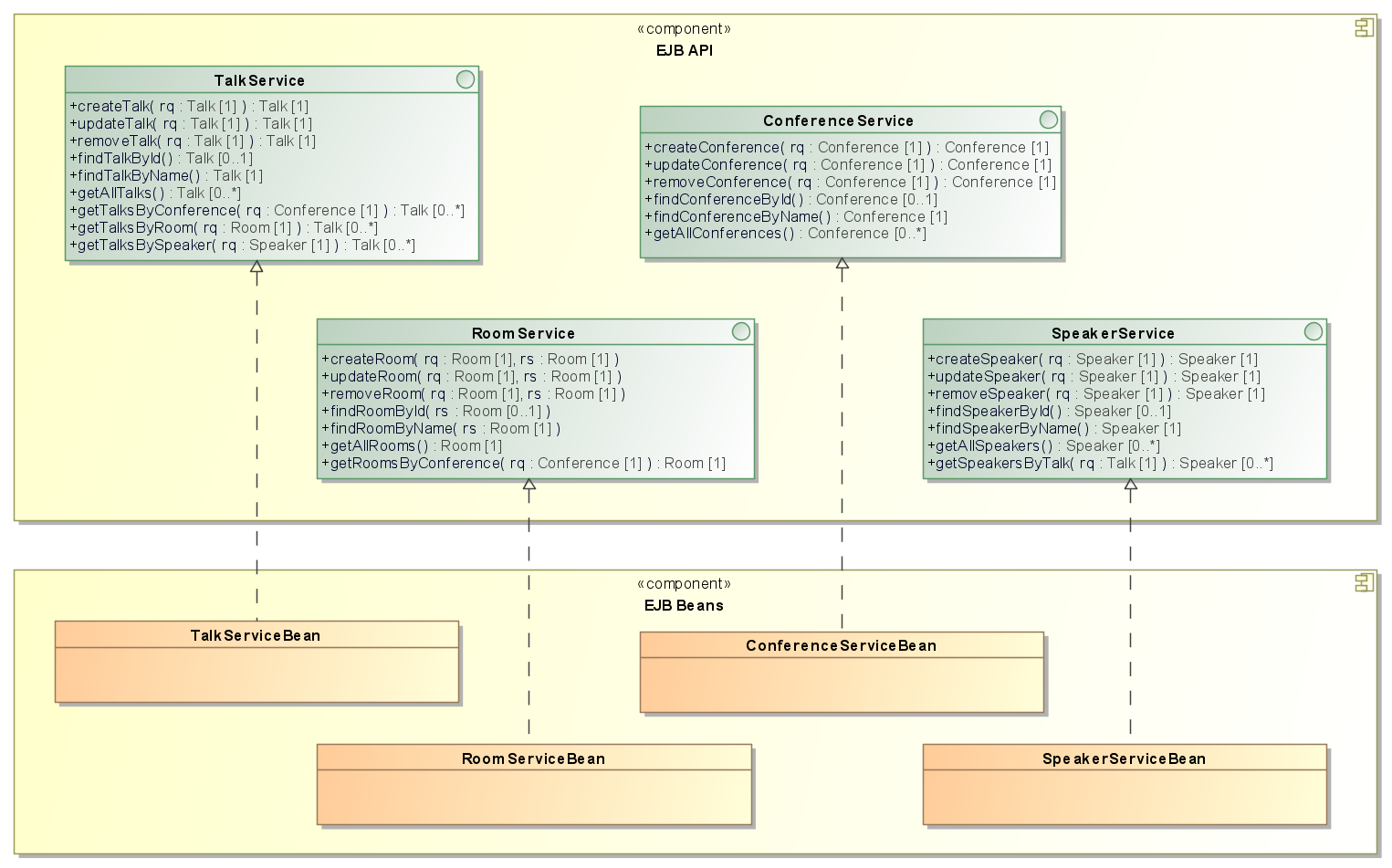
Zudem sind SessionBeans in der Lage auf die Persistence Unit der Anwendung und somit die Datenbank zuzugreifen. Operationen von SessionBeans sind zudem transaktionsgesichert.

Figure 8.5: EJB Services

Die Persistenz kapselt die Schnittstelle zu den Operationen der Datenbank. Realisiert wird die Persistenz mittels der „Java Persistence API (JPA) 2.0“ (siehe Java Community Process JPA Spezifikation: <http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html>).

