# Service Architecture Conference Application

# Technologie

- Java 1.7

- JBoss 7.1.1

- JSF 2.1, Primefaces 4.0

- JPA 2.0, Hibernate 4.0.1

- CDI API 1.0

- SLF4J 1.6.1

- MySQL v5.1.x

- JAX-RS 2.0, Jackson 1.9.2, RestEasy 2.3.2

- Maven 3.0

# Testing

- Arquillian 1.0.3. Final

- JUnit 4.10

# Konfiguration Laufzeit

Die benötigte Konfiguration zum Entwicklungszeitpunkt wird im GIT unter conference/conference-ear/conf/jboss verwaltet. Es handelt sich hierbei um eine Erweiterung des JBoss Standalone Profiles um die entsprechenden Messagingmodule für JMS. Darüber hinaus wird eine MySQL-Datasource unter java:jboss/datasources/ConferenceDS benötigt. Die JMS-Queue ist unter dem JNDI-Namen„queue/talk“ konfiguriert.

# Aufbau der Eclipse-Projekte

Grundsätzlich handelt es sich bei den Eclipse-Projekten um Maven Multi Module Projekte.

## conference-ear

Wird zur Erstellung des EAR-Artifaktes verwendet.

#### Unterordner conf

Enthält alle wichtigen Konfigurationseinstellungen (z.B. JBOSS-Konfiguration). Zu einem späteren Zeitpunkt sind hier auch Checkstyle, Logging-Konfigurationen zu pflegen.

#### Unterordner doc

Enthält sämtliche technische Architekturdokumenation (z.B. das vorliegende Dokument).

## conference-ejb-client

Enthält die Schnittstellen-Interfaces und Exceptions für die Service-Schicht.

## conference-ejb

Enthält die Implementierung des „conference-ejb-client“ in Form von Stateless Session Beans. Unter „src/main/resources/META-INF“ ist die „persistence-xml“ zu finden, welche für die Gesamtapplikation die Datenbankverbindung zentral konfiguriert. An gleicher Stelle ist die DDL-Logik zur Erstellung der Datenbank zu finden.

#### Package com.prodyna.conference.mbean

Enthält die MX-Bean Implementierung um Performancekennzahlen zur Laufzeit im JMX-Provider von JBoss bereitzustellen.

#### Package com.prodyna.conference.producer

Enthält CDI-Producer zur Bereitstellung der Logger und EntityManager-Instanz, sowie eine Abstraktion zum Senden von JMS-Nachrichten.

#### Package com.prodyna.conference.interceptor

Enthält alle querschnittlichen Interceptoren, welche auf allen Stateless Session Beans genutzt werden sollten. Zurzeit ist nur der MonitoringInterceptor enthalten, welche die Performancekennzahlen sammelt und detaillierte Log-Ausgaben zur den Serviceaufrufen ausgiebt.

## conference-model

Enthält alle Entitäten, welche sowohl im Server, als auch im Client verwendet werden. Darüber hinaus können hier Base-Klassen bzw. Utility-Klassen abgelegt werden, welche im Server und Client gleichermaßen benutzt werden sollen.

TODO: Eventuell sollten die Entitäten besser in das „ejb-client“ verschoben werden und lediglich die Base-Klassen verbleiben.

## conference-rest-client

Stellt die Funktionalität der Service Schicht als REST-Schnittstelle nach Außen bereit. Die Service-Interfaces werden nicht direkt implementiert, um entsprechend die Flexibilität zu erhalten, einige Methoden nicht oder orchestriert als REST-Schnittstelle bereitstellen zu können.

## conference-web

Enthält die Primefaces JSF-Klassen und HTML-Templates zur Darstellung des Webclients.

# Standards

## Entwicklungsumgebung

Als Entwicklungsumgebung soll das vorgefertigte Eclipse Juno aus unserem Wiki (TODO Adresse) verwendet werden. Alle Java, Textdateien sollen in UTF-8 encodiert sein.

Codeformatter und Codetemplates sind im conf-Ordner zu finden (TODO müssen noch erstellt werden).

## Code

Codesprache und Codedokumentation soll in Englisch erstellt werden.

## Namenstandards

* Entitäten sollen fachlich ohne Pre- oder Postfix benannt werden
* Serviceschnittstellen „xxxService“
* Implementierung der Serviceschnittstellen „xxxServiceBean“
* CRUD-Services:
  + save[Entity] - Persist und Merge-Methode (Verzweigung über technische ID != null)
  + delete[Entity]
  + getAll[Entity]s
* Interceptoren, Decorator, Controller, Producer mit entsprechenden Postfixes benennen.
* Named-Queries sollen zu Beginn den Namen der führenden Entität enthalten (z.B.Conference.ALL)
* Die .xhtml-Datei und der zugeordnete Controller sollte identisch benannt werden.

# Design und Technologieentscheidungen

In den nachfolgenden Kapiteln sollen für wichtige Technologie- und Designentscheidungen die einzelnen Evaluationsschritte und die Begründung für die Entscheidung dokumentiert werden.

## Exception

Innerhalb der Serviceschicht können zwei Arten von Exceptions geworfen werden:

* Ableitungen von *ConferenceServiceException*: Diese sollen immer dann verwendet werden, wenn es sich bei dem Fehler explizit um eine fachliche Fehlerkonstellation handelt, auf welche gegebenenfalls reagiert werden sollte.
* RuntimeExceptions: Sämtliche technische Fehler werden generisch sowohl im JSF-Client (DefaultExceptionHandler), als auch im REST-Client (RESTDefaultExceptionHandler) gefangen und entsprechend (später internationalisiert) im Clientformat ausgegeben. Im JSF-Client werden dazu Facesmessages verwendet. Im REST-Client ist für die Fehlermeldung ein spezielles Response-Objekt vorgesehen.

Durch die vermehrte Verwendung von Runtimeexceptions und der generischen Behandlung kann der Entwickler weitesgehend auf try/catch-Blöcke in den unterschiedlichen Schichten verzichten. Dies erhöht die Entwicklungsgeschwindigkeit. Durch die generische Behandlung ist sichergestellt, dass keine Exceptions „verloren“ gehen.

## Validierung

….

## Evaluation des Webframework

…