In unserem aktuellen Projekt ist die Kundenvorgabe, die Implementierung mit JEE-Standards vorzunehmen.
Es handelt sich um einen Microservice, welcher „stateless“ Informationen aus einer relationalen Datenbank und einem nachgelagerten REST-Service zusammenstellt.
Als Plattform dient JBoss 6.3. Laut Spezifikation unterstützt dieser die JEE 6 vollständig.
Die Projektanforderungen enthalten zunächst keine Besonderheiten, so dass das Projekt zuversichtlich mit diesen Rahmenbedingungen begonnen wird.
Während der Implementierung ergeben sich allerdings verschiedene Notwendigkeiten zur Optimierung:

- Asynchroner REST-Zugriff (JAX-RS 2.0):

Beim Zugriff auf den nachgelagerten REST-Service stellt sich heraus, dass dieser ein Bottleneck darstellt. Hier muss also ein Cache aufgebaut werden.
Der aus dem REST-Service abgerufene Wert, eine Logo-URL, ist für die Grundfunktionalität verzichtbar, so dass ein asynchroner Zugriff für den Cache-Aufbau verwendet werden kann.
Der erste Aufruf geht also auf einen Standard-Wert, nachfolgende Aufrufe können auf den mittlerweile aufgebauten Cache zugreifen.
JEE 7 ermöglicht asynchrone REST-Aufrufe. Durch Konfiguration des JBoss kann diese JEE7-Funktionalität bereit gestellt werden.
(https://dzone.com/articles/jax-rs-20-asynchronous-server-and-client)

- Löschen eines Objektes

Beim Löschen eines Objektes aus der DB stellt sich heraus, dass der remove()-Aufruf nicht greift, da das Objekt nicht mehr in einem DB-Session-Kontext verankert ist.
JPA hat hier deutliche Einschänkungen gegenüber Hibernate (s.a. https://stackoverflow.com/questions/912659/what-is-the-proper-way-to-re-attach-detached-objects-in-hibernate).
Abhilfe: JPQL-Query. Das ist nach JPA 1.x möglich, führt aber an dem gesamten Konzept des O/R-Mappings vorbei, da SQL (-ähnlicher) Code im Quellcode auftaucht.

- Löschen von Attributen aus der JSON-Response

Das explizite Parsen von JSON mit dem Jackson Object Mapper aus ist im Projekt gar kein Problem, Annotations wie @JsonIgnore werden problemlos verarbeitet.
Beim der deklarativen Verwendung von JAX-RS wird das Rückgabeobjekt zwar zunächst einwandfrei als JSON zurückgegeben – aber die Annotations werden ignoriert.
Nach einer mehrstündigen Debug-Session

(s.a. http://docs.jboss.org/resteasy/docs/2.3.7.Final/userguide/html_single/index.html)
stellt sich heraus, dass JBoss intern Jackson 1 und nicht Jackson 2 verwendet. Dies lässt die Wahl, entweder a.) alle JSON-Annotations auf Jackson 1 zurückzudrehen, b.) jeweils beide Annotations zu verwenden, c.) Den JBoss mit ca. 10-20 Zeilen so zu konfigurieren, dass Jackson 2 als Mapper verwendet wird oder d.) mit 3 Zeilen die Konvertierung Objekt -> JSON-String explizit beim Aufruf auszuprogrammieren. Um weitere Stunden der Konfiguration zu sparen, sind wir bei Option d.) gelandet, nicht schön, aber aus meiner Sicht angemessen, um nicht weitere Stunden zu investieren, um 3 Zeilen Code einzusparen.

Heraus kommt also ein Projekt, welches sehr spezifisch auf die Zielplattform ausprogrammiert wird. Der Vorteil ist, dass alle Anpassungen durchaus im JEE-Standard lauffähig sind.
Aber bereits der fehlende Standard für die JSON-Annotations in JEE 7(!) führt dazu, dass eine Abhängigkeit in eine konkretes Framework in das Projekt hineinwächst.
Dies gilt aber generell bei der Umsetzung von JEE-Projekten: Der Standard hängt mehrere Jahre hinterher. Getreu der 80/20 Regel stößt man im Projekt aber immer wieder auf Aufgaben, die mit aktuellen Frameworks schnell und unkompliziert zu lösen sind. Hier muss sich das Team entscheiden:

I. Werden Abhängigkeiten auf spezifische Versionen eines Frameworks in Kauf genommen?
II. werden die Möglichkeiten des Application Servers genutzt?
III. Oder wird die nötige Funktionalität des Frameworks nachprogrammiert.

Vor dieser Entscheidung stehen bereits mittelgroße Projekte, und die saubere Lösung III. wird aus nachvollziehbaren Gründen sehr selten in Erwägung gezogen.
Damit ist die lupenreine Verwendung des JEE-Standards für „real-life“ Projekte nur eine Illusion. Möchte man sich nicht an einen Hersteller binden, so ist I. der Weg für eine halbwegs zukunftssichere Lösung.

Ist die Entscheidung I. getroffen, aktuelle Frameworks im Projekt zu verwenden, ist der Weg zu Spring und Hibernate in der aktuellen Version nicht mehr weit.
Damit verwendet man die Vorreiter der nächsten und übernächsten JEE-Version bereits heute.
Mit dem aktuellen Trend zu Microservices wird die Gefahr von Versionskonflikten außerdem „by design“ noch vermindert.
JEE bleibt damit interessant für weniger komplexe Aufgaben, die maximal portierbar bleiben sollen.
Sobald aber agile Projekte mit unbekannter Komplexität angegangen werden, ist meiner Erfahrung nach das Risiko größer, die Nachteile beider Welten im Projekt zu vereinen.

Über den Autor

Wolff Holtmann

Für Ihre Fragen zu dem Thema stehen wir gerne zur Verfügung. Sie erreichen Herrn Wolff Holtmann per E-Mail:

E-Mail: wolff.holtmann@consiness.com