Conference Applikation - Service Architektur

# Versionshistorie

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Datum | Author | Beschreibung |
| 1.0 | 14.11.2013 | Christian Predikant | Initiale Erstellung |
| 1.1 | 18.11.2013 | Christian Predikant | Erste Vervollständigungen |
|  |  |  |  |
|  |  |  |  |

Inhaltsverzeichnis

[Versionshistorie 2](#_Toc372578325)

[Übersicht und Ziel 4](#_Toc372578326)

[Architektur der Applikation 4](#_Toc372578327)

[Schicht 1 – Persistenz 4](#_Toc372578328)

[Schicht 2 - Business-Logik 5](#_Toc372578329)

[Schicht 3 – Web 5](#_Toc372578330)

[Entwicklungsleitfaden 5](#_Toc372578331)

[Projektaufbau 5](#_Toc372578332)

[Conference-common 5](#_Toc372578333)

[Coneference-ear 5](#_Toc372578334)

[Conference-jsf-web 6](#_Toc372578335)

[Conference-rest-api 6](#_Toc372578336)

[Conference-rest-web 6](#_Toc372578337)

[Conference-service-api 6](#_Toc372578338)

[Conference-service-impl 6](#_Toc372578339)

[Conference-assembly 6](#_Toc372578340)

[Namenskonventionen 6](#_Toc372578341)

[Codingstandards 7](#_Toc372578342)

[Technologiestack 7](#_Toc372578343)

[Anhang A – Anforderungen 7](#_Toc372578344)

[Einleitung 7](#_Toc372578345)

[Anforderungen 8](#_Toc372578346)

[Hinweise 9](#_Toc372578347)

[Entscheidungen 9](#_Toc372578348)

[Deliverables 9](#_Toc372578349)

# Übersicht und Ziel

Dieses Projekt hat zum Ziel eine Conference-Applikation zur erstellen. Diese soll Coference-Besuchern ermöglichen sich über die Conference, die Talks, die Rooms und die Speaker zu informieren. Weiter soll es den Administratoren möglich sein diese Einheiten zu Pflegen.

Die Funktionalen und nicht Funktionalen Anforderungen, für die Applikation, finden sich im Anhang. Die Herausforderung an die Architektur, war es diese Anforderungen möglichst einfach zu realisieren, um dem Kunden später die Wartung und Pflege zu erleichtern.

# Architektur der Applikation

Die Applikation ist auf dem JEE 6 Stack entwickelt, den der JBoss Applicationserver in Version 7 bereitgestellt. Zu den Schlüsseltechnologien, die verwendet werden, gehören:

* Enterprise Java Beans 3.1
* JPA 2 mit Hibernate 4 als ORM-Provider
* JSF 2.1
* REST-API 1.1
* JMX
* JMS
* CDI

Die oben genannten Technologien stellen den Stack für die Entwicklung dar. Für die Build- und Dependency-Managemet wird Maven 3 verwendet, das SourceCode-Management wird über Git abgebildet und als Datenbank kommt MySQL zum Einsatz. Eine weitere wichtige Technologie ist Arquillian. Diese wird für das Testen aller entscheidenden Teile der Software verwendet und ermöglicht Remote-Testing auf dem Applicationserver.

Grob kann die Architektur in drei Schichten eingeteilt werden. Folgende Abbildung stellt dies schematisch da und ordnet die oben genannten Technologien den einzelnen Schichten zu.

Hier kommt eine Zeichnung hin.

Die Folgenden Abschnitte gehen auf die einzelnen Schichten und ihre Besonderheiten ein.

## Schicht 1 – Persistenz

Die Daten der Applikation werden in einer MySQL-Datenbank persistiert. Um den Zugriff auf die Relationalen Daten und das Übersetzen in die Objektorientierte Welf von Java zu erleichtern wird JPA 2 mit Hibernate 4 als Objekt-Relationales Mapping-Framework verwendet. Die Tabellen bzw. werden über JPA 2 Entities abgebildet. Die JPQL-Statements werden als NamedQueries an den Entitäten annotiert. Dies hat den Vorteil, dass deren Korrektheit (Syntaktisch) zur Deploymentzeit geprüft wird und Fehler frühzeitig erkannt. Im Hinblick auf die geplante REST-Schnittstelle werden M-N-Relationen über eine eigene Mappingtabelle abgebildet. Diese besitzt dann zwei ManyToOne Relationen zu den zu verbindenden Entitäten. Generell wird auf Relationen verzichtet, die eine Persistence Collection produzieren. Diese müssten Eager geladen werden, um keine Probleme in der REST-Schnittstelle zu machen. Hier zur Übersicht der UML-Entitäten Model:

UML-Entitäten Model

## Schicht 2 - Business-Logik

Die Businesslogik wird via Enterprise-Java-Beans (EJB). Die Businesslogik ist Stateless umgesetzt. Die notwendigen Ressourcen wie Logger, Entitymanger etc. werden über CDI-Producer-Methoden in die Services Injected. CDI spielte gerade in diesem Teil der Applikation eine tragende Rolle. CDI ermöglichte es elegant, die Anforderungen für Logging und Performance-Messung, sowie das Ansprechen der JMS-Queue zu erfüllen. Hierfür werden CDI Interceptoren (Aspektorientiertes Programmieren) und CDI-Decoratoren (Wrappen von Klassen) verwedet.

TO BE CONTINUED…

## Schicht 3 – Web

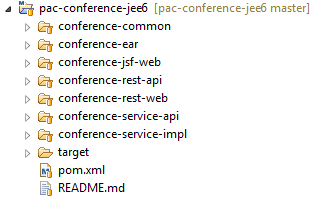
In dieser Schicht wird die Business-Logik verwendet und zur Anzeigen gebracht bzw. für den Zugriff via HTTP bereitgestellt. Dies geschieht einerseits per JSF-2.1 und auf der anderen Seite per JAX-RS 1.1 und JSON.

# Entwicklungsleitfaden

In den folgenden Abschnitten wird beschrieben, welche Namenskonventionen und welche Codingstandards gelten. Weiter wird der Aufbau des Projekts erklärt und welcher Technologie verwendet wird.

## Projektaufbau

Grundsätzlich ist zu erwähnen, dass als Build und Dependency Management Tool Maven 3 verwendet wird und es sich beim Projekt um ein Maven-Multiprojekt handelt. Der Projektaufbau wird durch den Folgenden Screenshot dargestellt:



Das Root-Projekt ist „pac-conference-jee6“ und beinhaltet alle Globalen Einstellungen für das Projekt und alle Module innerhalb dieses Projekts. Im Folgenden werden die einzelnen Module beschrieben.

### Conference-common

In diesem Modul werden Modulübergreifende querschnittliche Klassen und Utilities platziert. Dieses Modul ist ein einfaches JAR.

### Coneference-ear

Dieses Modul ist, wie der Name andeutet das Enterprise-Archive, welches alle anderen Module zusammen fasst und für das Deployment auf einen JEE –Server bereitstellt. Es bildet zugleich das Release-Unit des Projekts. Wird später ein zentrales Pesistence-Unit benötigt, würde sie hier platzieren werden.

### Conference-jsf-web

In diesem Modul ist die JSF-Webanwendung untergebracht. Es beinhaltet alle für die JSF-Webanwendung benötigten XHTML-Seiten, Konfigurationen, Controller-Klassen und Ressourcen. Er wird als WAR-File bereitgestellt.

### Conference-rest-api

Dieses JAR-Modul beinhaltet die Interfaces für die REST-Schnittstelle der Applikation. Die Schnittstelle ist in „Secure“ und „Unsecure“ Teile zerlegt worden. Damit ist es möglich die Teile der Schnittstelle abzusichern, welche nicht öffentlich sein sollen. Dies betrifft all Create, Delete und Update Funktionen.

### Conference-rest-web

Dieses Modul enthält die Implementierung der REST-Services ist wie die API in „Secure“ und „Unsecure“ unterteilt. Es wird als WAR-bereitgestellt und läuft völlig eigenständig neben der JSF-Webanwendung.

### Conference-service-api

Dieses Modul ist die API für eine Fachliche Domäne. Im konkreten Fall ist es die Domäne Conference. Die API enthält die benötigten Entities, Interfaces und Exceptions der Domäne. Dieses Modul wird als JAR zur Verfügung gestellt.

### Conference-service-impl

Dieses Modul ist die Implementierung der fachlichen Domäne und enthält neben den Service-Implementierungen die Interceptoren, Decorator, Monitoring und Messaging Komponenten. Ebenso besitzt das Modul das verwendete Persistence-Unit.