JPA setzt auf das objektrelationale Mapping. Dabei werden Klassen zu Tabellen und Instanzvariablen zu Tabellenspalten des jeweiligen Datenbankschemas konvertiert. Das Mapping erfolgt per Annotations an den jeweiligen Entitäten.

JPA stellt lediglich die Spezifikation des Java Persistenz-Standards dar, gibt jedoch keine konkrete Implementierung vor. Es wurde die Implementierung Hibernate gewählt, da sie als Standard JPA-Implementierung des JBoss AS mitgeliefert wird und dadurch eine optimal getestete Integration in die JBoss Suite liefert.

Die Interaktion mit Entitäten (Erstellen, Laden, Bearbeiten, Löschen, etc.) erfolgt in JPA mittels einer Instanz des EntityManagers. Der EntityManager kapselt die JDBC-Datenbank Interaktion.

Zur Abfrage von Datenbank Queries wird von JPA eine Object-Query-Language (OQL bzw. JPQL) eingesetzt. Diese Abfragesprache ähnelt der Structured Query Language (SQL), basiert jedoch nicht auf Tabellen sondern Objekten. Queries stellen eigene Objekte dar und können vom EntityManager erzeugt werden.

## Deployment

Im Folgenden wird der Aufbau des Deployment Artefaktes der Anwendung beschrieben. Dieses setzt sich aus einem Java Enterprise Archive (EAR) zusammen. Wie in der folgenden Abbildung sichtbar teilt sich das Archiv in einzelne Teilartefakte auf.

