# Service Architecture PAC Conference

# Verwendete Technologien

- Es werden entsprechend dem Kundenwunsch JBoss 7.1.1, JSF 2.1 und JPA (Hibernate) verwendet.

- Hibernate 4.0.1 (per JBoss)

- Arquillian 1.0.3Final

- Joda-time 1.1

- JUnit 4.10

- MySQL >= v5.1.26

- JAX-RS 2.0 (Jackson 1.9.2, RestEasy 2.3.2 per Jboss)

# Umgebung

Als Umgebung wird von einem JBoss version 7.1.1 ausgegangen.

## Datasources

Es wird eine mysql Datasource java:jboss/datasources/ConferenceDS benötigt

## Benutzer

Es wird ein Benutzer im ApplicationRealm mit der Rolle „conferenceBackoffice“ benötigt.

## Queues

Es wird eine QueueConnectionFactory mit dem Namen „ConnectionFactory“ benötigt (standardmässig in standalone-full enthalten).

Es wird eine Queue „queue/test“ benötigt (standardmässig in standalone-full enthalten).

## Testausführung

Die beim Build ausgeführten Arquillian Unittests erfordern einen gestarteten JBoss. Dieser muss einen als h2 benannten H2 DB Treiber bereitstellen (standardmässig in standalone-full enthalten)

## Startseite

Die Startseite kann mit <http://localhost:8080/conference-web> aufgerufen werden.

# Aufbau der Quellen im Repository

Das Repository "PAC" gliedert sich in drei Unterordner:

## conference

Dies ist der Wurzelknoten des Java Workspace

Die einzelnen Java-Projekte sind als Maven Multi Module Projekte organisiert.

### conference-base

Dieses Projekt beinhaltet querschnittliche Funktionen:

#### persistence.xml

Da alle Serviceprojekte letztlich auf die gleiche DB zugreifen und die Tabellen teilweise per FK miteinander verbunden sind, wird nur eine querschnittliche persistence.xml verwendet.

#### Producer

Der Producer für Logger, QueueConnection, InitialContext und EntityManager wird querschnittlich bereitgestellt.

#### PerformanceLoggingInterceptor

Loggt Service Aufruf Laufzeiten und Erfolgsquoten zur Ermittlung von Performanceproblemen

#### ServiceLoggingInterceptor

Schreibt übersichtliche Logeinträge zu Serviceaufrufen

### conference-room

Implementierung des RoomCRUDService und dessen Entität Room.

Keine Beziehungen zu anderen Services. So wird sichergestellt, dass Rooms im Objektgeflecht ein „Blatt“ bleiben und keine unnötigen Zirkulärbeziehungen entstehen

### conference-speaker

Implementierung des SpeakerCRUDService und dessen Entität Speaker.

Keine Beziehungen zu anderen Services. So wird sichergestellt, dass Speaker im Objektgeflecht ein „Blatt“ bleiben und keine unnötigen Zirkulärbeziehungen entstehen

### conference-conference

Beinhaltet die Implementierungen von ConferenceCRUDService sowie der Conference Entität.

Keine Beziehungen zu anderen Services. So wird sichergestellt, dass Rooms im Objektgeflecht ein „Blatt“ bleiben und keine unnötigen Zirkulärbeziehungen entstehen

### conference-talk

Beinhaltet die Implementierungen von TalkCRUDService sowie der Talk Entität.

Diese Komponente hat Abhängigkeiten zu conference-speaker-api, conference-room-api und conference-conference-api

Die n:m Zuordnung zwischen Talk und Speaker wird über die Hilfsentität TalkSpeakerAssignment abgebildet. Diese ist in conference-talk-impl abgelegt und somit nicht Teil des conference-talk-api jars, welches diese Entität ansonsten exponieren würde.

Ebenfalls Teil dieser Komponente ist ein Decorator des TalkServices (TalkChangeNotificationDecorator), welcher schreibende Zugriffe wrappt und eine Nachricht über die Änderung an dem jeweiligen Talk in eine Queue schreibt.