### Conference-assembly

Das Conference-Assembly Modul erstellt das Assembly für die Applikation und Enthält:

* Das Ear-File der Applikation
* Die Sourcen des Releases
* Die JavaDoc des Releases
* SQL für CREATE und UPDATE

## Namenskonventionen

Für die Entwicklung gelten folgende Namenskonventionen und Regeln:

* Das Basis-Package der Applikation ist „com.prodyna.pac.conference“
* Innerhalb der Module werden die Klassen technisch gruppiert. Und mit dem Infix des Moduls versehen. Dies führt zu folgenden Package-strukturen

|  |  |
| --- | --- |
| Komponente | Package |
| Interceptor + Binding | com.prodyna.pac.conference.{infix}.interceptor.CamelCaseClasses |
| Decorator | com.prodyna.pac.conference.{infix}.decorator.CamelCaseClasses |
| JMX | com.prodyna.pac.conference.{infix}.jmx.CamelCaseClasses |
| JMS | com.prodyna.pac.conference.{infix}.jmx.CamelCaseClasses |
| Domäne-Impl | com.prodyna.pac.conference.{infix}.CamelCaseClasses |
| Domäne-API | com.prodyna.pac.conference.{infix}.api.CamelCaseClasses |
| Exceptions | com.prodyna.pac.conference.{infix}.exception.CamelCaseClasses |
| Entities | com.prodyna.pac.conference.{infix}.model.CamelCaseClasses |
| Util | com.prodyna.pac.conference.{infix}.util.CamelCaseClasses |

## Codingstandards

Als Entwicklungsumgebung ist Eclipse zuverwenden, welche die JBOSS-Tools enthält oder das vorgefertigte Eclipse JBOSS-Tools von JBOSS. Dieses korrespondiert am besten mit dem JBOSS-AS-7 und Maven.

Für die Implementierung ist zwingend erforderlich, dass das Encoding der Entwicklungsumgebung auf UTF-8 eingestellt ist. Für die Formatierung der Quellcodes wird der Eclipse-Standard verwendet.

Das mindeste an Code-Dokumentation, was vorhanden sein soll ist JavaDoc der Klasse und JavaDoc aller Interface-Methoden. Klassen-Member müssen nicht unbedingt dokumentiert werden.

## Technologiestack

|  |  |
| --- | --- |
| Technologie | Version |
| Maven | 3.1.1 |
| JBoss AS | 7.1.1Flnal |
| MYSQL | 5.5.x |
| Java | 1.6 |
| PrimeFaces | 4.0 |
| JBoss.BOM | 1.0.7.CR8 |
| Davon abweichend: |  |
| Javax.faces | 2.1.26 |
| Myfaces-COODI | 1.0.5 |

# Anhang A – Anforderungen

Folgender Text beschreibt die Anforderungen für die Conference AppliKation

## Einleitung

Es soll eine Applikation für Konferenzen programmiert werden. Die Hauptanwender sind Gäste, die diese Applikation über das Web oder über mobile Endgeräte nutzen sollen. Eine Konferenz (**Conference**) hat eine Dauer von einem bis mehrere Tage und findet üblicherweise an einem Ort statt. Beispiele für Konferenzen sind W-JAX oder die PRODYNA Hausmesse. Auf einer Konferenz findet Vorträge (Talks) statt. Die Gäste sollen sich informieren können, welche Talks überhaupt existieren, wann sie sind, in welchem Raum (**Room**) sie stattfinden und von wem (**Speaker**) dieser Talk gehalten wird. Ein Talk kann von mehreren Speakern gehalten werden. Zusätzlich soll es ein Backoffice-Zugang geben, über welche die Stammdaten (als Conference, Speaker, Room, Talk) gepflegt werden können.

* Eine „**Conference**“ hat einen Namen, eine Kurzbeschreibung, ein Anfangs- und ein Enddatum.
* Ein „**Speaker“** hat einen Namen und eine Kurzbeschreibung
* Ein „**Room“** hat einen Namen (z.B. „Adria“) und eine Kapazität
* Ein „**Talk**“ hat einen Namen, eine Kurzbeschreibung, eine Länge (in Minuten), einen „**Room**“ und einen oder mehrere „**Speaker“** und gehört zu einer Conference.

## Anforderungen

Der Auftraggeber stellt die folgenden Anforderungen:

* Als Endanwender erwarte ich die folgenden Sichten in der Applikation
  + Startseite: Liste aller Konferenzen
  + Konferenz: Liste der Tage mit der Anzahl der Talks pro Tag
  + Talk: Details und Verweise auf Room und Speaker
  + Speaker: Details zu Verweise zu Talks
  + Room: Liste aller Talks, die in diesem Room stattfinden, nach Tagen getrennt, zeitlich sortiert
  + Konferenzübersicht: Darstellung einer Tabelle mit allen Tagen und aller Talks an diesem Tag.
* Als Backoffice möchte ich die folgenden Stammdaten pflegen
  + Conferences (CRUD)
  + Speaker (CRUD)
  + Rooms (CRUD)
  + Talks (CRUD), damit müssen aber Konflikte bemerkt und angezeigt werden
    - Speaker können nur einen Talk gleichzeitig halten
    - Ein Room kann nur von einem Talk gleichzeitig genutzt werden
    - Ein Talk kann nur innerhalb der Conference (zeitlich) stattfinden.
* Als Kunde möchte ich, dass ein Dokument "Service Architecture" verfasst wird, welches folgende Standard beschreibt
  + Verwendete Technologien
  + Aufbau der Quellen im Repository
  + Standard für Coding
  + Namenstandards für Projekte, Packages, Interfaces, Services, Klassen, Exceptions
* Als Kunde möchte ich, dass für das Backend Java EE >=6 (Session Beans, JPA) auf einem JBoss 7 verwendet wird, weil das Kundenstandard ist
  + Andere Application Server sind auch ok, z.B. Glassfish
* Als Kunde möchte ich, dass das Backend mit Hilfe von Arquillian getestet wird, und zwar so, dass jede Geschäftlogik abgedeckt ist, weil ich eine hohe Qualität erwarte
* Als Kunde möchte ich, dass für das Frontend JSF auf einem JBoss 7 verwendet wird, weil das Kundenstandard ist.
  + Ich akzeptiere andere Web-Frameworks, solange der Architekt diese verantworten kann.
* Als Kunde möchte ich, dass die Geschäftslogik als REST-Interface zur Verfügung gestellt wird, damit ich das System besser an meine vorhandenen Systeme integrieren kann.
* Als Kunde möchte ich, dass die einzelnen Komponenten möglichst feingranular geschnitten werden und die Sichtbarkeiten auf ein Minimum reduziert werden, weil ich wenig erfahrene Entwickler habe und diese Software von meinen Leuten weiterentwickelt werden wird
* Als Kunde möchte ich, dass Maven als Build- und Dependancy Management verwendet wird, weil das Kundenstandard ist
* Als Kunde möchte ich, dass beim Build alle Tests automatisch ausgeführt werden, weil ich später alles auf einem Jenkins/Travis-CI laufen lassen möchte
* Als Kunde möchte ich, dass GitHub (public) als Repository verwendet wird, weil das Kundenstandard ist
* Als Kunde möchte ich, dass MySQL als Datenbank verwendet wird, weil das Kundenstandard ist.
  + Als Alternative akzeptiere ich MongoDB, weil das zukunftsträchtig ist.
* Als Kunde möchte ich, dass wenn sich am Talk etwas ändert (z.B. anderer Raum) eine asynchrone Nachricht in eine Queue mit diesen Informationen geschrieben wird, damit ich eine Push-Notification/SMS verschicken kann
  + Als vorübergehende Lösung erwarte ich eine MDB, die diese Informationen ins Log schreibt
  + Es ist wichtig, dass diese Komponente gut getestet ist
* Als Ops möchte ich die Antwortzeiten der Applikation auf Methodenebene überwachen können, damit ich Performanceprobleme schnell orten und beheben kann.
  + Ich erwarte auf Service und Methodenebene Informationen über die Laufzeiten (Min, Max, Anzahl, Summe der Zeit, Durschnitt)
  + Der Zugriff auf diese Daten muss per JMX erfolgen können
* Als Ops möchte ich, dass alle Datenbankzugriffe als Prepared Statements ausgeführt werden, damit die Belastung der Datenbank geringer ist.
* Als Ops möchte ich, dass zu allen Service-Aufrufen ein Log-Eintrag erfolgt, wo man den Service, die Methode, die Parameter und die Antwortzeiten sehen kann
* Als Ops möchte ich, dass die Applikation nicht direkt die Datenbanktabellen anlegt oder manipuliert, sondern ein SQL-Script bereit liegt, weil bei uns Applikationen keine DDL-Logik enthalten dürfen

## Hinweise

* Wir verzichten auf Security, wer sowas implementiert erhält einen Bonus
* Die Qualität der GUI (Ergonomie) spielt keine Rolle, es geht nur um Funktionalität

## Entscheidungen

Als Architekt des Projekts müssen sie diverse Entscheidungen treffen

* Wie viele Release Units soll es geben?
* Welche Komponenten wird es geben?
* Wie soll die Maven-Struktur aussehen?
* Welche Datenbank soll verwendet werden (MySQL oder MongoDB)
* Welche Standards sollen gelten

## Deliverables

* Ein Dokument "Service Architekture"
* Ein oder mehrere git-Repositories mit jeweils eine Maven-Projekt
* Das Build des Maven-Projekts liefert ein Assembly, das alles enthält, um die Applikation zum Laufen zu bringen