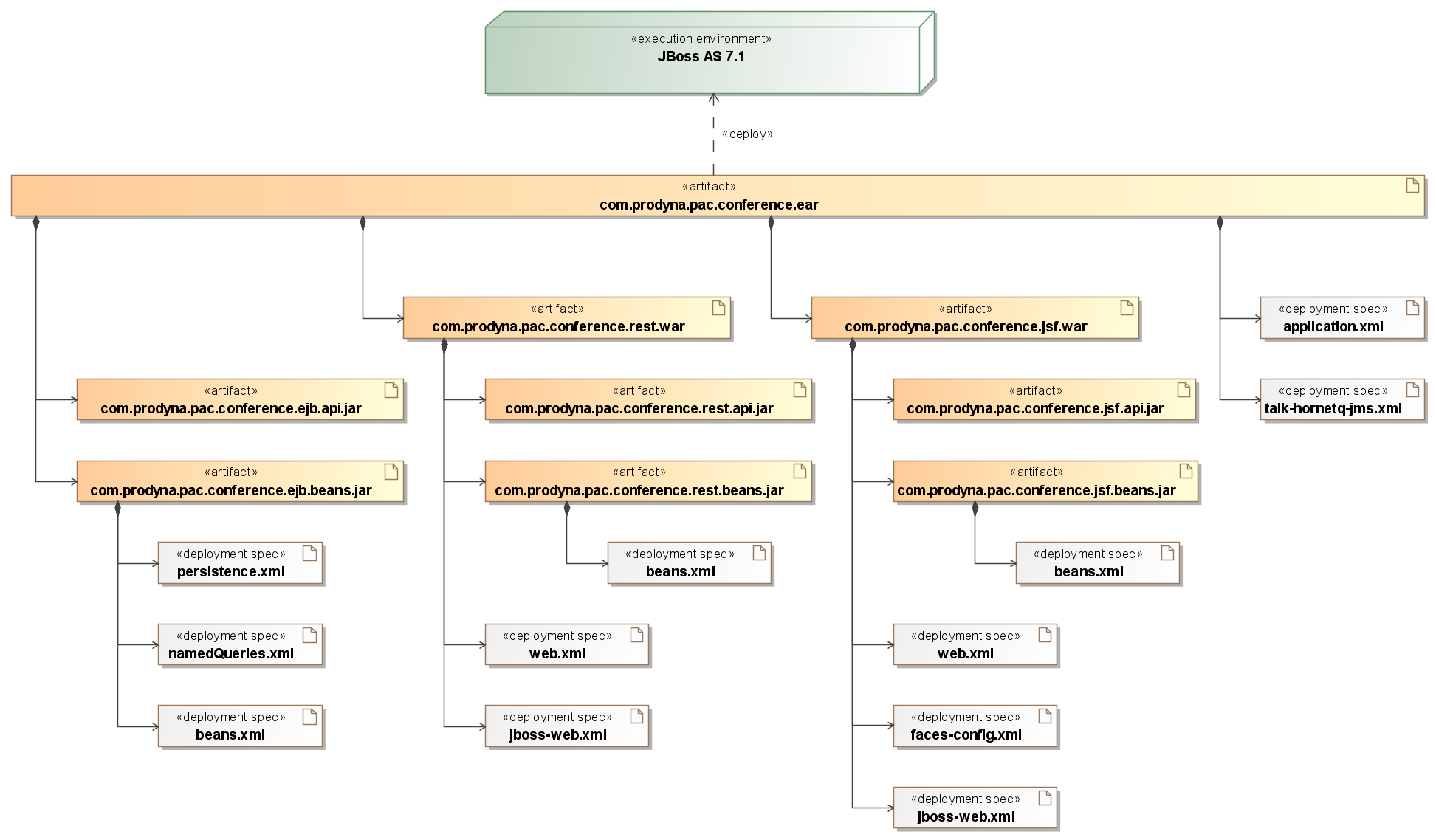


Figure 8.6: Deployment

* com.prodyna.pac.conference.ear
  + com.prodyna.pac.conference.ejb.api.jar
  + com.prodyna.pac.conference.ejb.beans.jar
  + com.prodyna.pac.conference.jsf.war
    - com.prodyna.pac.conference.jsf.api.jar
    - com.prodyna.pac.conference.jsf.beans.jar
  + com.prodyna.pac.conference.rest.war
    - com.prodyna.pac.conference.rest.api.jar
    - com.prodyna.pac.conference.rest.beans.jar

Zudem gibt es diverse Deployment Spezifikationen zu den einzelnen Artefakten. In der folgenden Tabelle werden diese Spezifikationen detailliert beschrieben.

|  |  |  |
| --- | --- | --- |
| Deployment Spezifikation | Artefakt | Beschreibung |
| application.xml | EAR | Definiert die Java EE Applikation und beinhaltete Module |
| talk-hornetq-jms.xml | EAR | Definiert eine JMS Queue, welche im JBoss bereitgestellt wird |
| persistence.xml | EJB JAR | Beschreibt die JPA Persistence Unit mit allen zugehörigen Entitäten und Queries |
| namedQueries.xml | EJB JAR | Beschreibt sämtliche JPQL Named Queries der JPA Persistence Unit |
| beans.xml | Alle JARs | Beschreibt CDI Konfiguration (Decorators, Interceptors, Alternatives, etc.) |
| web.xml | Alle WARs | Beschreibt die Web Applikation mit Standard Java EE Parametern |
| Jboss-web.xml | Alle WARs | Beschreibt die Web Applikation mit JBoss spezifischen Konfigurationparametern |
| faces-config.xml | JSF WAR | Beschreibt JSF Applikationsparameter (u.A. Navigationsregeln) |

Table 8.2: Deployment Specifications

# Projektstruktur

Im Folgenden wird die Struktur des Projektes beschrieben. Dabei wird zum einen die Struktur des gesamten Repositories beschrieben als auch auf die Struktur der einzelnen Teilprojekte eingegangen.

## Repository Struktur

Das Projekt befindet sich innerhalb eines Parent Projektes und unterteilt sich darunter in mehrere hierarchisch geschachtelte Teilprojekte. Die Projektstruktur ist in folgendem Diagramm beschrieben.

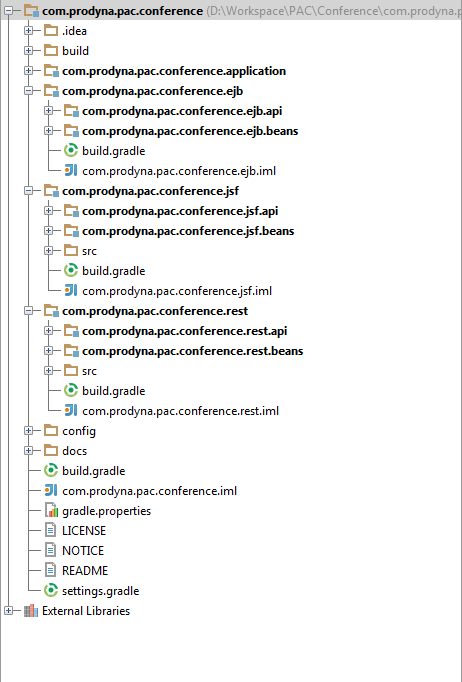


Figure 9.1: Project Structure

Unter dem Root-Projekt *com.prodyna.pac.conference* folgen die Projekte *com.prodyna.pac.conference.ejb* für das EJB Backend, *com.prodyna.pac.conference.jsf* für das JSF Frontend und *com.prodyna.pac.conference.rest* für die REST Schnittstelle. Jedes Teilprojekt unterteilt sich selbst wieder in ein API (Schnittstellen) und ein Bean (Implementierungen) Projekt.