### conference-facade

Diese Komponente beinhaltet die wesentliche Businesslogik (Validierungen), welche den Zugriff auf unterschiedliche Entitäten bedingen. Z.B. „kennt“ die Entity Conference keine Talks, jedoch müssen Änderungen am Datum der Conference gegen eventuell zugeordnete Talks validiert werden. Mangels Kenntnis von den Talks kann dies nicht auf der Ebene des ConferenceCRUDService geschehen.

Alle Zugriffe der web und rest Komponenten auf das Backend erfolgen über diese Komponente.

### conference-web

Diese Komponente beinhaltet die JSF GUI.

Die Security ist deklarativ in der web.xml abgebildet.

### conference-rest

Diese Komponente beinhaltet die REST Services

Die Security ist deklarativ in der web.xml abgebildet.

## doc

Dokumentation. Enthält dieses Dokument

## soapui

Enthält ein zugehöriges SoapUi Projekt zum Testen der Rest Services

## sql scripts

Enthält SQL DDL Skripte zum Anlegen der Tabellen, Testdaten und Benutzer

# Standard für Coding

* Es wird der Standard-Codeformatter von Eclipse Kepler WTP verwendet.
* Codesprache ist Englisch
* Methodenparameter sind minimal zu halten. Wenn statt einer Entität dessen id ausreicht, dann wird auch nur nach einer id verlangt.
* Abgeleitete Attribute, also Attribute welche zur Laufzeit aus anderen Attributen ermittelt werden, sind in Entitäten sind erlaubt. Deren Getter beginnen mit dem Prefix .build???() statt get???()

## Namenstandards

## Projekte

Einzelne Komponenten sind jeweils in einem eigenen Maven Modul untergebracht, welches sich wiederum in weitere Untermodule mit öffentlichem (API) und privatem (Implementierung) Teil gliedern.

Das API Modul ended jeweils auf –api, die Implementierung auf -impl

Die Module sind jeweils entsprechend ihrem vollen Namen/Funktion separiert mit Minus benannt.

Also z.B: Das Modul conference-talk beinhaltet conference-talk-api und conference-talk-impl

## Packages

Ausgangspackage ist “com.prodyna.academy.pac.conference”. Dieses wird abhängig von der jeweiligen Komponente weiter verfeinert. Z.B. “com.prodyna.academy.pac.conference.talk” für die conference-talk Komponente.

Entitäten liegen von dort aus wiederum im Package model: “com.prodyna.academy.pac.conference.talk.model”

Services liegen im Package service: “com.prodyna.academy.pac.conference.talk.service”

Weitere Werkzeuge sind in util unterzubringen: “com.prodyna.academy.pac.conference.talk.util”

## Interfaces, Klassen & Services

Namenstandard: Es wird ein gewöhnlicher fachlicher Klassenname verwendet, kein „I-Präfix“

Services enden auf ?????Service, z.B. „TalkService“ .

Elementare Services zur Datenmanipulation tragen CRUD im Namen, z.B. RoomCRUDService.

Die Implementierung von Services ended auf –Impl und beginnt mit dem Namen des Interfaces.

Alternative Implementierungen können durch einen Einschub zwischen diesen beiden Namenskomponenten differenziert werden

Z.B. “TalkServiceImpl”, “TalkServiceBlaBlaBlubImpl”

Namensstandard für lesende Services: get???()

Namensstandard für schreibende Services: save??? (), create??? (), delete??? ()

# Exceptions

Es werden unchecked Exceptions verwendet.

Für technische Probleme stehen durch Java bereits verschiedene Ausprägungen von RuntimeExceptions zur Verfügung; falls nötig wird in RuntimeException gewrappt und weitergeworfen.

Für fachlich gültige Fehlersituationen gibt es die unchecked Exception „BusinessException“, bei Bedarf auch als Oberklasse. Durch die Verwendung dieser Exception oder einer Ableitung davon werden fachliche Exceptions für das Monitoring kenntlich gemacht.

Namensstandard: Der Name von Exceptions endet mit „…Exception“