## Teilprojekt Struktur

Jedes Teilprojekt der Anwendung besteht aus vier Source Foldern:

* **src/main/java** für Java Klassen der Anwendung
* **src/main/resources** für Resourcen der Anwendung
* **src/test/java** für Java Klassen zu Testzwecken
* **src/test/resources** für Resourcen zu Testzwecken

Die folgende Abbildung zeigt das die Aufteilung des Projektes *com.prodyna.pac.conference.ejb.beans* in die vier Source Folder.

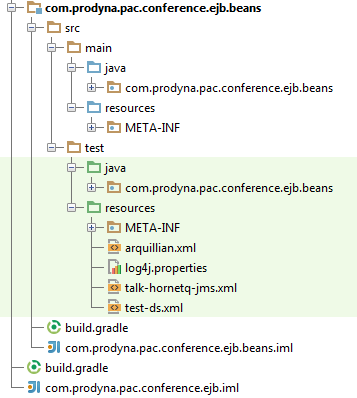


Figure 9.2: Teilprojektstruktur

# Coding Standards

Im Folgenden werden Prozesse und Standards definiert, welche zur Entwicklung der Anwendung eingehalten werden müssen.

Die Coding Standards basieren auf den Oracle Java Coding Conventions definiert in <http://www.oracle.com/technetwork/java/codeconv-138413.html>.

## Naming Standards

Die folgenden Package- und Klassen-Bezeichnungen werden angewandt.

### Common

Folgende Namenskonventionen sind für Klassen aller Komponenten gültig:

* CDI Decorators erhalten das Suffix Decorator
* CDI Events erhalten das Suffix Event
* CDI Interceptors erhalten das Suffix Interceptor
* CDI Producers erhalten das Suffix Producer
* JMX MBeans erhalten das Suffix MBean

### JSF

Folgende Namenskonventionen sind für Packages der JSF Komponente gültig:

* Alle Bestandteile der REST Schnittstelle werden innerhalb des Packages *com.prodyna.pac.conference.jsf.api* abgelegt.
* Alle Bestandteile der REST Implementierung werden innerhalb des Packages *com.prodyna.pac.conference.jsf.beans* abgelegt.
* Managed Beans mit administrativen Operationen werden innerhalb des Sub-Packages *admin* abgelegt.

Folgende Namenskonventionen sind für Klassen der JSF Komponenten gültig:

* Managed Beans erhalten das Suffix Bean.
* Managed Beans zur Verwaltung von Listen erhalten das Suffix ListBean
* Managed Beans zur Administration von Entitäten erhalten das Suffix AdminBean
* JSF Converter erhalten das Suffix Converter

### REST

Folgende Namenskonventionen sind für Packages der REST Komponente gültig:

* Alle Bestandteile der REST Schnittstelle werden innerhalb des Packages *com.prodyna.pac.conference.rest.api* abgelegt.
* Alle Bestandteile der REST Implementierung werden innerhalb des Packages *com.prodyna.pac.conference.rest.beans* abgelegt.
* Ressourcen mit administrativen Operationen werden innerhalb des Sub-Packages *admin* abgelegt.

Folgende Namenskonventionen sind für Klassen der REST Komponenten gültig:

* REST Interfaces erhalten das Suffix Resource
* REST Interfaces zur Administration erhalten das Suffix AdminResource
* REST Implementierungen erhalten das Suffix ResourceBean
* REST Implementierungen zur Administration erhalten das Suffix AdminResourceBean

### EJB

Folgende Namenskonventionen sind für Packages der EJB Komponente gültig:

* Alle Bestandteile der EJB Schnittstelle werden innerhalb des Packages *com.prodyna.pac.conference.ejb.api* abgelegt.
* Alle Bestandteile der EJB Implementierung werden innerhalb des Packages *com.prodyna.pac.conference.ejb.beans* abgelegt.
* Services und Service Interfaces werden innerhalb des Sub-Packages *service* abgelegt.

Folgende Namenskonventionen sind für Klassen der REST Komponenten gültig:

* Service Implementierungen erhalten das Suffix Bean
* MessageDrivenBeans erhalten das Suffix NotificationBean

# Verwendete Standards und Technologien

Im Folgenden werden die verwendeten Technologien und Bibliotheken beschrieben, die zur Entwicklung und Laufzeit der Anwendung benötigt werden.

## Standards

Folgende (Java) Standards wurden zur Entwicklung der Anwendung benötigt:

|  |  |
| --- | --- |
| Name | Version |
| JSF (Java Server Faces) | 2.2 |
| Java Servlets | 3.0 |
| JAX-RS (Java API for RESTful Web Services) | 1.1 |
| CDI (Contexts and Dependency Injection) | 1.0 |
| EJB (Enterprise Java Beans) | 3.1 |
| JPA (Java Persistence API) | 2.0 |
| JMS (Java Message Service) | 1.1 |
| JMX (Java Management Extensions) | 2.0 |

## Laufzeitumgebung

Folgende Technologien werden zur Ausführung der Anwendung benötigt:

|  |  |
| --- | --- |
| Name | Version |
| JBoss AS | 7.1.1.Final |
| MySQL Server | 5.1.49 |
| MySQL JDBC Driver | 5.1.26 |

## Bibliotheken

Zur Entwicklung der Anwendung wurden folgende Bibliotheken eingesetzt:

|  |  |  |
| --- | --- | --- |
| Name | Version | Beschreibung |
| JBoss EJB | 1.0.1.Final | EJB Specification and Interface |
| JBoss CDI | 1.0-SP4 | CDI Framework |
| Hibernate | 1.0.1.Final | JPA Framework |
| Hibernate Validator | 4.2.0.Final | Validations Framework |
| RestEasy | 2.3.2.Final | JAX-RS Framework |
| JSF | 2.0.1.Final | JSF Framework |
| JSF Richfaces | 4.3.4.Final | JSF Framework |
| SLF4J | 1.7.5 | Logging Framework |
| JUnit | 4.11 | Test Framework |
| Hamcrest | 1.2 | Test Framework |
| Arquillian | 1.1.1.Final | Test Framework |
| Shrinkwrap | 2.0.0-alpha-3 | Test Framework |
| Mockito | 1.9.5 | Test Framework |
| Jacoco | 0.6.0.201210061924 | Test Reporting Framework |

# Entwicklungstools

Im Folgenden werden die verwendeten Entwicklungswerkzeuge zur Umsetzung der Conference Anwendung beschrieben.

## Versions Kontrolle

Zur historisierten Ablage von Quellen wird das Versionsverwaltungstool git eingesetzt (siehe <http://git-scm.com/>).

Das Repository wird durch den Provider GitHub (siehe <https://github.com/>) verwaltet und kann unter der folgenden URL abgerufen werden:

* <https://github.com/pd-nmoser/Conference/>

## Build

Zur automatisierten Erstellung von Release Artefakten wird das Build Werkzeug Gradle eingesetzt (siehe <http://www.gradle.org/>). Dieses ermöglicht die automatisierte

|  |  |
| --- | --- |
| Dateiname | Beschreibung |
| build.gradle | Definiert den Ablauf des Build Prozesses. Jedes Gradle Projekt bzw. Teilprojekt erhält eine build.gradle Datei mit projektspezifischen Settings. |
| settings.gradle | Beschreibt die Gradle Projektstruktur bzw. alle beteiligten Gradle Projekte ausgehend vom Root Projekt. |
| arquillian.gradle | Definiert sämtliche benötigten Dependencies und Tasks zur ausführung von Arquillian Tests innerhalb des Builds. |
| database.gradle | Definiert Tasks zur Ausführung von ausgewählten DDL Skripten um Tabellen der darunter liegenden Datenbank zu modifizieren. |
| java.gradle | Definiert sämtliche Build Tasks und Einstellungen in Bezug auf Java Kompilation und Ausführung. |
| quality.gradle | Definiert Plugins und Tasks zur Ausführung von Code-Qualitäts-Checks (Findbugs, Checkstyle, PMD, etc.) während des Buildprozesses. |
| test.gradle | Definiert Tasks und Einstellungen in Bezug auf Ausführung von Java Unit Tests. |

Table 12.1: Build Files

Die folgende Tabelle beschreibt die wichtigsten Gradle Build Tasks.

|  |  |
| --- | --- |
| Task | Beschreibung |
| compile | Kompiliert alle Java Dateien des jeweilligen Projektes. |
| jar | Erstellt ein Java Archiv (Jar) mit den kompilierten Klassen des Projektes. |
| war | Erstellt ein Web Archiv (War) mit Klassen, Libraries, Content des Projektes. |
| ear | Erstellt ein Enterprise Archiv (Ear) aus allen Modulen. |
| test | Führt sämtliche Unit Tests des jeweiligen Projektes aus. |
| build | Führt den gesamten Build (inkl. Test) des jeweiligen Projektes aus. |

Table 12.2: Gradle Tasks

Die Ausgabe des Builds (Reports, Artefakte, temporäre Dateien, etc.) erfolgt pro Projekt in dem Verzeichnis *build/* innerhalb des entsprechenden Gradle Projekts.

## Test

Zur Ausführung der Unit Tests wird das Framework JUnit eingesetzt. Zudem wird das Test Framework Arquillian zum Testen gegen den JBoss AS verwendet.

Durch den Build ausgeführte Tests erzeugen einen HTML Testreport, welcher innerhalb des *build/reports/test* Verzeichnis abgelegt wird.

Zudem können durch das Tool Jacoco Test Coverage Reports erstellt werden. Diese werden im Verzeichnis *build/reports/jacoco* abgelegt.

## Continuous Integration

Zur kontinuierlichen Ausführung (Continuous Integration bzw. CI) des Java Kompilers und Projekttests wird das Tool Travis-CI eingesetzt. Dieses ermöglicht die frühzeitige Identifikation von Fehlern und erhöht somit die Code Qualität.

Travis-CI ist ein Software-As-A-Service (SAAS) Lösung und wird durch ein Plugin mit der Sourceverwaltung von Github verknüpft um somit auf die Source- bzw. Build-Dateien zugreifen zu können.

Die CI der Conference Application kann unter folgender URL eingesehen werden:

* <https://travis-ci.org/pd-nmoser/Conference>

Konfiguriert wird die CI mittels der Konfigurationsdatei travis.yml innerhalb des Root Projektes. Diese Datei definiert Konfigurationsparameter (Java Version, etc.), sowie den CI Lifecyclye (Shell Scripte, Datenbank Setup, etc.) und führt letztendlich den Buildprozess aus.

## Datenbank

Die MySQL Datenbank kann mittels der bereitgestellten DDL initialisiert werden. Dafür muss zuerst ein neues Datenbankschema mit dem Namen „*conference“* angelegt werden.

Anschließend können, je nach Absicht, die DDL Scripte ausgeführt werden. Folgende DDL Scripte stehen zur Verfügung:

|  |  |
| --- | --- |
| DDL Script | Beschreibung |
| create-tables.sql | Erstellt alle relevanten Tabellen. |
| create-tables-testdata.sql | Erstellt alle relevanten Tabellen und fügt Testdaten ein. |
| drop-create-tables.sql | Löscht alle Tabellen und erstellt diese neu. |
| drop-create-tables-testdata.sql | Löscht alle Tabellen und erstellt diese neu inkl. Testdaten. |
| Insert-testdata.sql | Fügt Testdaten in ein Schema mit leeren Tabellen ein. |

Table 12.3: DDL Skripte

# Abkürzungsverzeichnis

Im Folgenden werden die verwendeten Abkürzungen, Akronyme und Begriffe des Projekts aufgelistet und beschrieben.

|  |  |
| --- | --- |
| Abkürzung | Name |
| API | Application Programming Interface |
| AS | Application Server |
| CDI | Contexts and Dependency Injection |
| DDL | Data Definition Language |
| EAR | Enterprise Archive |
| EJB | Enterprise Java Beans |
| HTML | Hypertext Markup Language |
| HTTP | Hypertext Transfer Protocol |
| ID | Identifier |
| JAR | Java Archive |
| JAX | Java API for XML |
| JAX-RS | Java API for RESTful Web Services |
| JCP | Java Community Process |
| JMS | Java Message Service |
| JMX | Java Management Extensions |
| JPA | Java Persistence API |
| JSF | Java Server Faces |
| JSON | JavaScript Object Notation |
| JSP | Java Server Pages |
| PAC | PRODYNA Architecture Certification |
| REST | Representational State Transfer |
| SCM | Source Control Management |
| SFSB | Stateful Session Bean |
| SLF4J | Simple Logging Facade for Java |
| SQL | Structured Query Language |
| WAR | Web Archive |
| XHTML | Extensible HyperText Markup Language |
| XML | Extensible Markup Language |

Table 13.1: Abkürzungen